V1.1(NEW)

Implementing the CIDOC Conceptual Reference Model in RDF

Version 1.1
Martin Doerr, Richard Light, Gerald Hiebel
October 23, 2020.

Introduction

The CIDOC Conceptual Reference Model (the "CRM") [1] is defined in terms of an object-oriented metamodel[2]. It is an abstract, logical expression of the concepts and relationships (or classes and properties) which are relevant to cultural heritage documentation. In its native form it can be used to analyse and compare different standards and systems. For this purpose, there is no need to be concerned with implementation details, which vary between encoding paradigms and over time. Since the first conception of the CIDOC CRM, object-oriented and the related knowledge representation models have changed frequently: OMG standards, KL-ONE, TELOS, KIF, DAML, OIL, to name just a few. Therefore, these are typically not specified in the Definition of the CRM.

The CIDOC CRM however has been promoted as ISO standard when RDF became a W3C recommendation in 1998, the first knowledge representation model to acquire a status of an international recommendation, which, by the way, did not deviate substantially from the one used to define the CIDOC CRM. Nevertheless, the CIDOC CRM has not been defined using RDF Schema itself, because it aims at providing the widest possible logical basis for comparing and integrating data implemented under different encoding paradigms, seeking a compromise between useful expressive power for the ontology and enough simplicity to be implementable in different relevant encoding paradigms, either directly by their built-in constructs or requiring some additional S/W code. In the meanwhile, RDF is also undergoing its own evolution becoming refined and extended by OWL. Therefore the CIDOC CRM maintains a logical form which is independent from RDF.

Characteristically, the original CRM object-oriented model uses constructs, such as 'properties of properties', which cannot be expressed in RDF, and a set of logical deductions (such as the "shortcuts"), which can be implemented with additional code or by the standard query system in relevant current database or knowledge base types.

Another serious reason for keeping the logical independence lies in a much more subtle, but fundamental difference between describing things in the world in terms of logic and describing data by a data definition and encoding language, which becomes most apparent when distinguishing between real world things and their identifiers and when describing mathematical value spaces, which is necessarily common to all data encoding paradigms, including RDF. The logical-ontological compatibility of data implemented in different data

definition and encoding languages cannot be defined based on a data definition or encoding language.
RDF is currently the knowledge encoding paradigm closest to the CIDOC CRM and the most widely spread. Therefore, for each official release of the CRM, an RDF Schema expression of the CRM will be published as official resource on the CIDOC CRM web site[3]. However, on its own this Schema does not contain sufficient information to allow potential implementers to design a complete CRM-compatible framework in RDF, as indicated above. Also, particular global identification conventions of CRM RDF implementations of different CRM versions need explanation.
The aim of this document is to complement the Definition of the CRM and the respective RDF Schema implementation by providing guidance on recommended techniques for using the CIDOC CRM within an RDF implementation, for instance running on an RDF-enabled Triple Store or Graph database.

The CRM RDF Schema

The CRM RDF Schema defines most[4] of the classes and properties which make up the CRM, and specifies the domain and range of these. Furthermore, the URLs specified in the Schema (with base http://www.cidoc-crm.org/cidoc-crm/) act as valid dereferenceable Linked Data identifiers. So you can define a CRM prefix:

@PREFIX crm: <http://www.cidoc-crm.org/cidoc-crm/>

and then use CRM identifiers (e.g. “crm:P53_Place”) freely within your data.
The following design decisions have been taken when designing the CRM RDF Schema. (These will appear as a comment at the start of the CRM RDF Schema document)

The Equivalent Part

Here we describe the part of the CRM RDF Schema that corresponds one-to-one to the Definition of the CIDOC CRM.
1. The RDF class and property naming rules do not allow “space” characters. Hence the “space” character is replaced in RDF names by the “underscore” (spacing underscore, low line, horizontal bar) character. For instance "E63 Beginning of Existence" becomes in RDF "E63_Beginning_of_Existence" or "P2i is type of" becomes in RDF "P2i_is_type_of".
2. RDF does not allow one to instantiate bi-directional properties. Therefore, each CRM property is represented as two RDFS properties, each with a specific “direction”. For instance "P2 has type (is type of)" is represented as:

- "P2_has_type" for the domain to range direction
- "P2i_is_type_of" for the range to domain direction

The only logical distinction between the two directions is the fact of inverse reading. In the recommended OWL version[5], this is explicated by declaring one direction as “inverse_of” the other. Practically, this means that an implementation may decide to replace one direction with the opposite via software, exchanging domain and range accordingly, or add or remove one direction for convenience without affecting compatibility or meaning. This can be useful for optimizing display, storage or querying.
3. Scope notes within the Schema are represented as <rdfs:comment> elements.

4. The encoding contains labels in languages other than English, which are taken from the latest translations of current or previous versions of the CIDOC CRM.
5. Any other differences in labels, scope notes and semantic relationships of this encoding from the respective authoritative definition of the CIDOC CRM are unintended transfer errors and not alternative definitions. The authoritative reference is the textual definition of the CIDOC CRM and not the RDF Schema. FORTH appreciates your feedback on such errors.
6. RDF does not support properties of properties. Therefore, users are recommended two ways to work around:
The current properties of properties in the CRM have all as range “E55 Type”. Therefore they correspond to subtyping of the respective property by a local vocabulary.
For the cases in which the local vocabulary is not fixed, there is a recommended form of reification via an auxiliary “property class”. This replaces the former recommendation to use E13 Attribute assignment in order to introduce user defined property types.
See section below on properties of properties about the pros and cons.

Implementing Datatypes

Among the entities information systems can refer to, one can distinguish those that cannot reside in an information system itself because of their nature, such as people, material objects, weather etc. and their interactions, from those that can have exhaustive representations in an information system, such as texts, numbers etc., because they are generically digital in nature[6]. Those that cannot reside in the information system, we can only refer to by identifiers. For those that can exhaustively be represented, all data models offer some elementary “built-in” datatypes, such as “integer”, “char”, “string”, “real”. They are recognized syntactically, and not by classification, and their identity is given by their “content” and their syntactic type. Consequently any occurrence of the same content is identical, regardless the context of reference[7].

Therefore, the CIDOC CRM, being an ontology and not a data model, refers to them in an abstract way as instances of “E59 Primitive Value”. To which degree they are actually “primitive” or elementary, or composed of more elementary ones, such as dates, is not the critical characteristic, but the fact, that their identity is completely defined by the syntactic type and digital content. In purpose, they match exactly with rdfs:Datatype.
The class E59 Primitive Value “comprises values of primitive data types of programming languages or database management systems and datatypes composed of such values used as documentation elements, as well as their mathematical abstractions”.
They are not considered as elements of the universe of discourse this model aims at defining and analysing. Rather, they play the role of a symbolic interface between the scope of this model and the world of mathematical and computational manipulations and the symbolic objects they define and handle.
In particular they comprise lexical forms encoded as "strings" or series of characters and symbols based on encoding schemes (characterised by being a limited subset of the respective mathematical abstractions) such as UNICODE and values of datatypes that can be encoded in a lexical form, including quantitative specifications of time-spans and geometry. They have in common that instances of E59 Primitive Value define themselves by virtue of their encoded value, regardless the nature of their mathematical abstractions.

Therefore they should not be represented in an implementation by a universal identifier associated with a content model of different identity. In a concrete application, it is recommended that the primitive value system from a chosen implementation platform and/or data definition language be used to substitute for this class and its subclasses"[8].

For encoding the CIDOC CRM in RDF this translates to trying to find suitable RDF datatypes, to the degree there is an equivalence with respective subclasses of E59 Primitive Value, or at least finding datatypes that represent wide enough value ranges for typical applications. It must however be understood that datatypes typically implement only subsets of values of respective mathematical spaces, whereas an ontological definition refers to the mathematical space itself. For instance, an "integer" value may be limited to 32 bits, which is "nearly nothing" against the unlimited size of natural numbers, but much more than most applications will ever encounter in object descriptions. Nevertheless, data referring to values outside of a particular datatype, but inside the respective mathematical space, must be regarded compatible with the CRM.

In principle, the literal encoding of mathematical values is unlimited, if the respective platform can represent unlimited literals. Consequently, rdfs:Literal is the superclass of all RDFS datatypes ("Each instance of rdfs:Datatype is a subclass of rdfs:Literal." in: RDF Schema 1.1). Therefore, all properties of the CRM having a primitive value as range are compatible with using rdfs:Literal or an adequate datatype, as long as the meaning is compatible. Since this does not provide a particular guidance how to encode values nor any more formal constraints,we analyze in this document separately each subclass of E59 Primitive Value and the use of rdfs:label for their compatibility with RDFS datatypes and make **detailed recommendations**.
Nevertheless, applications may encounter cases in which no datatype recommended by RDFS or recommended below does fit the required value range. In that case it is recommended to find **other standards** to represent these values in an XSD-compatible form and to store them in an rdfs:Literal. More details are also given in the section "Defining custom datatypes".

Recording dates

The range of the properties "P81 ongoing throughout" and "P82 at some time within" are defined in the CRM as E61 Time Primitive. Instances of E61 Time Primitive are defined as closed intervals on the natural time dimension in which we live.
Since the E61 Time Primitive of the CRM cannot be expressed in RDFS directly, in the official RDF implementation of the CIDOC CRM, we define four properties in the CRM RDFS: "P82a_begin_of_the_begin", "P82b_end_of_the_end", "P81a_end_of_the_begin", "P81b_begin_of_the_end", all with range **xsd:dateTime**, which replace "P81 ongoing throughout" and "P82 at some time within" of the CRM. For more details about the meaning of these four properties, see the guidelines in the Annex below.
Extremely old paleontological material and astronomic dates can be **below the range** of xsd:dateTime. If such dates need to be recorded, we recommend to discuss an extension with CRM-SIG.
All other RDF datatypes for time with more limited precision, such as years only (xsd:gYear), should not be used, because their interpretation either as duration or interval of

indeterminacy causes significant implementation overhead at query time, whereas the properties P82a,P81a,P81b,P82b can express the same information unambiguously. Notwithstanding, data entry forms may offer any simplification for specifying dates they want, and convert internally the representation into the recommended form. Ease of data entry is therefore no argument for the choice of a datatype.

Recording space

The recommended datatypes of RDF1.1 do not contain datatypes for describing geometric entities on the surface of earth. On the other side, they become increasingly important, and the CIDOC CRM version 6.2 on defines E94 Space Primitive, subclass of: E59 Primitive Value, as:

“This class comprises instances of E59 Primitive Value for space that should be implemented with appropriate validation, precision and references to spatial coordinate systems to express geometries on or relative to earth, or any other stable constellations of matter, relevant to cultural and scientific documentation.

An E94 Space Primitive defines an E53 Place in the sense of a declarative place as elaborated in CRMgeo (Doerr and Hiebel 2013), which means that the identity of the place is derived from its geometric definition. This declarative place allows for the application of all place properties to relate phenomenal places to their approximations expressed with geometries.

Definitions of instances of E53 Place using different spatial reference systems always result in definitions of different instances of E53 place approximating each other. It is possible for a place to be defined by phenomena causal to it, such as a settlement or a riverbed, or other forms of identification rather than by an instance of E94 Space Primitive. Any geometric approximation of such a place by an instance of E94 Space Primitive constitutes an instance of E53 Place in its own right, i.e., the approximating one.

Instances of E94 Space Primitive provide the ability to link CRM encoded data to the kinds of geometries used in maps or Geoinformation systems. They may be used for visualisation of the instances of E53 Place they define, in their geographic context and for computing topological relations between places based on these geometries.

E94 Space Primitive is not further elaborated upon within this model. Compatibility with OGC standards are recommended.”

These standards currently do not have a common form comprising all others. Further, geometries defined with respect to particular object shapes, such as rotationally symmetric ones, are possibly open ended.

Therefore we define in the CRM RDFS the range of properties that use E94 Space Primitive in the definition of the CRM as rdfs:Literal, and recommend the user to instantiate it with adequate datatypes compatible with rdfs:Datatype. These are for the surface of Earth for example “geo:gmlLiteral” or “geo:wktLiteral”
(xmlns:geo="http://www.opengis.net/ont/geosparql#”) or
. In order to accommodate for very large literals, see section “Very large Primitive Values” for additional definitions.

In the current version of the CIDOC CRM, only the property “*P168 place is defined by (defines place)*” has range E94 Space Primitive[9].

Since any instance of E94 Space Primitive identifies unambiguously an instance of E53 Place by a symbolic expression, E94 Space Primitive must logically be regarded as a subclass of E41 Appellation. Consequently, we define *P168 place is defined by (defines place)* as subproperty of "*P1 is identified by*", and all literals used as its range instances implicitly as instances of E41 Appellation (see section "RDF implementation tests" item 1.). See also section "Recording Names".

Recording spacetime

Recording spacetime is very similar to recording space in all aspects: The recommended datatypes of RDF1.1 do not contain datatypes for describing spacetime volumes. Developing the CIDOC CRM and CRMgeo in particular, it appeared that all phenomena that can be named and can serve for determining by their spatial extent a place do also more or less change their spatial extent over time. If their maximal spatial extent is not sufficient for the purpose of documentation, the only consistent way to approximate these "places" is to approximate them by declarative spacetime volumes.
The CIDOC CRM version 6.2 on defines E94 Space Primitive, subclass of: E59 Primitive Value, as:
"This class comprises instances of E59 Primitive Value for spacetime volumes that should be implemented with appropriate validation, precision and reference systems to express geometries being limited and varying over time on or relative to earth, or any other stable constellations of matter, relevant to cultural and scientific documentation. A Spacetime Primitive may consist of one expression including temporal and spatial information like in GML or a different form of expressing spacetime in an integrated way like a formula containing all 4 dimensions.
An E95 Spacetime Primitive defines an E92 Spacetime Volume in the sense of a declarative spacetime volume as defined in CRMgeo (Doerr & Hiebel 2013), which means that the identity of the spacetime volume is derived from its geometric and temporal definition. This declarative spacetime volume allows for the application of all E92 Spacetime Volume properties to relate phenomenal spacetime volumes of periods and physical things to propositions about their spatial and temporal extents.
Definitions of spacetime volumes using different spacetime reference systems always result in definitions of different spacetime volumes approximating each other. It is possible for a spacetime volume to be defined by phenomena causal to it, such as an expanding and declining realm, a settlement structure or a battle, or other forms of identification rather than by an instance of E95 Spacetime Primitive. Any spatiotemporal approximation of such a phenomenon by an instance of E95 Spacetime Primitive constitutes an instance of E92 Spacetime volume in its own right, i.e., the approximating one. E95 Spacetime Primitive is not further elaborated upon within this model. Compatibility with OGC standards are recommended."

There are very few standardized formats for spacetime volumes. The most simple representations are a sort of 3D/4-D right prisms[10](geometry), in which the 2D /3D-base is a geometry kept constant over a time interval (the "height" of the prism). A more elaborate method is proposed by (Niccolucci & Hermon 2015), which uses aggregates of rectangular boxes for approximating irregular spacetime volumes.

Therefore we define in the CRM RDFS the range of properties that use E95 Spacetime Primitive in the definition of the CRM as rdfs:Literal, and recommend the user to instantiate it with adequate datatypes compatible with rdfs:Datatype. In order to accommodate for very large literals, see section “Very large Primitive Values” for additional definitions.

In the current version of the CIDOC CRM, only the property “*P169 defines spacetime volume (spacetime volume is defined by)*” has range E95 Spacetime Primitive.
Since any instance of E95 Spacetime Primitive identifies unambiguously an instance of E92 Spacetime volume by a symbolic expression, E95 Spacetime Primitive must logically be regarded as a subclass of E41 Appellation. . Consequently, we define “*P169 defines spacetime volume (spacetime volume is defined by)”* as subproperty of “*P1 is identified by*”, and all literals used as its range instances implicitly as instances of E41 Appellation (see section “RDF implementation tests” item 1.). See also section “Recording Names”.

Recording numbers

“Number” is a very general mathematical term. The CIDOC CRM version 6.2 on defines E60 Number, subclass of E59 Primitive Value, as:

“This class comprises any encoding of computable (algebraic) values such as integers, real numbers, complex numbers, vectors, tensors etc., including intervals of these values to express limited precision.

Numbers are fundamentally distinct from identifiers in continua, such as instances of E50 Date and E47 Spatial Coordinate, even though their encoding may be similar. Instances of E60 Number can be combined with each other in algebraic operations to yield other instances of E60 Number, e.g., 1+1=2. Identifiers in continua may be combined with numbers expressing distances to yield new identifiers, e.g., 1924-01-31 + 2 days = 1924-02-02. Cf. E54 Dimension””

In the CIDOC CRM, the class E60 Number appears only twice, as range of “E19 Physical Object: P57 has number of parts”, and as range of “E54 Dimension: P90 has value: E60 Number”.

In CRM RDFS, the range of “P57 has number of parts” should be xsd:nonNegativeInteger.

Due to the genericity of E54 Dimension, the range of “P90 has value” cannot be identified with a particular XSD datatype. The typical museum application of this class are the spatial dimensions of an object. In that case, and for all other linear dimensions, it is recommended to instantiate the range of “P90 has value” as xsd:double.

However, the class is relevant for describing the results of all kinds of measurements and other quantitative observations, which may use very complex representations of quantities. It is not in the scope of the CRM to develop exhaustive standards for these cases, because it is much more in the expertise of the respective natural sciences to define them. Respective communities of practice are invited to propose specializations of E54 Dimension and “P90_has_value”. For instance, sensor arrays, more and more in use, pose the issue of a single measurement resulting in an array of numbers which altogether form one quantitative statement about the observed. We can describe such structures easily as one complex type

of unit (and define an IRI for it), and then regard the value to a matrix of numbers, in which each position obeys subunits as defined in the complex unit type. In order to accommodate for very large literals, see section “Very large Primitive Values” for additional definitions.

Whereas the CRM regards that intervals of primitive values are primitive values by themselves, there is currently no corresponding practice in RDF. Therefore, in analogy to the properties of E52 Time-Span, we define in CRM RDFS two more subproperties of P90 has value: “P90a_has_lower_value_limit” and “P90b_has_upper_value_limit”. Even if we regard complex matrices of numbers as one value for an instance of E54 Dimension, such as RGB images, we can argue that minimal and maximal values exist as two separate matrices of the same structure. The precise guidelines for using these properties are given in the section “Guidelines for using P90a, P90, P90b” below.

Recording string values

The CIDOC CRM version 6.2 defines E62 String, subclass of E59 Primitive Value, as:

"This class comprises coherent sequences of binary-encoded symbols. They correspond to the content of an instance of E90 Symbolic object. Instances of E62 String represent only the symbol sequence itself. They may or may not contain a language code. In contrast, instances of other subclasses of E59 Primitive value represent entities in mathematical spaces different from that of symbol sequences, by using binary-encoded symbols, such as date expressions or numbers in decimal encoding. For instance, different syntactic forms of a date expression may represent the same date, but different strings."

E62 String appears in the CRM only as range of P3_has_note and its subproperties P79_beginning_is_qualified_by and P80_end_is_qualified_by, and in particularly in the newly proposed property “E90 Symbolic Object: has symbolic content”.

E62 String corresponds to rdfs:Literal, with the above described interpretation. Instantiation with rdf:langString and xsd:string is compatible.

Recording names

In the CRM names are modelled as instances of E41 Appellation. This class comprises any symbolic object used or created to name something without requiring further meaning. The CIDOC CRM version 6.2 defines E41 Appellation, subclass of E90 Symbolic Object, as: “This class comprises signs, either meaningful or not, or arrangements of signs following a specific syntax, that are used or can be used to refer to and identify a specific instance of some class or category within a certain context.

Instances of E41 Appellation do not identify things by their meaning, even if they happen to have one, but instead by convention, tradition, or agreement. Instances of E41 Appellation are cultural constructs; as such, they have a context, a history, and a use in time and space by some group of users. A given instance of E41 Appellation can have alternative forms, i.e., other instances of E41 Appellation that are always regarded as equivalent independent from the thing it denotes. “

The CRM is an ontology in the proper sense. Therefore, instances of physical things and phenomena of the physical worlds are regarded to be the things themselves, and not their machine representation, and any identifier or name used for something from the material world is different from the thing itself. For instance, I, Martin Doerr, am an instance of E21 Person, and not any of the URIs or records that may represent me in an information system. I am unique in this world, as is any particular thing, in contrast to representations of me. In the CRM, the property "P1 is identified by" from "E1 CRM Entity" to "E41 Appellation" relates the things to their names or identifiers.

In any knowledge representation schema, any item that cannot "reside" in the machine itself due to its nature, must be represented by one selected primary identifier, in the case of RDF by a URI. For an information system to be consistent with the described reality, these selected identifiers should map one-to-one to the ontological instances they stand for. Therefore, any instance of a class represented by a URI in RDF plays a dual role: it stands for the ontological instance and is an identifier for it (see also Meghini et al. 2014). For practical reasons, we do not represent this duality by a recursive use of "P1 is identified by" from an instance to itself in its second capacity as an identifier. However, all other names and identifiers are related to the select primary identifier via "P1 is identified by". This implies that the choice about which of multiple identifiers is the primary one may be changed without changing the meaning. In contrast, owl:same_as relates two primary URIs of things as different representation of the same real world thing, aggregating the properties of both representations as valid for the real world thing.

In practice, only the URIs, literals and datatypes "reside" themselves directly in a machine and need no additional identification because they are completely identified by their content. We may distinguish four different kinds of Appellations: URIs, identifiers from local application contexts, literally defined names used in human written communication and names from oral communication and tradition. Typically, URIs and local identifiers have a unique representation as strings. However, the situation for names is more complex. For instance, 北京 is a literally defined name for the capital of China. "Bei Jing" is meant to be an representation of the same name in Latin characters (underspecified without accent marks), and not meant to be another name for the same city. "Doerr is a respelling of Dörr, a German surname[11]". The most elaborate and effective good practice for registering proper names comes from the library community (Doerr, Riva and Zumer 2012). The FRBR Review Group of IFLA decided for practical reasons to identify a name ("Nomen" in their terminology) by the identical sequence of characters in a given script, not by the binary encoding.

For historical research however, in particular capturing oral tradition, this definition is too narrow, and we are confronted in relevant CRM applications with cases of names with spelling variants and even spoken variants. All cases of names that cannot uniquely be identified with a character sequence must be represented with a URI and further properties of description must be added, by preference the new property "*E90 Symbolic Object: P190 has symbolic content*". Also, if someone wants to document facts about a name other than its spelling, a URI must first be assigned, because a character string itself cannot be referred to in RDF. This case must not be confused with documenting facts about the relation

between a name and a particular carrier of that name, because that would be a reification of this relation, and not talking about the name.

Summarizing, there are two cases:
a) A name or identifier is completely defined and identified by a character sequence or any digitally, unambiguously encoded symbol.
b) A name or identifier is identified but not defined by a URI.
As a matter of fact, RDFS provides the property rdfs:label, which implements exactly the case a) above, without the possibility to add descriptions of the name itself. SKOS specializes rdfs:label into properties such as skos:prefLabel and skos:altLabel, which define indeed the names by which things are called by people. We take therefore the use of rdfs:label as existing good practice.

Consequently, we have to regard rdfs:label as a special case of “P1 is identified by”, and all literals used as range instances of rdfs:label implicitly as instances of E41 Appellation (see section “RDF implementation tests” item 1.).
Unfortunately, our KR languages have not foreseen the case that an instance of a datatype is also an instance of a user-defined class. This causes a range conflict, which can be overcome by “punning” the range of “P1 is identified by” to be both rdfs:Literal and E41 Appellation (see section “RDF implementation tests” item 2.).

This recommended implementation allows for using both models for Appellations, via an additional URI or directly as literal, and returning with one query all range instances of “P1 is identified by” following this interpretation. The SPARQL query result separates URIs from literals automatically. So, there is no ambiguity about the nature of the result.
Only if the same name is described both directly via rdfs:label and indirectly via a URI, the matching of both would need another query.
So, the frequently asked question remains, why not avoiding this double definition and describe any instance of E41 Appellation via another URI? The answer is that actually the cases that require explicit representation of E41 Appellation are relevant but rare. On the other side, good practice requires all nodes in a semantic graph represented by a URI to carry a human-readable label in addition. This means that the storage volume and query performance would heavily be hampered by such a “pure-logic-driven” decision.
The only ambiguity that remains is the case in which the instance of Appellation is literally the URI itself, and not a URI representing an Appellation of different form. There are two solution to this problem: Either classify this URI by the class of things it identifies and use owl:same_as, or we define a specific subclass of E41 Appellation “URI”.

Language of an Appellation

Whereas common words always belong to a language, proper names normally do not belong to a language. For persons, they are normally used in the form the carrier of the name uses it, and for place names in the form the local population uses it. The latter place names are called “vernacular”. Only important and historical places use to have name variants in use in other language groups. The same holds for some meaningful titles of paintings, and translations of books, movies etc. Even specialized terms, even though not

being proper names, often are not translated. The “language” of such names is more a useful distinction for the user to recognize and distinguish the target group of a label. **We therefore recommend** the use of *rdfs:langString* for all Appellations being regarded specific to or characteristic for a language group and being directly described by a literal and not indirectly via a URI. For Appellations being described indirectly via a URI, we recommend *the use of E41 Appellation > P72 has language > E56 Language*. This holds also in the case the language to be documented is not among those that can be specified by *rdfs:langString.* For convenience, the next version of the CRM RDFS will contain the class *“E41_E33_Linguistic Appellation”,* sublass of *E41 Appellation* and *E33 Linguistic Object.*

Very large Primitive Values

In general, representations of primitive values do not have a size limit, except for time expressions. In particular, geometries may be very large polygon sequences or other large datasets, as well as arrays of numbers from scientific data.

Very large strings one would normally describe in a file and instantiate E90 Symbolic Object or a subclass of it with the URL. However, the question is, if the URL would indeed be a good persistent identifier, since the URL stands for a physical location, albeit indirectly addressed. The Linked Open Data community has not yet given satisfactory answers for the long term validity of resolvable URIs. If the URL is not a good identifier, another, primary URI should be chosen, and the content found under this URL should be related to the primary URI as a representative of the content of the symbolic object identified with the primary URI. **We recommend** for this relation a specific property to be decided by the CRM Special Interest Group, either specific to CRM RDF, or in CRMbase.

In the case of the other Primitive Values, except for time, **we recommend** a “punning” solution: The properties that have as range a Primitive Value in the CRM should be defined in CRM RDF to have as range rdfs:Literal **and** the respective subclass of E59 Primitive Value:

· *P90 has value*: has range both rdfs:Literal and has range E60 Number”.

· *P168 place is defined by (defines place)*” has range both rdfs:Literal and has range E94 Space Primitive

· *P169 defines spacetime volume (spacetime volume is defined by)* has range both rdfs:Literal and has range E95 Spacetime Primitive

The respective subclass should **only be instantiated** with a URL of a file containing the content in the case the content does **not fit** well in an rdfs:Literal. See examples in the section “RDF implementation tests” item 4.

Defining custom datatypes

New datatypes can be defined, published in a respective namespace and added to the RDFS datatypes, for instance using a cidoc crm namespace. The section “RDF implementation tests” item 3 shows how a cidoc crm space primitive can be defined as datatype.

Any string of a datatype is stored in a triple store as a literal. If the datatype is a compound value, such as xsd:dateTime, there are specific functions that are in reality String Functions which can isolate the different parts, for instance that of a date (year, month etc), at query time, and makes them accessible to be specified as query elements. See “RDF implementation tests” item 5.

The alternative, to define for each kind of compound value a series of subproperties of P_has_value, makes data entry, data display and computation with these values much more complex. We do not recommend this solution.

This means that it is up to the designer of these functions to define a convenient syntax within the respective literal, and of course a question of standardization. The respective String Functions are either compiled into the query software, or invoked by code that runs on query results. The latter is much easier to handle by users, if they have no IT support to embed the code in the query system. For our purposes, most custom datatypes need not be broken up into its parts by the query system itself, because they will be interpreted anyway after querying, often just by the user reading them.

We recommended users to find respective custom datatypes and their syntax at the communities of practice dealing most with these kinds of values and to propose them to the CIDOC CRM Special Interest Group for approval.

Properties of properties

As mentioned above, RDF does not support properties of properties. Therefore, users are recommended two ways to work around:

A. The current properties of properties in the CRM have all as range “E55 Type”. Therefore they correspond to subtyping of the respective property by a local vocabulary.

B. For the cases in which the local vocabulary is not fixed, there is a recommended form of reification via an auxiliary “property class”. This replaces the former recommendation to use E13 Attribute assignment in order to introduce user defined property types.

The two solutions have pros and cons with respect to query performance, user interface programming and flexibility to cater for a local, evolving terminology.

Solution A:

Users that have fixed vocabularies of property types for those properties foreseeing in the CRM a “type” or “Pxx.1” property, may transform these types into their own subproperties for the respective CRM properties, such a as "P3 has note":

Instead of P3 has note (P3.1 has type : parts description) declare

```
<rdf:Property rdf:about="P3_parts_description">
        <rdfs:domain rdf:resource="E1_CRM_Entity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
        <rdfs:subPropertyOf rdf:resource="P3_has_note"/>
</rdf:Property>
```

This ensures that a graph using these subproperties can consistently be retrieved via their superproperties. In other terms, a system using this solution a) is query compatible with the CIDOC CRM without using properties of properties in the query and b) it ensures that typing

a property instantiates the base property and therefore the complete graph is contained in the respective query answer. It is the most **efficient implementation**, both in terms of **query performance and storage** overhead. User interface programmers should query these additional subproperties and create **at run-time** selection lists of them, rather than hard-coding the vocabularies. If such extensions are widely built, they can **reveal an emerging** good practice and become subject to a standardization of its own. The drawback is that the vocabulary must be loaded to database platform before-hand, and this mechanism only applies to types of properties. It is the **preferred solution**.

In case users have no other choice than to deal with open vocabularies of property types, or would need extensions with properties of properties other than types, they should resort to solution B. This solution uses “property classes”, i.e., representing an n-ary property by an auxiliary RDF class, which are provided together with the CRM RDF Schema[12]. This solution is logically adequate and further extensible, but is more complex with respect to query formulation, has considerably slower query response times in current knowledge base platforms than the above and more storage overhead. It is more obvious for user interface programmers to create the respective selection lists, and users may introduce new types at runtime.

About implementing multiple Instantiation

Knowledge Representation models can assign multiple classes to a given instance identifier. After that, all properties of each assigned class are applicable for this identifier. This construct is called “multiple instantiation. For instance, a calligraphy is an “image” and a “linguistic object”, having a language and a painting style. This is not possible with Relational data structures, because instance identification is limited to the entity (class) or with XML-like data structures, because instance identification is by structural position (additional identifiers can be used for linking).

Therefore many users are not aware of this feature, and even KR tools do not systematically guide users to use it: Once an instance is classified by one class, the tool should not allow for using a property of another class, but most likely will not advise the user that she could add the additional class to the instance. Nevertheless, it is a key feature of KR models that facilitating modularizing ontologies and the often advertised ability to combine different ontologies.
The CRM as ontology relies heavily on multiple instantiation: Combination of classes that are applicable to some instances only incidentally and have no properties specific to this combination are not modelled in the CRM individually as subclasses of multiple parent classes. The latter would be called “multiple IsA”. To avoid multiple IsA in such cases is an important normalization principle to keep the ontology very compact and unambiguous.
In the specification modules of mapping software used to transform data into a CRM-compatible form, care must be taken to foresee and allow the user to combine RDF classes systematically.
Some combinations of classes may more frequently occur, such as combining E41 Appellation with E33 Linguistic Object in order to reach E56 Language via P72 has language. In a local system that does not easily support multiple instantiation, the candidate cases for multiple instantiation may be combined in subclasses using multiple IsA. For their

labels, we recommend to aggregate the class identifier codes as in: “E41_E33_Linguistic Appellation”. Such a replacement is query compatible with the standard. A respective import/export system simply needs to make the trivial replacements of the respective class combinations with their multiple IsA counterparts and vice-versa in order to achieve import/export compatibility.
Users may provide feedback about frequent cases where multiple instantiation is used, in order to guide users to these modelling cases. These could systematically be entered into the CRM RDF implementation, without requiring the CRM standard itself to repeat them.

CIDOC CRM and other frameworks

The CIDOC CRM generally foresees to be used together with other ontologies and respective implementations for all domains that fall outside the scope of the CRM and for which an active, internationally acknowledge community exist that maintains the respective ontology. It is not the intention of the CIDOC CRM to compete with communities that have superior domain knowledge, but rather to benefit from their insight, given the ontology has been created with a compatible methodology or diligence. The CRM may deliberately reduce its scope in favour of such a community.

This theoretical principle finds in practice the following obstacles: The respective domain ontology will most likely define general concepts that overlap in an incompatible way with the CRM. For instance, OGC defines a few fundamental concepts differently analysed in the CRM, and many concepts the CRM never intends to deal with, but wants to recommend their use (see Doerr, Hiebel and Eide 2013).

This problem can be solved by a so-called “articulation”. Overlapping concepts are redefined and new concepts are introduced to create a more detailed model of the overlapping area which can be mapped to both ontologies. Users must replace the incompatible parts of both ontologies with the refined model, and use all other concepts of both ontologies together with it. The CIDOC CRM SIG aims at adapting the CRM to important domain ontologies by adopting the refined model.

Such a model is **CRMgeo**, linking the CIDOC CRM with **OGC standards** (Doerr, Hiebel and Eide 2013, Doerr and Hiebel 2013). I.e., OGC standards can be used, except for those concepts redefined in CRMgeo, together with CRMgeo and CRM “base”.
The bibliographic “**FRBR Family of models**” by IFLA on the other side was formulated as Entity-Relationship model, a methodology incompatible with the CRM. Therefore, both communities, CIDOC and IFLA, have engaged in a compatible, complete reformulation of the FRBR models, now “Library Reference Model (**LRM**)” as the CRM-compatible ontology FRBRoo version 1-3, version 3 now renamed to “LRMoo”.

SKOS[13] is an RDF schema originally designed to describe terminologies of universals of entities. *E55 Type* may in general refer to even less formal systems of terminology than SKOS and it is also used in the CRM to refer to property types. Therefore, **it is recommended** to define *E55 Type* as superclass of skos:Concept and *P127_has_broader_term* as superproperty of skos:broader / *P127i_has_narrower_term* as superproperty of skos:narrower. It is **not recommended** and incompatible with the CRM **to**

use skos:Concept for **places and persons**. It is also incompatible to use skos:exactMatch to link from CRM instances to authorities such as VIAF, ULAN and TGN. Authorities often define places and people as skos:Concept for classification in their discourses and assign a URI for each instance of skos:Concept. These URIs are instances of E41 Appellation. Therefore it is recommended that instances of places and people should link to authorities with the property P1_is_identified_by. See also LRMoo about the distinction between natural persons and literary characters derived from those.

The Dublin Core Metadata Element Set, Version 1.1, has limited compatibility. The properties, dc:relation and dc:date are underspecified, and their use leads to ambiguous overlaps with CRM-based descriptions. The properties dc:publisher, dc:creator, dc:contributor, dc:source may be interpreted as shortcuts of CRM properties, but lack the important intermediate events. It is not recommended to combine DC with the CRM. Alternative, separate descriptions of things with The Dublin Core Metadata Element Set are, of course, no problem.

The compatibility of other frameworks with the CRM needs to be investigated. The CRM SIG will be glad to receive request and collaborate with respective initiatives.

References

Doerr, Martin & Riva, Pat & Žumer, Maja. (2012). *FRBR Entities: Identity and Identification*. In: Cataloging & Classification Quarterly. Volume 50, 2012 - Issue 5-7: The FRBR Family of Models. Pages 517-541 DOI: 10.1080/01639374.2012.681252.

Hiebel, G.H, Doerr, M., & Eide, Ø. (2013). Integration of CIDOC CRM with OGC Standards to model spatial information (Session5, 522). *Computer Applications and Quantitative Methods in Archaeology (CAA) 2013*, Perth-Australia, 25th -28th March 2013. (pdf).

Doerr, M., & Hiebel, G.H (2013). CRMgeo: Linking the CIDOC CRM to GeoSPARQL through a Spatiotemporal Refinement. 2013.TR435_CRMgeo_CIDOC_CRM_GeoSPARQL.pdf

Franco Niccolucci, Sorin Hermon (2015) "*Representing gazetteers and period thesauri in four-dimensional space–time*", Published 2015 in International Journal on Digital Libraries, DOI:10.1007/s00799-015-0159-x

Meghini, C., Spyratos, N., Sugibuchi, T. and Jitao Yang. (2014). *A Model for Digital Libraries and its Translation to RDF*. In: Journal on Data Semantics, June 2014, Volume 3, Issue 2, pp 107–139. https://doi.org/10.1007/s13740-013-0029-x

Meghini, C. and Doerr, M., 2018, 'A first-order logic expression of the CIDOC Conceptual Reference Model', International Journal of Metadata, Semantics and Ontologies.

Annex

Commented overview of RDFS datatypes

	Datatype	Value space (informative)	CRM recommendation	Comment
Core types	xsd:string	Character strings (but not all Unicode character strings)	IsA E62 String and default.	E62 may contain more kinds of symbols/scripts, such as xsd:hexBinary or Linear B
	xsd:boolean	true, false	IsA I6 Belief Value. Do not use.	Only belief values in CRMInf may use these values, but they should at least be three-valued: True, False, Unknown.
	xsd:decimal	Arbitrary-precision decimal numbers	Do not use	
	xsd:integer	Arbitrary-size integer numbers	IsA E60 Number	
IEEE floating-point numbers	xsd:double	64-bit floating point numbers incl. ±Inf, ±0, NaN	IsA E60 Number	
	xsd:float	32-bit floating point numbers incl. ±Inf, ±0, NaN	IsA E60 Number	
Time and date	xsd:date	Dates (yyyy-mm-dd) with or without timezone	IsA E61 Time Primitive, do not use	It could be used for P81,P82, but only for intervals of days. That does not seem to make much sense. It must not be used for P81a,P81b, P82a, P82b

	xsd:time	Times (hh:mm:ss.sss…) with or without timezone	Do not use	
	xsd:dateTime	Date and time with or without timezone	Use pairwise for E61 Time Primitive	Use for P81a,P81b, P82a, P82b.
	xsd:dateTimeStamp	Date and time with required timezone	Use pairwise for E61 Time Primitive	Use for P81a,P81b, P82a, P82b.
Recurring and partial dates	xsd:gYear	Gregorian calendar year	Do not use	
	xsd:gMonth	Gregorian calendar month	Do not use	
	xsd:gDay	Gregorian calendar day of the month	Do not use	
	xsd:gYearMonth	Gregorian calendar year and month	Do not use	
	xsd:gMonthDay	Gregorian calendar month and day	Do not use	
	xsd:duration	Duration of time	IsA E54 Dimension,	use for P83 had at least duration (was minimum duration of): E54 Dimension P84 had at most duration (was maximum duration of): E54 Dimension
	xsd:yearMonthDuration	Duration of time (months and years only)	See above	
	xsd:dayTimeDuration	Duration of time (days, hours, minutes, seconds only)	See above	
Limited-range integer numbers	xsd:byte	-128…+127 (8 bit)	Do not use	
	xsd:short	-32768…+32767 (16 bit)	Do not use	
	xsd:int	-2147483648…+2147483647 (32 bit)	Do not use	
	xsd:long	-9223372036854775808…+9223372036854775807 (64 bit)	Do not use	
	xsd:unsignedByte	0…255 (8 bit)	Do not use	
	xsd:unsignedShort	0…65535 (16 bit)	Do not use	
	xsd:unsignedInt	0…4294967295 (32 bit)	Do not use	
	xsd:unsignedLong	0…18446744073709551615 (64 bit)	Do not use	
	xsd:positiveInteger	Integer numbers >0	May be used in extensions.	

	xsd:nonNegativeInteger	Integer numbers ≥0	IsA E60 Number.	Use for P57 has number of parts. It may be useful to distinguish zero parts from not knowing parts.
	xsd:negativeInteger	Integer numbers <0	Do not use	
	xsd:nonPositiveInteger	Integer numbers ≤0	Do not use	
Encoded binary data	xsd:hexBinary	Hex-encoded binary data	IsA E62 String.	Note, that it represents bits and not the hex symbols. Can be useful for content models.
	xsd:base64Binary	Base64-encoded binary data	Do not use	
Miscellaneous XSD types	xsd:anyURI	Absolute or relative URIs and IRIs		
	xsd:language	Language tags per [*BCP47*]	IsA E56 Language	Since there are many more historical languages than xsd:language comprise, we may better use it as rdf:label or identifier for those covered by xsd:language.
	xsd:normalizedString	Whitespace-normalized strings	IsA E62 String	
	xsd:token	Tokenized strings	IsA E62 String	
	xsd:NMTOKEN	XML NMTOKENs	Do not use	
	xsd:Name	XML Names	Do not use	
	xsd:NCName	XML NCNames	Do not use	

Guidelines for using P81a, P81b, P82a, P82b

Oct 23, 2020

The range of the properties "P81 ongoing throughout" and "P82 at some time within" are defined in the CRM as E61 Time Primitive. Instances of E61 Time Primitive are defined as closed, contiguous intervals on the natural time dimension in which we live. “Closed” means that the endpoints belong to the interval. “Contiguous” means that there are no gaps between the endpoints in the interval (which holds for “intervals” in general).
The reason to describe time spans with inner and outer intervals is the existence of a very efficient algebra for calculating resulting areas of determinacy and indeterminacy (Cowley & Plexousakis 2000). Further, they are motivated by the British MIDAS Heritage standards [https://en.wikipedia.org/wiki/MIDAS_Heritage] and easy to define in Relational databases.
Since the E61 Time Primitive of the CRM cannot be expressed in RDF directly, in the official RDF implementation of the CIDOC CRM, we define four properties replacing P81 and P82 adequately using xsd:dateTime.

P81 ongoing throughout

Property P81 describes the maximum known temporal extent of an E52 Time-Span, i.e. the extent it is ongoing throughout. It is replaced in this RDF version by the property "P81a_end_of_the_begin" and "P81b_begin_of_the_end", to be used together.
"P81a_end_of_the_begin" should be instantiated as the earliest point in time the user is sure that the respective temporal phenomenon is indeed ongoing. We call it “end_of_the_begin”, because it also constitutes an upper limit to the end of the indeterminacy or fuzziness of the beginning of the described temporal phenomenon.
"P81b_begin_of_the_end" should be instantiated as the latest point in time the user is sure that the respective temporal phenomenon is indeed ongoing. We call it “begin_of_the_end”, because it also constitutes a lower limit to the beginning of the indeterminacy or fuzziness of the end of the described temporal phenomenon.

It is correct to assign the same value to “P81a_end_of_the_begin” and “P81b_begin_of_the_end”, if no other positive knowledge exists. It is also correct not to instantiate P81 for a time span, if there is no evidence that the temporal phenomenon was definitely occurring at a particular time.

If a respective reasoning is installed, and no evidence exists about the point in time that the phenomenon was definitely ongoing, one may specify “P81a_end_of_the_begin” as being later than “P81b_begin_of_the_end”, indicating that the indeterminacy of knowledge (not of being) of the begin overlaps with the indeterminacy of knowledge (not of being) of the end. Formally, this constitutes a negative interval for *P81 ongoing throughout* (Holmen&Ore 2010).

If a value for “P81a_end_of_the_begin” is given with a precision less than that of xsd:dateTime (i.e. seconds), such as in days or years, the implementation should “round it up” to the last instant of this time expression, e.g. 1971 = Dec 31 1971 23:59:59. Respectively, for

“P81b_begin_of_the_end” the implementation should “round it down”, e.g. 1971 = Jan 1 1971 0:00:00. If values are needed that are not within the range or precision of xsd:dateTime, e.g., for paleontology, this property should be extended with another, suitable data type.

P82 at some time within

Property P82 describes the narrowest known outer bounds of the temporal extent of an E52 Time-Span, i.e. that the described temporal phenomenon is definitely ongoing “at some time within” this interval. It is replaced in the official RDF version by the properties "P82a_begin_of_the_begin" and "P82b_end_of_the_end", to be used together.

"P82a_begin_of_the_begin" should be instantiated as the latest point in time the user is sure that the respective temporal phenomenon is indeed not yet happening. We call it “begin_of_the_begin”, because it also constitutes a lower limit to the beginning of the indeterminacy or fuzziness of the beginning of the described temporal phenomenon.
"P82b_end_of_the_end" should be instantiated as the earliest point in time the user is sure that the respective temporal phenomenon is indeed no longer ongoing. We call it “end_of_the_end”, because it also constitutes an upper limit to the end of the indeterminacy or fuzziness of the end of the described temporal phenomenon.

It is not correct to assign the same value to “P82a_begin_of_the_begin” and “P82b_end_of_the_end”. If a value for “P82a_begin_of_the_begin” is given with a precision less than that of xsd:dateTime (i.e. seconds), such as in days or years, the implementation should “round it down” to the first instant of this time expression, e.g. 1971 = Jan 1 1971 0:00:00. Respectively, for “P82b_end_of_the_end” the implementation should “round it up”, e.g. 1971 = Dec 31 1971 23:59:59.

It must always hold that “P82a_begin_of_the_begin” is before “P82b_end_of_the_end”, “P81a_end_of_the_begin” and “P81b_begin_of_the_end”.
It must always hold that “P82b_end_of_the_end” is after ““P82a_begin_of_the_begin”, “P81a_end_of_the_begin” and “P81b_begin_of_the_end”.

“P82a_begin_of_the_begin” and “P82b_end_of_the_end” should always be assigned a value for any past phenomenon. The scholarly practice of not giving outer bounds for an event, because they are not known down to a desired precision (e.g. of three years), is not helpful for automated reasoning. In that case, the machine may conclude that a historical event could have happened at the time of the dinosaurs. Therefore any value is better than no value, even if it is relatively far away from the most likely value. It is an error to associate any implicit degree of approximation with these values. Only for phenomena that may not yet have ended at the time of documentation the end of the time-span should not be specified.

References:

Holmen, Jon; Ore, Christian-Emil Smith. Deducing event chronology in a cultural heritage documentation system. I : Frischer, Bernard, Jane Webb Crawford and David Koller (eds), “Making History Interactive”. Computer Applications and Quantitative Methods in Archaeology (CAA). Proceedings of the 37th International Conference. Archaeopress 2010 ISBN 9781407305561

Wes Cowley, Dimitris Plexousakis: An Interval Algebra for Indeterminate Time. AAAI/IAAI 2000: 470-475

Guidelines for using P90a, P90, P90b

The CRM recommends to approximate numerical values of Dimensions with intervals. The range of the respective property "P90 has value" is defined in the CRM as E60 Number. Whereas the CRM regards that intervals of primitive values are primitive values by themselves, there is currently no corresponding practice in RDF. Therefore, in analogy to the properties of E52 Time-Span, we define in CRM RDFS two more subproperties of P90 has value: "*P90a_has_lower_value_limit*" and "*P90b_has_upper_value_limit*".

The reasons for recommending this approximation are the following: All scientific measurements of non-discrete values are imprecise because of the tolerances of the measurement devices, shortcomings in applying the procedures and the indeterminacy of the measured effect itself. In natural sciences, important results of measurements are associated with possibly complex probabilistic distributions for the true value of the measured effect.

The most complex case relevant for cultural-historical data are the so-called "battleship curves" for calibrated C14 dating data. Many of these distribution models actually extend to infinity with non-zero probability, which is neither practical nor always justified. In the case of C14 however, the actual width of the distribution is often underestimated. Nevertheless, even data with a given probabilistic uncertainty to infinity are typically associated by scientists with narrower "confidence intervals" at one to three "standard deviations", i.e., with a probability of some 68% – 99.7% for the value to be in the given range (https://en.wikipedia.org/wiki/Standard_deviation).

Whereas querying globally a very large aggregation of cultural-historical data by time intervals is highly relevant, querying and reasoning with different approximations of dimensions is normally restricted to quite narrow questions. For many cases, a medium value without explicit limits is sufficient for the application, such as the length of a museum object in millimeters for packaging it in a box. Nevertheless, querying explicit representation of actual outer limits or at least reasonably wide confidence intervals is computationally highly effective, and therefore a good way to ensure recall at query time, i.e., that the relevant results are contained in the answer to the query, even if it also contains irrelevant ones.

We therefore recommend to use *P90_has_value* for documenting a medium value or a value without error estimates, when the precision appears to be self-evident or irrelevant.

We recommend to use *P90a_has_lower_value_limit* for documenting the highest explicit lower limit available for the respective value, even if it provides very wide margins. It is an error to omit the lower limit even if it appears to be overly pessimistic.

We recommend to use *P90b_has_upper_value_limit* for documenting the lowest explicit upper limit available for the respective, even if it provides very wide margins. It is an error to omit the upper limit even if it appears to be overly pessimistic.

In case of approximating probabilistic distributions, we recommend to keep lower and upper limit at two standard deviations or enclosing the true value with 95% probability.

P90a_has_lower_value_limit should always be used together with *P90b_has_upper_value_limit.* If they are used, the property *P90_has_value* may be used as well or be omitted.

RDF implementation tests

1. rdfs:label as subproperty of P1 is identifiedby:

<rdf:Property rdf:about=**="http://www.w3.org/2000/01/rdf-schema#label">**
<rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
<rdfs:range rdf:resource=" http://www.w3.org/2000/01/rdf-schema#Literal "/>
<rdfs:subPropertyOf rdf:resource="P1_is_identified_by"/>
</rdf:Property>

Query (Give me all the superproperties of rdfs:label) :

```
select * where {
rdfs:label rdfs:subPropertyOf ?p
}
```

Result from Virtuoso:

p:
http://www.cidoc-crm.org/cidoc-crm/P1_is_identified_by

2. Adding rdf:Literal as range of P1 is identified by:

The cidoc_crm.rdfs was altered to include the following:

<rdf:Property rdf:about="P1_is_identified_by">
<rdfs:label xml:lang="en">is identified by</rdfs:label>
<rdfs:domain rdf:resource="E1_CRM_Entity"/>
<rdfs:range rdf:resource="E41_Appellation"/>
</rdf:Property>
<rdf:Property rdf:about="P1_is_identified_by">
<rdfs:label xml:lang="en">is identified by</rdfs:label>
<rdfs:domain rdf:resource="E1_CRM_Entity"/>
<rdfs:rangerdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

The cidoc crm schema was uploaded in virtuoso and the following query (give me the range of P1_is_identified_property) was executed to be sure that the changes have been applied:

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select * where { crm:P1_is_identified_by rdfs:range ?range}
```

result:

range
http://www.cidoc-crm.org/cidoc-crm/E41_Appellation
http://www.w3.org/2000/01/rdf-schema#Literal

So, it is confirmed that the two ranges have been added. We repeat at this point that Virtuoso **does not apply** any semantic validation. The purpose of this test is to prove that this exercise is possible even though conceptually it may not be correct.

Data example:

1. The ttl data that was presented previously has been added in virtuoso:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix crm: <http://www.cidoc-crm.org/cidoc-crm/> .

<http://example.com/person/alexander_the_great>
crm:P1_is_identified_by <http://example.com/appellation/alexander_the_great> .

<http://example.com/appellation/alexander_the_great>
rdfs:label  "Alexander the Great" .

<http://example.com/person/alexander_the_great>
 rdfs:label  "Alexander the Great" .

<http://example.com/person/alexander_the_great>
crm:P1_is_identified_by  "Alexander the Great" .
```

2. A query to return all the “identifiers” of alexander the great using the is identified property was applied:

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select * where
{ <http://example.com/person/alexander_the_great>  crm:P1_is_identified_by ?identifier }
```

result:

identifier
http://example.com/appellation/alexander_the_great
Alexander the Great

3. Defining a CIDOC CRM custom datatype:

We need a cidoc crm namespace. An initial suggestion would be the following:
Prefix: cdt: http://www.cidoc-crm.org/cidoc-crm/datatypes/

Namespace for space primitive : **cdt:space_primitive**

What needs to be defined in cidoc crm rdfs to create the new cidoc crm datatypes?

The class rdfs:Literal is the class of literal values such as strings and integers. Property values such as textual strings are examples of RDF literals. rdfs:Literal is an instance of rdfs:Class. rdfs:Literal is subclass of rdfs:Resource. rdfs:Datatype is the class of datatypes.

All instances of rdfs:Datatype correspond to the RDF model of a datatype rdfs:Datatype is both an instance of and a subclass of rdfs:Class. Each instance of rdfs:Datatype is a subclass of rdfs:Literal.

So, the cidoc crm datatypes must be instances of rdfs:Datatype that needs to be a subclass of rdfs:Literal that is a subclass of rdfs:Class and **each instance of rdfs:Datatype must be a subclass of rdfs:Literal.**

The addition to cidoc crm rdfs is the following (using the example of space primitive):

```
<rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Literal">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Literal</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Datatype">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Datatype</rdfs:label>
  <rdfs:comment>The class of RDF datatypes.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://www.cidoc-crm.org/cidoc-crm/datatypes/space_primitive">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Datatype</rdfs:label>
  <rdfs:comment>The class of RDF datatypes.</rdfs:comment>
  <rdfs:subClassOf rdf:resource=" http://www.w3.org/2000/01/rdf-schema#Literal "/>
</rdfs:Class>
```

And afterwards there needs to be the information in the triple store (or in RDF data in general) that the cidoc – crm datatype is an instance of rdfs:datatype.

```
<rdf:Description rdf:about="http://www.cidoc-crm.org/cidoc crm/datatypes/space_primitive">
<rdf:type resource=" http://www.w3.org/2000/01/rdf-schema#Datatype"/>
</rdf:Description>
```

And then the following query (select * datatypes) was executed to validate the work done:

```
select ?dt where {
?dt a <http://www.w3.org/2000/01/rdf-schema#Datatype> .
?dt rdfs:subClassOf <http://www.w3.org/2000/01/rdf-schema#Literal>
}
```

result**[14][15][16]**:

?dt
http://www.cidoc-crm.org/cidoc-crm/datatypes/space_primitive

Now we use this datatype to describe a birthplace:

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/">
<rdf:Description rdf:about="http://example.com/actor/rob">
  <rdf:type rdf:resource="http://www.cidoc-crm.org/cidoc-crm/E21_Person"/>
    <rdfs:label>Rob</rdfs:label>
                    <crm:p98i_was_born>
          <crm:E67_Birth rdf:about="http://example.com/event/rob_birth">
            <crm:p7_took_place_at>
          <crm:E53_Place rdf:about="http://example.com/place/rangiora">
                  <rdfs:label>Rangiora</rdfs:label>
                  <crm:P168_place_is_defined_by rdf:datatype="http://www.cidoc-crm.org/cidoc-
crm/datatypes/space_primitive">
 "POLYGON((172.565456 -43.285409, 172.622116 -43.285409, 172.622116 -43.323697,   172.565456 -
43.323697, 172.565456 -43.285409))"
                        </crm:P168_place_is_defined>
          </crm:E53_Place>
                </crm:p7_took_place_at>
          </crm:E67_Birth>
                  </crm:p98i_was_born>
                  </rdf:Description>
</rdf:RDF>
```

3. Instantiating E94 with a file:

In the this version of the above example the place is defined by a shape file:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
xmlns:crm=http://www.cidoc-crm.org/cidoc-crm/>
<rdf:Description rdf:about="http://example.com/actor/rob">
 <rdf:type rdf:resource="http://www.cidoc-crm.org/cidoc-crm/E21_Person"/>
  <rdfs:label>Rob</rdfs:label>
   <crm:p98i_was_born>
          <crm:E67_Birth rdf:about="http://example.com/event/rob_birth">
          <crm:p7_took_place_at>
            <crm:E53_Place rdf:about="http://example.com/place/rangiora">
                  <rdfs:label>Rangiora</rdfs:label>
                  <crm:P168_place_is_defined_by>
          <crm:E94_Space_Primitive rdf:about="http://example.com/file/rangiora.shp">
          <rdfs:label>Rangiora.shp</rdfs:label>
          <crm:p2_has_type rdf:resource="http://example.com/type/ESRIshapefile"/>
                  </crm:E94_Space_Primitive >
                  </crm:P168_place_is_defined_by>
        </crm:E53_Place>
                   </crm:p7_took_place_at>
                  </crm:E67_Birth>
                  </crm:p98i_was_born>
          </rdf:Description>
</rdf:RDF>
```

---

[1]http://www.cidoc-crm.org/sites/default/files/2017-09-30%23CIDOC%20CRM_v6.2.2_esIP.pdf

[2] The metamodel of the CIDOC CRM is actually one of the variants of knowledge representation models that uses generalization / specialization constructs as object-oriented models do. An exact definition can be found in (Meghini & Doerr 2018).

[3] http://www.cidoc-crm.org/sites/default/files/cidoc_crm_v6.2-draft-2015August.rdfs

[4] The only classes it does not define are the class E59 Primitive Value and its subclasses. How these are implemented will be described later in this text

[5] Insert URL of OWL version

[6] A “digital surrogate”, such as a 3D model of an object, must not be confused with the real thing it depicts. It does not bring the real thing into a machine.

[7] This does not strictly hold for texts. The problem of text identity is discussed in the section “Recording String Values”.

[8] This is the scope note of E59 Primitive Value of the CIDOC CRM version 6.

[9] The concepts E47 Spatial Coordinates,crmgeo: SP5 Geometric Place Expression,

crmgeo:Q10 defines place and P168 place is defined by (defines place) will be revised soon. E94 Space Primitive should replace E47 Spatial Coordinates and SP5 Geometric Place Expression. P168 place is defined by (defines place) should replace Q10 defines place. It may be useful in the CRM RDFS to specify two subproperties of P168, one having as range “geo:wktLiteral” and another “geo:gmlLiteral” (xmlns:geo="http://www.opengis.net/ont/geosparql#”)
.

[10] https://en.wikipedia.org/wiki/Prism_(geometry)
[11] https://en.wikipedia.org/wiki/Doerr
[12] http://www.cidoc-crm.org/sites/default/files/CRMpc_v1.1_0.rdfs
[13] https://www.w3.org/2004/02/skos/
[14] https://www.w3.org/TR/swbp-xsch-datatypes/
[15] https://www.w3.org/TR/rdf11-concepts/
[16] http://infolab.stanford.edu/~melnik/rdf/datatyping/

V1.0 (previous version)

Implementing the CIDOC Conceptual Reference Model in RDF

Version 1.0
Martin Doerr, Richard Light, Gerald Hiebel
January 2020

Introduction

The CIDOC Conceptual Reference Model (the "CRM") [1] is defined in terms of an object-oriented metamodel[2]. It is an abstract, logical expression of the concepts and relationships (or classes and properties) which are relevant to cultural heritage documentation. In its native form it can be used to analyse and compare different standards and systems. For this purpose, there is no need to be concerned with implementation details, which vary between encoding paradigms and over time. Since the first conception of the CIDOC CRM, object-oriented and the related knowledge representation models have changed frequently: OMG standards, KL-ONE, TELOS, KIF, DAML, OIL, to name just a few. Therefore, these are typically not specified in the Definition of the CRM.

The CIDOC CRM however has been promoted as ISO standard when RDF became a W3C recommendation in 1998, the first knowledge representation model to acquire a status of an international recommendation, which, by the way, did not deviate substantially from the one used to define the CIDOC CRM. Nevertheless, the CIDOC CRM has not been defined using RDF Schema itself, because it aims at providing the widest possible logical basis for comparing and integrating data implemented under different encoding paradigms, seeking a compromise between useful expressive power for the ontology and enough simplicity to be implementable in different relevant encoding paradigms, either directly by their built-in constructs or requiring some additional S/W code. In the meanwhile, RDF is also undergoing its own evolution becoming refined and extended by OWL. Therefore the CIDOC CRM maintains a logical form which is independent from RDF.

Characteristically, the original CRM object-oriented model uses constructs, such as 'properties of properties', which cannot be expressed in RDF, and a set of logical deductions (such as the "shortcuts"), which can be implemented with additional code or by the standard query system in relevant current database or knowledge base types.

Another serious reason for keeping the logical independence lies in a much more subtle, but fundamental difference between describing things in the world in terms of logic and describing data by a data definition and encoding language, which becomes most apparent when distinguishing between real world things and their identifiers and when describing mathematical value spaces, which is necessarily common to all data encoding paradigms, including RDF. The logical-ontological compatibility of data implemented in different data definition and encoding languages cannot be defined based on a data definition or encoding language.

RDF is currently the knowledge encoding paradigm closest to the CIDOC CRM and the most widely spread. Therefore, for each official release of the CRM, an RDF Schema expression of the CRM will be published as official resource on the CIDOC CRM web site[3]. However, on its own this Schema does not contain sufficient information to allow potential implementers to design a complete CRM-compatible framework in RDF, as indicated above. Also, particular global identification conventions of CRM RDF implementations of different CRM versions need explanation.
The aim of this document is to complement the Definition of the CRM and the respective RDF Schema implementation by providing guidance on recommended techniques for using the CIDOC CRM within an RDF implementation, for instance running on an RDF-enabled Triple Store or Graph database.

The CRM RDF Schema

The CRM RDF Schema defines most[4] of the classes and properties which make up the CRM, and specifies the domain and range of these. Furthermore, the URLs specified in the Schema (with base http://www.cidoc-crm.org/cidoc-crm/) act as valid dereferenceable Linked Data identifiers. So you can define a CRM prefix:

@PREFIX crm: <http://www.cidoc-crm.org/cidoc-crm/>

and then use CRM identifiers (e.g. “crm:P53_Place”) freely within your data.
The following design decisions have been taken when designing the CRM RDF Schema. (These will appear as a comment at the start of the CRM RDF Schema document)

The Equivalent Part

Here we describe the part of the CRM RDF Schema that corresponds one-to-one to the Definition of the CIDOC CRM.
1. The RDF class and property naming rules do not allow “space” characters. Hence the “space” character is replaced in RDF names by the “underscore” (spacing underscore, low line, horizontal bar) character. For instance "E63 Beginning of Existence" becomes in RDF "E63_Beginning_of_Existence" or "P2i is type of" becomes in RDF "P2i_is_type_of".
2. RDF does not allow one to instantiate bi-directional properties. Therefore, each CRM property is represented as two RDFS properties, each with a specific “direction”. For instance "P2 has type (is type of)" is represented as:

- "P2_has_type" for the domain to range direction
- "P2i_is_type_of" for the range to domain direction

The only logical distinction between the two directions is the fact of inverse reading. In the recommended OWL version[5], this is explicated by declaring one direction as “inverse_of” the other. Practically, this means that an implementation may decide to replace one direction with the opposite via software, exchanging domain and range accordingly, or add or remove one direction for convenience without affecting compatibility or meaning. This can be useful for optimizing display, storage or querying.
3. Scope notes within the Schema are represented as <rdfs:comment> elements.
4. The encoding contains labels in languages other than English, which are taken from the latest translations of current or previous versions of the CIDOC CRM.

5. Any other differences in labels, scope notes and semantic relationships of this encoding from the respective authoritative definition of the CIDOC CRM are unintended transfer errors and not alternative definitions. The authoritative reference is the textual definition of the CIDOC CRM and not the RDF Schema. FORTH appreciates your feedback on such errors.
6. RDF does not support properties of properties. Therefore, users are recommended two ways to work around:

· The current properties of properties in the CRM have all as range “E55 Type”. Therefore they correspond to subtyping of the respective property by a local vocabulary.

· For the cases in which the local vocabulary is not fixed, there is a recommended form of reification via an auxiliary “property class”. This replaces the former recommendation to use E13 Attribute assignment in order to introduce user defined property types.

See section below on properties of properties about the pros and cons.

Implementing Datatypes

Among the entities information systems can refer to, one can distinguish those that cannot reside in an information system itself because of their nature, such as people, material objects, weather etc. and their interactions, from those that can have exhaustive representations in an information system, such as texts, numbers etc., because they are generically digital in nature[6]. Those that cannot reside in the information system, we can only refer to by identifiers. For those that can exhaustively be represented, all data models offer some elementary “built-in” datatypes, such as “integer”, “char”, “string”, “real”. They are recognized syntactically, and not by classification, and their identity is given by their “content” and their syntactic type. Consequently any occurrence of the same content is identical, regardless the context of reference[7].

Therefore, the CIDOC CRM, being an ontology and not a data model, refers to them in an abstract way as instances of “E59 Primitive Value”. To which degree they are actually “primitive” or elementary, or composed of more elementary ones, such as dates, is not the critical characteristic, but the fact, that their identity is completely defined by the syntactic type and digital content. In purpose, they match exactly with rdfs:Datatype.
The class E59 Primitive Value “comprises values of primitive data types of programming languages or database management systems and datatypes composed of such values used as documentation elements, as well as their mathematical abstractions”.
They are not considered as elements of the universe of discourse this model aims at defining and analysing. Rather, they play the role of a symbolic interface between the scope of this model and the world of mathematical and computational manipulations and the symbolic objects they define and handle.
In particular they comprise lexical forms encoded as "strings" or series of characters and symbols based on encoding schemes (characterised by being a limited subset of the respective mathematical abstractions) such as UNICODE and values of datatypes that can be encoded in a lexical form, including quantitative specifications of time-spans and geometry. They have in common that instances of E59 Primitive Value define themselves by virtue of their encoded value, regardless the nature of their mathematical abstractions. Therefore they should not be represented in an implementation by a universal identifier associated with a content model of different identity. In a concrete application, it is

recommended that the primitive value system from a chosen implementation platform and/or data definition language be used to substitute for this class and its subclasses”[8].

For encoding the CIDOC CRM in RDF this translates to trying to find suitable RDF datatypes, to the degree there is an equivalence with respective subclasses of E59 Primitive Value, or at least finding datatypes that represent wide enough value ranges for typical applications. It must however be understood that datatypes typically implement only subsets of values of respective mathematical spaces, whereas an ontological definition refers to the mathematical space itself. For instance, an “integer” value may be limited to 32 bits, which is “nearly nothing” against the unlimited size of natural numbers, but much more than most applications will ever encounter in object descriptions. Nevertheless, data referring to values outside of a particular datatype, but inside the respective mathematical space, must be regarded compatible with the CRM.

In principle, the literal encoding of mathematical values is unlimited, if the respective platform can represent unlimited literals. Consequently, rdfs:Literal is the superclass of all RDFS datatypes (“Each instance of rdfs:Datatype is a subclass of rdfs:Literal.” in: RDF Schema 1.1). Therefore, all properties of the CRM having a primitive value as range are compatible with using rdfs:Literal or an adequate datatype, as long as the meaning is compatible. Since this does not provide a particular guidance how to encode values nor any more formal constraints,we analyze in this document separately each subclass of E59 Primitive Value and the use of rdfs:label for their compatibility with RDFS datatypes and make **detailed recommendations**.
Nevertheless, applications may encounter cases in which no datatype recommended by RDFS or recommended below does fit the required value range. In that case it is recommended to find **other standards** to represent these values in an XSD-compatible form and to store them in an rdfs:Literal. More details are also given in the section “Defining custom datatypes”.

Recording dates

The range of the properties "P81 ongoing throughout" and "P82 at some time within" are defined in the CRM as E61 Time Primitive. Instance of E61 Time Primitive are defined as closed intervals on the natural time dimension in which we live.
Since the E61 Time Primitive of the CRM cannot be expressed in RDFS directly, in the official RDF implementation of the CIDOC CRM, we define four properties in the CRM RDFS: “P82a_begin_of_the_begin”, “P82b_end_of_the_end”, “P81a_end_of_the_begin”, “P81b_begin_of_the_end”, all with range **xsd:dateTime**, which replace "P81 ongoing throughout" and "P82 at some time within" of the CRM. For more details about the meaning of these four properties, see the guidelines in the Annex below.
Extremely old paleontological material and astronomic dates can be **below the range** of xsd:dateTime. If such dates need to be recorded, we recommend to discuss an extension with CRM-SIG.
All other RDF datatypes for time with more limited precision, such as years only (xsd:gYear), should not be used, because their interpretation either as duration or interval of indeterminacy causes significant implementation overhead at query time, whereas the properties P82a,P81a,P81b,P82b can express the same information unambiguously.

Notwithstanding, data entry forms may offer any simplification for specifying dates they want, and convert internally the representation into the recommended form. Ease of data entry is therefore no argument for the choice of a datatype.

Recording space

The recommended datatypes of RDF1.1 do not contain datatypes for describing geometric entities on the surface of earth. On the other side, they become increasingly important, and the CIDOC CRM version 6.2 on defines E94 Space Primitive, subclass of: E59 Primitive Value, as:

"This class comprises instances of E59 Primitive Value for space that should be implemented with appropriate validation, precision and references to spatial coordinate systems to express geometries on or relative to earth, or any other stable constellations of matter, relevant to cultural and scientific documentation.

An E94 Space Primitive defines an E53 Place in the sense of a declarative place as elaborated in CRMgeo (Doerr and Hiebel 2013), which means that the identity of the place is derived from its geometric definition. This declarative place allows for the application of all place properties to relate phenomenal places to their approximations expressed with geometries.

Definitions of instances of E53 Place using different spatial reference systems always result in definitions of different instances of E53 place approximating each other. It is possible for a place to be defined by phenomena causal to it, such as a settlement or a riverbed, or other forms of identification rather than by an instance of E94 Space Primitive. Any geometric approximation of such a place by an instance of E94 Space Primitive constitutes an instance of E53 Place in its own right, i.e., the approximating one.

Instances of E94 Space Primitive provide the ability to link CRM encoded data to the kinds of geometries used in maps or Geoinformation systems. They may be used for visualisation of the instances of E53 Place they define, in their geographic context and for computing topological relations between places based on these geometries.

E94 Space Primitive is not further elaborated upon within this model. Compatibility with OGC standards are recommended."

These standards currently do not have a common form comprising all others. Further, geometries defined with respect to particular object shapes, such as rotationally symmetric ones, are possibly open ended.

Therefore we define in the CRM RDFS the range of properties that use E94 Space Primitive in the definition of the CRM as rdfs:Literal, and recommend the user to instantiate it with adequate datatypes compatible with rdfs:Datatype. These are for the surface of Earth for example "geo:gmlLiteral" or "geo:wktLiteral"
(xmlns:geo="http://www.opengis.net/ont/geosparql#") or
. In order to accommodate for very large literals, see section "Very large Primitive Values" for additional definitions.

In the current version of the CIDOC CRM, only the property "*P168 place is defined by (defines place)*" has range E94 Space Primitive[9].

Since any instance of E94 Space Primitive identifies unambiguously an instance of E53 Place by a symbolic expression, E94 Space Primitive must logically be regarded as a

subclass of E41 Appellation. Consequently, we define *P168 place is defined by (defines place)* as subproperty of "*P1 is identified by*", and all literals used as its range instances implicitly as instances of E41 Appellation (see section "RDF implementation tests" item 1.). See also section "Recording Names".

Recording spacetime

Recording spacetime is very similar to recording space in all aspects: The recommended datatypes of RDF1.1 do not contain datatypes for describing spacetime volumes. Developing the CIDOC CRM and CRMgeo in particular, it appeared that all phenomena that can be named and can serve for determining by their spatial extent a place do also more or less change their spatial extent over time. If their maximal spatial extent is not sufficient for the purpose of documentation, the only consistent way to approximate these "places" is to approximate them by declarative spacetime volumes.
The CIDOC CRM version 6.2 on defines E94 Space Primitive, subclass of: E59 Primitive Value, as:
"This class comprises instances of E59 Primitive Value for spacetime volumes that should be implemented with appropriate validation, precision and reference systems to express geometries being limited and varying over time on or relative to earth, or any other stable constellations of matter, relevant to cultural and scientific documentation. A Spacetime Primitive may consist of one expression including temporal and spatial information like in GML or a different form of expressing spacetime in an integrated way like a formula containing all 4 dimensions.
An E95 Spacetime Primitive defines an E92 Spacetime Volume in the sense of a declarative spacetime volume as defined in CRMgeo (Doerr & Hiebel 2013), which means that the identity of the spacetime volume is derived from its geometric and temporal definition. This declarative spacetime volume allows for the application of all E92 Spacetime Volume properties to relate phenomenal spacetime volumes of periods and physical things to propositions about their spatial and temporal extents.
Definitions of spacetime volumes using different spacetime reference systems always result in definitions of different spacetime volumes approximating each other. It is possible for a spacetime volume to be defined by phenomena causal to it, such as an expanding and declining realm, a settlement structure or a battle, or other forms of identification rather than by an instance of E95 Spacetime Primitive. Any spatiotemporal approximation of such a phenomenon by an instance of E95 Spacetime Primitive constitutes an instance of E92 Spacetime volume in its own right, i.e., the approximating one. E95 Spacetime Primitive is not further elaborated upon within this model. Compatibility with OGC standards are recommended."

There are very few standardized formats for spacetime volumes. The most simple representations are a sort of 3D/4-D right prisms[10](geometry), in which the 2D /3D-base is a geometry kept constant over a time interval (the "height" of the prism). A more elaborate method is proposed by (Niccolucci & Hermon 2015), which uses aggregates of rectangular boxes for approximating irregular spacetime volumes.

Therefore we define in the CRM RDFS the range of properties that use E95 Spacetime Primitive in the definition of the CRM as rdfs:Literal, and recommend the user to instantiate it with adequate datatypes compatible with rdfs:Datatype. In order to accommodate for very large literals, see section “Very large Primitive Values” for additional definitions.

In the current version of the CIDOC CRM, only the property “*P169 defines spacetime volume (spacetime volume is defined by)*” has range E95 Spacetime Primitive.
Since any instance of E95 Spacetime Primitive identifies unambiguously an instance of E92 Spacetime volume by a symbolic expression, E95 Spacetime Primitive must logically be regarded as a subclass of E41 Appellation. . Consequently, we define “*P169 defines spacetime volume (spacetime volume is defined by)”* as subproperty of “*P1 is identified by*”, and all literals used as its range instances implicitly as instances of E41 Appellation (see section “RDF implementation tests” item 1.). See also section “Recording Names”.

Recording numbers

“Number” is a very general mathematical term. The CIDOC CRM version 6.2 on defines E60 Number, subclass of E59 Primitive Value, as:

“This class comprises any encoding of computable (algebraic) values such as integers, real numbers, complex numbers, vectors, tensors etc., including intervals of these values to express limited precision.

Numbers are fundamentally distinct from identifiers in continua, such as instances of E50 Date and E47 Spatial Coordinate, even though their encoding may be similar. Instances of E60 Number can be combined with each other in algebraic operations to yield other instances of E60 Number, e.g., 1+1=2. Identifiers in continua may be combined with numbers expressing distances to yield new identifiers, e.g., 1924-01-31 + 2 days = 1924-02-02. Cf. E54 Dimension””

In the CIDOC CRM, the class E60 Number appears only twice, as range of “E19 Physical Object: P57 has number of parts”, and as range of “E54 Dimension: P90 has value: E60 Number”.

In CRM RDFS, the range of “P57 has number of parts” should be xsd:nonNegativeInteger.

Due to the genericity of E54 Dimension, the range of “P90 has value” cannot be identified with a particular XSD datatype. The typical museum application of this class are the spatial dimensions of an object. In that case, and for all other linear dimensions, it is recommended to instantiate the range of “P90 has value” as xsd:double.

However, the class is relevant for describing the results of all kinds of measurements and other quantitative observations, which may use very complex representations of quantities. It is not in the scope of the CRM to develop exhaustive standards for these cases, because it is much more in the expertise of the respective natural sciences to define them. Respective communities of practice are invited to propose specializations of E54 Dimension and “P90_has_value”. For instance, sensor arrays, more and more in use, pose the issue of a single measurement resulting in an array of numbers which altogether form one quantitative statement about the observed. We can describe such structures easily as one complex type of unit (and define an IRI for it), and then regard the value to a matrix of numbers, in which

each position obeys subunits as defined in the complex unit type. In order to accommodate for very large literals, see section “Very large Primitive Values” for additional definitions.

Whereas the CRM regards that intervals of primitive values are primitive values by themselves, there is currently no corresponding practice in RDF. Therefore, in analogy to the properties of E52 Time-Span, we define in CRM RDFS two more subproperties of P90 has value: “P90a_has_lower_value_limit” and “P90b_has_upper_value_limit”. Even if we regard complex matrices of numbers as one value for an instance of E54 Dimension, such as RGB images, we can argue that minimal and maximal values exist as two separate matrices of the same structure. The precise guidelines for using these properties are given in the section “Guidelines for using P90a, P90, P90b” below.

Recording string values

The CIDOC CRM version 6.2 defines E62 String, subclass of E59 Primitive Value, as:

"This class comprises coherent sequences of binary-encoded symbols. They correspond to the content of an instance of E90 Symbolic object. Instances of E62 String represent only the symbol sequence itself. They may or may not contain a language code. In contrast, instances of other subclasses of E59 Primitive value represent entities in mathematical spaces different from that of symbol sequences, by using binary-encoded symbols, such as date expressions or numbers in decimal encoding. For instance, different syntactic forms of a date expression may represent the same date, but different strings."

E62 String appears in the CRM only as range of P3_has_note and its subproperties P79_beginning_is_qualified_by and P80_end_is_qualified_by, and in particularly in the newly proposed property “E90 Symbolic Object: has symbolic content”.

E62 String corresponds to rdfs:Literal, with the above described interpretation. Instantiation with rdf:langString and xsd:string is compatible.

Recording names

In the CRM names are modelled as instances of E41 Appellation. This class comprises any symbolic object used or created to name something without requiring further meaning. The CIDOC CRM version 6.2 defines E41 Appellation, subclass of E90 Symbolic Object, as: “This class comprises signs, either meaningful or not, or arrangements of signs following a specific syntax, that are used or can be used to refer to and identify a specific instance of some class or category within a certain context.

Instances of E41 Appellation do not identify things by their meaning, even if they happen to have one, but instead by convention, tradition, or agreement. Instances of E41 Appellation are cultural constructs; as such, they have a context, a history, and a use in time and space by some group of users. A given instance of E41 Appellation can have alternative forms, i.e., other instances of E41 Appellation that are always regarded as equivalent independent from the thing it denotes. “

The CRM is an ontology in the proper sense. Therefore, instances of physical things and phenomena of the physical worlds are regarded to be the things themselves, and not their machine representation, and any identifier or name used for something from the material world is different from the thing itself. For instance, I, Martin Doerr, am an instance of E21 Person, and not any of the URIs or records that may represent me in an information system. I am unique in this world, as is any particular thing, in contrast to representations of me. In the CRM, the property "P1 is identified by" from "E1 CRM Entity" to "E41 Appellation" relates the things to their names or identifiers.

In any knowledge representation schema, any item that cannot "reside" in the machine itself due to its nature, must be represented by one selected primary identifier, in the case of RDF by a URI. For an information system to be consistent with the described reality, these selected identifiers should map one-to-one to the ontological instances they stand for. Therefore, any instance of a class represented by a URI in RDF plays a dual role: it stands for the ontological instance and is an identifier for it (see also Meghini et al. 2014). For practical reasons, we do not represent this duality by a recursive use of "P1 is identified by" from an instance to itself in its second capacity as an identifier. However, all other names and identifiers are related to the select primary identifier via "P1 is identified by". This implies that the choice about which of multiple identifiers is the primary one may be changed without changing the meaning. In contrast, owl:same_as relates two primary URIs of things as different representation of the same real world thing, aggregating the properties of both representations as valid for the real world thing.

In practice, only the URIs, literals and datatypes "reside" themselves directly in a machine and need no additional identification because they are completely identified by their content. We may distinguish four different kinds of Appellations: URIs, identifiers from local application contexts, literally defined names used in human written communication and names from oral communication and tradition. Typically, URIs and local identifiers have a unique representation as strings. However, the situation for names is more complex. For instance, 北京 is a literally defined name for the capital of China. "Bei Jing" is meant to be an representation of the same name in Latin characters (underspecified without accent marks), and not meant to be another name for the same city. "Doerr is a respelling of Dörr, a German surname[11]". The most elaborate and effective good practice for registering proper names comes from the library community (Doerr, Riva and Zumer 2012). The FRBR Review Group of IFLA decided for practical reasons to identify a name ("Nomen" in their terminology) by the identical sequence of characters in a given script, not by the binary encoding.

For historical research however, in particular capturing oral tradition, this definition is too narrow, and we are confronted in relevant CRM applications with cases of names with spelling variants and even spoken variants. All cases of names that cannot uniquely be identified with a character sequence must be represented with a URI and further properties of description must be added, by preference the new property "*E90 Symbolic Object: P190 has symbolic content*". Also, if someone wants to document facts about a name other than its spelling, a URI must first be assigned, because a character string itself cannot be referred to in RDF. This case must not be confused with documenting facts about the relation

between a name and a particular carrier of that name, because that would be a reification of this relation, and not talking about the name.

Summarizing, there are two cases:
a) A name or identifier is completely defined and identified by a character sequence or any digitally, unambiguously encoded symbol.
b) A name or identifier is identified but not defined by a URI.
As a matter of fact, RDFS provides the property rdfs:label, which implements exactly the case a) above, without the possibility to add descriptions of the name itself. SKOS specializes rdfs:label into properties such as skos:prefLabel and skos:altLabel, which define indeed the names by which things are called by people. We take therefore the use of rdfs:label as existing good practice.

Consequently, we have to regard rdfs:label as a special case of “P1 is identified by”, and all literals used as range instances of rdfs:label implicitly as instances of E41 Appellation (see section “RDF implementation tests” item 1.).
Unfortunately, our KR languages have not foreseen the case that an instance of a datatype is also an instance of a user-defined class. This causes a range conflict, which can be overcome by “punning” the range of “P1 is identified by” to be both rdfs:Literal and E41 Appellation (see section “RDF implementation tests” item 2.).

This recommended implementation allows for using both models for Appellations, via an additional URI or directly as literal, and returning with one query all range instances of “P1 is identified by” following this interpretation. The SPARQL query result separates URIs from literals automatically. So, there is no ambiguity about the nature of the result.
Only if the same name is described both directly via rdfs:label and indirectly via a URI, the matching of both would need another query.
So, the frequently asked question remains, why not avoiding this double definition and describe any instance of E41 Appellation via another URI? The answer is that actually the cases that require explicit representation of E41 Appellation are relevant but rare. On the other side, good practice requires all nodes in a semantic graph represented by a URI to carry a human-readable label in addition. This means that the storage volume and query performance would heavily be hampered by such a “pure-logic-driven” decision.
The only ambiguity that remains is the case in which the instance of Appellation is literally the URI itself, and not a URI representing an Appellation of different form. There are two solution to this problem: Either classify this URI by the class of things it identifies and use owl:same_as, or we define a specific subclass of E41 Appellation “URI”.

Language of an Appellation

Whereas common words always belong to a language, proper names normally do not belong to a language. For persons, they are normally used in the form the carrier of the name uses it, and for place names in the form the local population uses it. The latter place names are called “vernacular”. Only important and historical places use to have name variants in use in other language groups. The same holds for some meaningful titles of paintings, and translations of books, movies etc. Even specialized terms, even though not

being proper names, often are not translated. The “language” of such names is more a useful distinction for the user to recognize and distinguish the target group of a label. **We therefore recommend** the use of *rdfs:langString* for all Appellations being regarded specific to or characteristic for a language group and being directly described by a literal and not indirectly via a URI. For Appellations being described indirectly via a URI, we recommend *the use of E41 Appellation > P72 has language > E56 Language*. This holds also in the case the language to be documented is not among those that can be specified by *rdfs:langString.* For convenience, the next version of the CRM RDFS will contain the class *“E41_E33_Linguistic Appellation”,* sublass of *E41 Appellation* and *E33 Linguistic Object.*

Very large Primitive Values

In general, representations of primitive values do not have a size limit, except for time expressions. In particular, geometries may be very large polygon sequences or other large datasets, as well as arrays of numbers from scientific data.

Very large strings one would normally describe in a file and instantiate E90 Symbolic Object or a subclass of it with the URL. However, the question is, if the URL would indeed be a good persistent identifier, since the URL stands for a physical location, albeit indirectly addressed. The Linked Open Data community has not yet given satisfactory answers for the long term validity of resolvable URIs. If the URL is not a good identifier, another, primary URI should be chosen, and the content found under this URL should be related to the primary URI as a representative of the content of the symbolic object identified with the primary URI. **We recommend** for this relation a specific property to be decided by the CRM Special Interest Group, either specific to CRM RDF, or in CRMbase.

In the case of the other Primitive Values, except for time, **we recommend** a “punning” solution: The properties that have as range a Primitive Value in the CRM should be defined in CRM RDF to have as range rdfs:Literal **and** the respective subclass of E59 Primitive Value:

· *P90 has value*: has range both rdfs:Literal and has range E60 Number”.

· *P168 place is defined by (defines place)*” has range both rdfs:Literal and has range E94 Space Primitive

· *P169 defines spacetime volume (spacetime volume is defined by)* has range both rdfs:Literal and has range E95 Spacetime Primitive

The respective subclass should **only be instantiated** with a URL of a file containing the content in the case the content does **not fit** well in an rdfs:Literal. See examples in the section “RDF implementation tests” item 4.

Defining custom datatypes

New datatypes can be defined, published in a respective namespace and added to the RDFS datatypes, for instance using a cidoc crm namespace. The section “RDF implementation tests” item 3 shows how a cidoc crm space primitive can be defined as datatype.

Any string of a datatype is stored in a triple store as a literal. If the datatype is a compound value, such as xsd:dateTime, there are specific functions that are in reality String Functions which can isolate the different parts, for instance that of a date (year, month etc), at query time, and makes them accessible to be specified as query elements. See “RDF implementation tests” item 5.

The alternative, to define for each kind of compound value a series of subproperties of P_has_value, makes data entry, data display and computation with these values much more complex. We do not recommend this solution.

This means that it is up to the designer of these functions to define a convenient syntax within the respective literal, and of course a question of standardization. The respective String Functions are either compiled into the query software, or invoked by code that runs on query results. The latter is much easier to handle by users, if they have no IT support to embed the code in the query system. For our purposes, most custom datatypes need not be broken up into its parts by the query system itself, because they will be interpreted anyway after querying, often just by the user reading them.

We recommended users to find respective custom datatypes and their syntax at the communities of practice dealing most with these kinds of values and to propose them to the CIDOC CRM Special Interest Group for approval.

Properties of properties

As mentioned above, RDF does not support properties of properties. Therefore, users are recommended two ways to work around:

A. The current properties of properties in the CRM have all as range “E55 Type”. Therefore they correspond to subtyping of the respective property by a local vocabulary.

B. For the cases in which the local vocabulary is not fixed, there is a recommended form of reification via an auxiliary “property class”. This replaces the former recommendation to use E13 Attribute assignment in order to introduce user defined property types.

The two solutions have pros and cons with respect to query performance, user interface programming and flexibility to cater for a local, evolving terminology.

Solution A:

Users that have fixed vocabularies of property types for those properties foreseeing in the CRM a “type” or “Pxx.1” property, may transform these types into their own subproperties for the respective CRM properties, such a as "P3 has note":

Instead of P3 has note (P3.1 has type : parts description) declare

```
<rdf:Property rdf:about="P3_parts_description">
        <rdfs:domain rdf:resource="E1_CRM_Entity"/>
        <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
        <rdfs:subPropertyOf rdf:resource="P3_has_note"/>
</rdf:Property>
```

This ensures that a graph using these subproperties can consistently be retrieved via their superproperties. In other terms, a system using this solution a) is query compatible with the CIDOC CRM without using properties of properties in the query and b) it ensures that typing

a property instantiates the base property and therefore the complete graph is contained in the respective query answer. It is the most **efficient implementation**, both in terms of **query performance and storage** overhead. User interface programmers should query these additional subproperties and create **at run-time** selection lists of them, rather than hard-coding the vocabularies. If such extensions are widely built, they can **reveal an emerging** good practice and become subject to a standardization of its own. The drawback is that the vocabulary must be loaded to database platform before-hand, and this mechanism only applies to types of properties. It is the **preferred solution**.

In case users have no other choice than to deal with open vocabularies of property types, or would need extensions with properties of properties other than types, they should resort to solution B. This solution uses “property classes”, i.e., representing an n-ary property by an auxiliary RDF class, which are provided together with the CRM RDF Schema[12]. This solution is logically adequate and further extensible, but is more complex with respect to query formulation, has considerably slower query response times in current knowledge base platforms than the above and more storage overhead. It is more obvious for user interface programmers to create the respective selection lists, and users may introduce new types at runtime.

About implementing multiple Instantiation

Knowledge Representation models can assign multiple classes to a given instance identifier. After that, all properties of each assigned class are applicable for this identifier. This construct is called “multiple instantiation. For instance, a calligraphy is an “image” and a “linguistic object”, having a language and a painting style. This is not possible with Relational data structures, because instance identification is limited to the entity (class) or with XML-like data structures, because instance identification is by structural position (additional identifiers can be used for linking).

Therefore many users are not aware of this feature, and even KR tools do not systematically guide users to use it: Once an instance is classified by one class, the tool should not allow for using a property of another class, but most likely will not advise the user that she could add the additional class to the instance. Nevertheless, it is a key feature of KR models that facilitating modularizing ontologies and the often advertised ability to combine different ontologies.
The CRM as ontology relies heavily on multiple instantiation: Combination of classes that are applicable to some instances only incidentally and have no properties specific to this combination are not modelled in the CRM individually as subclasses of multiple parent classes. The latter would be called “multiple IsA”. To avoid multiple IsA in such cases is an important normalization principle to keep the ontology very compact and unambiguous.
In the specification modules of mapping software used to transform data into a CRM-compatible form, care must be taken to foresee and allow the user to combine RDF classes systematically.
Some combinations of classes may more frequently occur, such as combining E41 Appellation with E33 Linguistic Object in order to reach E56 Language via P72 has language. In a local system that does not easily support multiple instantiation, the candidate cases for multiple instantiation may be combined in subclasses using multiple IsA. For their

labels, we recommend to aggregate the class identifier codes as in: “E41_E33_Linguistic Appellation”. Such a replacement is query compatible with the standard. A respective import/export system simply needs to make the trivial replacements of the respective class combinations with their multiple IsA counterparts and vice-versa in order to achieve import/export compatibility.
Users may provide feedback about frequent cases where multiple instantiation is used, in order to guide users to these modelling cases. These could systematically be entered into the CRM RDF implementation, without requiring the CRM standard itself to repeat them.

CIDOC CRM and other frameworks

The CIDOC CRM generally foresees to be used together with other ontologies and respective implementations for all domains that fall outside the scope of the CRM and for which an active, internationally acknowledge community exist that maintains the respective ontology. It is not the intention of the CIDOC CRM to compete with communities that have superior domain knowledge, but rather to benefit from their insight, given the ontology has been created with a compatible methodology or diligence. The CRM may deliberately reduce its scope in favour of such a community.

This theoretical principle finds in practice the following obstacles: The respective domain ontology will most likely define general concepts that overlap in an incompatible way with the CRM. For instance, OGC defines a few fundamental concepts differently analysed in the CRM, and many concepts the CRM never intends to deal with, but wants to recommend their use (see Doerr, Hiebel and Eide 2013).

This problem can be solved by a so-called “articulation”. Overlapping concepts are redefined and new concepts are introduced to create a more detailed model of the overlapping area which can be mapped to both ontologies. Users must replace the incompatible parts of both ontologies with the refined model, and use all other concepts of both ontologies together with it. The CIDOC CRM SIG aims at adapting the CRM to important domain ontologies by adopting the refined model.

Such a model is **CRMgeo**, linking the CIDOC CRM with **OGC standards** (Doerr, Hiebel and Eide 2013, Doerr and Hiebel 2013). I.e., OGC standards can be used, except for those concepts redefined in CRMgeo, together with CRMgeo and CRM “base”.
The bibliographic “**FRBR Family of models**” by IFLA on the other side was formulated as Entity-Relationship model, a methodology incompatible with the CRM. Therefore, both communities, CIDOC and IFLA, have engaged in a compatible, complete reformulation of the FRBR models, now “Library Reference Model (**LRM**)” as the CRM-compatible ontology FRBRoo version 1-3, version 3 now renamed to “LRMoo”.

SKOS[13] is an RDF schema originally designed to describe terminologies of universals of entities. *E55 Type* may in general refer to even less formal systems of terminology than SKOS and it is also used in the CRM to refer to property types. Therefore, **it is recommended** to define *E55 Type* as superclass of skos:Concept and *P127_has_broader_term* as superproperty of skos:broader / *P127i_has_narrower_term* as superproperty of skos:narrower. It is **not recommended** and incompatible with the CRM **to**

use skos:Concept for **places and persons**. See also LRMoo about the distinction between natural persons and literary characters derived from those.

The Dublin Core Metadata Element Set, Version 1.1, has limited compatibility. The properties, dc:relation and dc:date are underspecified, and their use leads to ambiguous overlaps with CRM-based descriptions. The properties dc:publisher, dc:creator, dc:contributor, dc:source may be interpreted as shortcuts of CRM properties, but lack the important intermediate events. It is not recommended to combine DC with the CRM. Alternative, separate descriptions of things with The Dublin Core Metadata Element Set are, of course, no problem.

The compatibility of other frameworks with the CRM needs to be investigated. The CRM SIG will be glad to receive request and collaborate with respective initiatives.

References

Doerr, Martin & Riva, Pat & Žumer, Maja. (2012). *FRBR Entities: Identity and Identification*. In: Cataloging & Classification Quarterly. Volume 50, 2012 - Issue 5-7: The FRBR Family of Models. Pages 517-541 DOI: 10.1080/01639374.2012.681252.

Hiebel, G.H, Doerr, M., & Eide, Ø. (2013). Integration of CIDOC CRM with OGC Standards to model spatial information (Session5, 522). *Computer Applications and Quantitative Methods in Archaeology (CAA) 2013*, Perth-Australia, 25th -28th March 2013. (pdf).

Doerr, M., & Hiebel, G.H (2013). CRMgeo: Linking the CIDOC CRM to GeoSPARQL through a Spatiotemporal Refinement. 2013.TR435_CRMgeo_CIDOC_CRM_GeoSPARQL.pdf

Franco Niccolucci, Sorin Hermon (2015) "*Representing gazetteers and period thesauri in four-dimensional space–time*", Published 2015 in International Journal on Digital Libraries, DOI:10.1007/s00799-015-0159-x

Meghini, C., Spyratos, N., Sugibuchi, T. and Jitao Yang. (2014). *A Model for Digital Libraries and its Translation to RDF*. In: Journal on Data Semantics, June 2014, Volume 3, Issue 2, pp 107–139. https://doi.org/10.1007/s13740-013-0029-x

Meghini, C. and Doerr, M., 2018, 'A first-order logic expression of the CIDOC Conceptual Reference Model', International Journal of Metadata, Semantics and Ontologies.

Annex

Commented overview of RDFS datatypes

	Datatype	Value space (informative)	CRM recommendation	Comment
Core types	xsd:string	Character strings (but not all Unicode character strings)	IsA E62 String and default.	E62 may contain more kinds of symbols/scripts, such as xsd:hexBinary or Linear B
	xsd:boolean	true, false	IsA I6 Belief Value. Do not use.	Only belief values in CRMInf may use these values, but they should at least be three-valued: True, False, Unknown.
	xsd:decimal	Arbitrary-precision decimal numbers	Do not use	
	xsd:integer	Arbitrary-size integer numbers	IsA E60 Number	
IEEE floating-point numbers	xsd:double	64-bit floating point numbers incl. ±Inf, ±0, NaN	IsA E60 Number	
	xsd:float	32-bit floating point numbers incl. ±Inf, ±0, NaN	IsA E60 Number	
Time and date	xsd:date	Dates (yyyy-mm-dd) with or without timezone	IsA E61 Time Primitive, do not use	It could be used for P81,P82, but only for intervals of days. That does not seem to make much sense. It must not be used for P81a,P81b, P82a, P82b
	xsd:time	Times (hh:mm:ss.sss…) with or without timezone	Do not use	
	xsd:dateTime	Date and time with or without timezone	Use pairwise for E61 Time Primitive	Use for P81a,P81b, P82a, P82b.

	xsd:dateTimeStamp	Date and time with required timezone	Use pairwise for E61 Time Primitive	Use for P81a,P81b, P82a, P82b.
Recurring and partial dates	xsd:gYear	Gregorian calendar year	Do not use	
	xsd:gMonth	Gregorian calendar month	Do not use	
	xsd:gDay	Gregorian calendar day of the month	Do not use	
	xsd:gYearMonth	Gregorian calendar year and month	Do not use	
	xsd:gMonthDay	Gregorian calendar month and day	Do not use	
	xsd:duration	Duration of time	IsA E54 Dimension,	use for P83 had at least duration (was minimum duration of): E54 Dimension P84 had at most duration (was maximum duration of): E54 Dimension
	xsd:yearMonthDuration	Duration of time (months and years only)	See above	
	xsd:dayTimeDuration	Duration of time (days, hours, minutes, seconds only)	See above	
Limited-range integer numbers	xsd:byte	-128…+127 (8 bit)	Do not use	
	xsd:short	-32768…+32767 (16 bit)	Do not use	
	xsd:int	-2147483648…+2147483647 (32 bit)	Do not use	
	xsd:long	-9223372036854775808…+9223372036854775807 (64 bit)	Do not use	
	xsd:unsignedByte	0…255 (8 bit)	Do not use	
	xsd:unsignedShort	0…65535 (16 bit)	Do not use	
	xsd:unsignedInt	0…4294967295 (32 bit)	Do not use	
	xsd:unsignedLong	0…18446744073709551615 (64 bit)	Do not use	
	xsd:positiveInteger	Integer numbers >0	May be used in extensions.	
	xsd:nonNegativeInteger	Integer numbers ≥0	IsA E60 Number.	Use for P57 has number of parts. It may be useful to distinguish zero parts

				from not knowing parts.
	xsd:negativeInteger	Integer numbers <0	Do not use	
	xsd:nonPositiveInteger	Integer numbers ≤0	Do not use	
Encoded binary data	xsd:hexBinary	Hex-encoded binary data	IsA E62 String.	Note, that it represents bits and not the hex symbols. Can be useful for content models.
	xsd:base64Binary	Base64-encoded binary data	Do not use	
Miscellaneous XSD types	xsd:anyURI	Absolute or relative URIs and IRIs		
	xsd:language	Language tags per [*BCP47*]	IsA E56 Language	Since there are many more historical languages than xsd:language comprise, we may better use it as rdf:label or identifier for those covered by xsd:language.
	xsd:normalizedString	Whitespace-normalized strings	IsA E62 String	
	xsd:token	Tokenized strings	IsA E62 String	
	xsd:NMTOKEN	XML NMTOKENs	Do not use	
	xsd:Name	XML Names	Do not use	
	xsd:NCName	XML NCNames	Do not use	

Guidelines for using P82a, P82b, P81a, P82b

Jan 7, 2018

The range of the properties "P81 ongoing throughout" and "P82 at some time within" are defined in the CRM as E61 Time Primitive. Instance of E61 Time Primitive are defined as closed, contiguous intervals on the natural time dimension in which we live. “Closed” means that the endpoints belong to the interval. “Contiguous” means that there are no gaps between the endpoints in the interval (which holds for “intervals” in general).

The reason to describe time spans with inner and outer intervals is the existence of a very efficient algebra for calculating resulting areas of determinacy and indeterminacy [Plexousakis et al.XXXX]. Further, they are motivated by the British MIDAS Heritage standards [https://en.wikipedia.org/wiki/MIDAS_Heritage] and easy to define in Relational databases.

Since the E61 Time Primitive of the CRM cannot be expressed in RDF directly, in the official RDF implementation of the CIDOC CRM, we define four properties replacing P81 and P82 adequately using xsd:dateTime.

P81 ongoing throughout

Property P81 describes the maximum known temporal extent of an E52 Time-Span, i.e. the extent it is ongoing throughout. It is replaced in this RDF version by the property "P81a_end_of_the_begin" and "P81b_begin_of_the_end", to be used together.

"P81a_end_of_the_begin" should be instantiated as the earliest point in time the user is sure that the respective temporal phenomenon is indeed ongoing. We call it “end_of_the_begin”, because it also constitutes an upper limit to the end of the indeterminacy or fuzziness of the beginning of the described temporal phenomenon.

"P81b_begin_of_the_end" should be instantiated as the latest point in time the user is sure that the respective temporal phenomenon is indeed ongoing. We call it “begin_of_the_end”, because it also constitutes a lower limit to the begin of the indeterminacy or fuzziness of the end of the described temporal phenomenon.

It is correct to assign the same value to “P81a_end_of_the_begin” and “P81b_begin_of_the_end”, if no other positive knowledge exists. It is also correct not to instantiate P81 for a time span, if there is no evidence that the temporal phenomenon was definitely occurring at a particular time.

If a respective reasoning is installed, and no evidence exists about the point in time that the phenomenon was definitely ongoing, one may specify “P81a_end_of_the_begin” as being later than “P81b_begin_of_the_end”, indicating that the indeterminacy of knowledge (not of being) of the begin overlaps with the indeterminacy of knowledge (not of being) of the end [see Christian-Emil Ore XXX].

If a value for “P81a_end_of_the_begin” is given with a precision less than that of xsd:dateTime (i.e. seconds), such as in days or years, the implementation should “round it up” to the last instant of this time expression, e.g. 1971 = Dec 31 1971 23:59:59. Respectively, for “P81b_begin_of_the_end” the implementation should “round it down”, e.g. 1971 = Jan 1 1971 0:00:00. If values are needed that are not within the range or precision of

xsd:dateTime, e.g., for paleontology, this property should be extended with another, suitable data type.

P82 at some time within

Property P82 describes the narrowest known outer bounds of the temporal extent of an E52 Time-Span, i.e. that the described temporal phenomenon is ongoing “at some time within” this interval. It is replaced in the official RDF version by the properties "P82a_begin_of_the_begin" and "P82b_end_of_the_end", to be used together.
"P82a_begin_of_the_begin" should be instantiated as the latest point in time the user is sure that the respective temporal phenomenon is indeed not yet happening. We call it “begin_of_the_begin”, because it also constitutes a lower limit to the beginning of the indeterminacy or fuzziness of the begin of the described temporal phenomenon.
"P82b_end_of_the_end" should be instantiated as the earliest point in time the user is sure that the respective temporal phenomenon is indeed no longer ongoing. We call it “end_of_the_end”, because it also constitutes an upper limit to the end of the indeterminacy or fuzziness of the end of the described temporal phenomenon.
It is not correct to assign the same value to “P82a_begin_of_the_begin” and “P82b_end_of_the_end”. If a value for “P82a_begin_of_the_begin” is given with a precision less than that of xsd:dateTime (i.e. seconds), such as in days or years, the implementation should “round it down” to the first instant of this time expression, e.g. 1971 = Jan 1 1971 0:00:00. Respectively, for “P82b_end_of_the_end” the implementation should “round it up”, e.g. 1971 = Dec 31 1971 23:59:59.
It must always hold that “P82a_begin_of_the_begin” is before “P82b_end_of_the_end”, “P81a_end_of_the_begin” and “P81b_begin_of_the_end”.
It must always hold that “P82b_end_of_the_end” is after “P82b_end_of_the_end”, “P81a_end_of_the_begin” and “P81b_begin_of_the_end”.
“P82a_begin_of_the_begin” and “P82b_end_of_the_end” should always be assigned a value for any past phenomenon. The scholarly practice of not giving outer bounds for an event, because they are not known down to a desired precision (e.g. of three years), is not helpful for automated reasoning. In that case, the machine may conclude that a historical event could have happened at the time of the dinosaurs. Therefore any value is better than no value, even if it is relatively far away from the most likely value. It is an error to associate any implicit degree of approximation with these values. Only for phenomena that may not yet have ended at the time of documentation the end of the time-span should not be specified.

Guidelines for using P90a, P90, P90b

The CRM recommends to approximate numerical values of Dimensions with intervals. The range of the respective property "P90 has value" is defined in the CRM as E60 Number. Whereas the CRM regards that intervals of primitive values are primitive values by themselves, there is currently no corresponding practice in RDF. Therefore, in analogy to the properties of E52 Time-Span, we define in CRM RDFS two more subproperties of P90 has value: “*P90a_has_lower_value_limit*” and “*P90b_has_upper_value_limit*”.

The reasons for recommending this approximation are the following: All scientific measurements of non-discrete values are imprecise because of the tolerances of the measurement devices, shortcomings in applying the procedures and the indeterminacy of the measured effect itself. In natural sciences, important results of measurements are associated with possibly complex probabilistic distributions for the true value of the measured effect.

The most complex case relevant for cultural-historical data are the so-called “battleship curves” for calibrated C14 dating data. Many of these distribution models actually extend to infinity with non-zero probability, which is neither practical nor always justified. In the case of C14 however, the actual width of the distribution is often underestimated. Nevertheless, even data with a given probabilistic uncertainty to infinity are typically associated by scientists with narrower “confidence intervals” at one to three “standard deviations”, i.e., with a probability of some 68% – 99.7% for the value to be in the given range (https://en.wikipedia.org/wiki/Standard_deviation).

Whereas querying globally a very large aggregation of cultural-historical data by time intervals is highly relevant, querying and reasoning with different approximations of dimensions is normally restricted to quite narrow questions. For many cases, a medium value without explicit limits is sufficient for the application, such as the length of a museum object in millimeters for packaging it in a box. Nevertheless, querying explicit representation of actual outer limits or at least reasonably wide confidence intervals is computationally highly effective, and therefore a good way to ensure recall at query time, i.e., that the relevant results are contained in the answer to the query, even if it also contains irrelevant ones.

We therefore recommend to use *P90_has_value* for documenting a medium value or a value without error estimates, when the precision appears to be self-evident or irrelevant.

We recommend to use *P90a_has_lower_value_limit* for documenting the highest explicit lower limit available for the respective value, even if it provides very wide margins. It is an error to omit the lower limit even if it appears to be overly pessimistic.

We recommend to use *P90b_has_upper_value_limit* for documenting the lowest explicit upper limit available for the respective, even if it provides very wide margins. It is an error to omit the upper limit even if it appears to be overly pessimistic.

In case of approximating probabilistic distributions, we recommend to keep lower and upper limit at two standard deviations or enclosing the true value with 95% probability.

P90a_has_lower_value_limit should always be used together with *P90b_has_upper_value_limit.* If they are used, the property *P90_has_value* may be used as well or be omitted.

RDF implementation tests

1. rdfs:label as subproperty of P1 is identifiedby:

<rdf:Property rdf:about=**="http://www.w3.org/2000/01/rdf-schema#label">**
<rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
<rdfs:range rdf:resource=" http://www.w3.org/2000/01/rdf-schema#Literal "/>
<rdfs:subPropertyOf rdf:resource="P1_is_identified_by"/>
</rdf:Property>

Query (Give me all the superproperties of rdfs:label) :

```
select * where {
rdfs:label rdfs:subPropertyOf ?p
}
```

Result from Virtuoso:

p:
http://www.cidoc-crm.org/cidoc-crm/P1_is_identified_by

2. Adding rdf:Literal as range of P1 is identified by:

The cidoc_crm.rdfs was altered to include the following:

<rdf:Property rdf:about="P1_is_identified_by">
<rdfs:label xml:lang="en">is identified by</rdfs:label>
<rdfs:domain rdf:resource="E1_CRM_Entity"/>
<rdfs:range rdf:resource="E41_Appellation"/>
</rdf:Property>
<rdf:Property rdf:about="P1_is_identified_by">
<rdfs:label xml:lang="en">is identified by</rdfs:label>
<rdfs:domain rdf:resource="E1_CRM_Entity"/>
<rdfs:rangerdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

The cidoc crm schema was uploaded in virtuoso and the following query (give me the range of P1_is_identified_property) was executed to be sure that the changes have been applied:

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select * where { crm:P1_is_identified_by rdfs:range ?range}
```

result:

	range
	http://www.cidoc-crm.org/cidoc-crm/E41_Appellation
	http://www.w3.org/2000/01/rdf-schema#Literal

So, it is confirmed that the two ranges have been added. We repeat at this point that Virtuoso **does not apply** any semantic validation. The purpose of this test is to prove that this exercise is possible even though conceptually it may not be correct.

Data example:

1. The ttl data that was presented previously has been added in virtuoso:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix crm: <http://www.cidoc-crm.org/cidoc-crm/> .

<http://example.com/person/alexander_the_great>
crm:P1_is_identified_by <http://example.com/appellation/alexander_the_great> .

<http://example.com/appellation/alexander_the_great>
rdfs:label  "Alexander the Great" .

<http://example.com/person/alexander_the_great>
 rdfs:label  "Alexander the Great" .

<http://example.com/person/alexander_the_great>
crm:P1_is_identified_by  "Alexander the Great" .
```

2. A query to return all the "identifiers" of alexander the great using the is identified property was applied:

```
prefix crm: <http://www.cidoc-crm.org/cidoc-crm/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select * where
{ <http://example.com/person/alexander_the_great>  crm:P1_is_identified_by ?identifier }
```

result:

identifier
http://example.com/appellation/alexander_the_great
Alexander the Great

3. Defining a CIDOC CRM custom datatype:

We need a cidoc crm namespace. An initial suggestion would be the following:
Prefix: cdt: http://www.cidoc-crm.org/cidoc-crm/datatypes/

Namespace for space primitive : **cdt:space_primitive**

What needs to be defined in cidoc crm rdfs to create the new cidoc crm datatypes?
The class rdfs:Literal is the class of literal values such as strings and integers. Property values such as textual strings are examples of RDF literals. rdfs:Literal is an instance of rdfs:Class. rdfs:Literal is subclass of rdfs:Resource. rdfs:Datatype is the class of datatypes.
All instances of rdfs:Datatype correspond to the RDF model of a datatype rdfs:Datatype is both an instance of and a subclass of rdfs:Class. Each instance of rdfs:Datatype is a subclass of rdfs:Literal.
So, the cidoc crm datatypes must be instances of rdfs:Datatype that needs to be a subclass of rdfs:Literal that is a subclass of rdfs:Class and **each instance of rdfs:Datatype must be a subclass of rdfs:Literal.**
The addition to cidoc crm rdfs is the following (using the example of space primitive):

```
<rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Literal">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Literal</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
```

```
</rdfs:Class>
<rdfs:Class rdf:about="http://www.w3.org/2000/01/rdf-schema#Datatype">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Datatype</rdfs:label>
  <rdfs:comment>The class of RDF datatypes.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdfs:Class>
<rdfs:Class rdf:about="http://www.cidoc-crm.org/cidoc-crm/datatypes/space_primitive">
  <rdfs:isDefinedBy rdf:resource="http://www.w3.org/2000/01/rdf-schema#"/>
  <rdfs:label>Datatype</rdfs:label>
  <rdfs:comment>The class of RDF datatypes.</rdfs:comment>
  <rdfs:subClassOf rdf:resource=" http://www.w3.org/2000/01/rdf-schema#Literal "/>
</rdfs:Class>
```

And afterwards there needs to be the information in the triple store (or in RDF data in general) that the cidoc – crm datatype is an instance of rdfs:datatype.

```
<rdf:Description rdf:about="http://www.cidoc-crm.org/cidoc crm/datatypes/space_primitive">
<rdf:type resource=" http://www.w3.org/2000/01/rdf-schema#Datatype"/>
</rdf:Description>
```

And then the following query (select * datatypes) was executed to validate the work done:

```
select ?dt where {
?dt a <http://www.w3.org/2000/01/rdf-schema#Datatype> .
?dt rdfs:subClassOf <http://www.w3.org/2000/01/rdf-schema#Literal>
}
```

result**[14][15][16]**:

?dt
http://www.cidoc-crm.org/cidoc-crm/datatypes/space_primitive

Now we use this datatype to describe a birthplace:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
xmlns:crm="http://www.cidoc-crm.org/cidoc-crm/">
<rdf:Description rdf:about="http://example.com/actor/rob">
  <rdf:type rdf:resource="http://www.cidoc-crm.org/cidoc-crm/E21_Person"/>
    <rdfs:label>Rob</rdfs:label>
                    <crm:p98i_was_born>
          <crm:E67_Birth rdf:about="http://example.com/event/rob_birth">
            <crm:p7_took_place_at>
          <crm:E53_Place rdf:about="http://example.com/place/rangiora">
                  <rdfs:label>Rangiora</rdfs:label>
                  <crm:P168_place_is_defined_by rdf:datatype="http://www.cidoc-crm.org/cidoc-
crm/datatypes/space_primitive">
 "POLYGON((172.565456 -43.285409, 172.622116 -43.285409, 172.622116 -43.323697,  172.565456 -
43.323697, 172.565456 -43.285409))"
                      </crm:P168_place_is_defined>
          </crm:E53_Place>
                </crm:p7_took_place_at>
          </crm:E67_Birth>
                  </crm:p98i_was_born>
                  </rdf:Description>
</rdf:RDF>
```

3. Instantiating E94 with a file:

In the this version of the above example the place is defined by a shape file:

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
xmlns:crm=http://www.cidoc-crm.org/cidoc-crm/>
<rdf:Description rdf:about="http://example.com/actor/rob">
 <rdf:type rdf:resource="http://www.cidoc-crm.org/cidoc-crm/E21_Person"/>
  <rdfs:label>Rob</rdfs:label>
   <crm:p98i_was_born>
          <crm:E67_Birth rdf:about="http://example.com/event/rob_birth">
          <crm:p7_took_place_at>
            <crm:E53_Place rdf:about="http://example.com/place/rangiora">
                  <rdfs:label>Rangiora</rdfs:label>
                  <crm:P168_place_is_defined_by>
```

```
        <crm:E94_Space_Primitive rdf:about="http://example.com/file/rangiora.shp">
        <rdfs:label>Rangiora.shp</rdfs:label>
        <crm:p2_has_type rdf:resource="http://example.com/type/ESRIshapefile"/>
                </crm:E94_Space_Primitive >
                </crm:P168_place_is_defined_by>
      </crm:E53_Place>
                 </crm:p7_took_place_at>
                </crm:E67_Birth>
                </crm:p98i_was_born>
        </rdf:Description>
</rdf:RDF>
```

---

[1]http://www.cidoc-crm.org/sites/default/files/2017-09-30%23CIDOC%20CRM_v6.2.2_esIP.pdf

[2] The metamodel of the CIDOC CRM is actually one of the variants of knowledge representation models that uses generalization / specialization constructs as object-oriented models do. An exact definition can be found in (Meghini & Doerr 2018).

[3] http://www.cidoc-crm.org/sites/default/files/cidoc_crm_v6.2-draft-2015August.rdfs

[4] The only classes it does not define are the class E59 Primitive Value and its subclasses. How these are implemented will be described later in this text

[5] Insert URL of OWL version

[6] A “digital surrogate”, such as a 3D model of an object, must not be confused with the real thing it depicts. It does not bring the real thing into a machine.

[7] This does not strictly hold for texts. The problem of text identity is discussed in the section “Recording String Values”.

[8] This is the scope note of E59 Primitive Value of the CIDOC CRM version 6.

[9] The concepts E47 Spatial Coordinates,crmgeo: SP5 Geometric Place Expression, crmgeo:Q10 defines place and P168 place is defined by (defines place) will be revised soon. E94 Space Primitive should replace E47 Spatial Coordinates and SP5 Geometric Place Expression. P168 place is defined by (defines place) should replace Q10 defines place. It may be useful in the CRM RDFS to specify two subproperties of P168, one having as range “geo:wktLiteral” and another “geo:gmlLiteral” (xmlns:geo="http://www.opengis.net/ont/geosparql#”)

.

[10] https://en.wikipedia.org/wiki/Prism_(geometry)

[11] https://en.wikipedia.org/wiki/Doerr

[12] http://www.cidoc-crm.org/sites/default/files/CRMpc_v1.1_0.rdfs

[13] https://www.w3.org/2004/02/skos/

[14] https://www.w3.org/TR/swbp-xsch-datatypes/

[15] https://www.w3.org/TR/rdf11-concepts/

[16] http://infolab.stanford.edu/~melnik/rdf/datatyping/