Guidelines for using the PREMIS Version 3 OWL Ontology

Acknowledgments

The PREMIS Editorial Committee appointed a working group to revise the previous ontology consisting of the following members:

Charles Blair (University of Chicago)
Lina Bountouri (Publications Office of the European Union)
Bertrand Caron (Bibliothèque nationale de France)
Esmé Cowles (Princeton University)
Angela DiIorio (Sapienza Università di Roma)
Rebecca Guenther (Consultant, Library of Congress)
Evelyn McLellan (Artefactual Systems)
Elizabeth Russey Roke (Emory University)

Introduction

PREMIS is based on a data model that defines the entities that are described (Objects, Events, Agents and Rights), the properties of those entities (semantic units), and relationships between them. This PREMIS OWL ontology provides an RDF encoding reflecting that model. This document is based on the PREMIS Data Dictionary version 3 (http://www.loc.gov/standards/premis/v3/premis-3-0-final.pdf). Readers should refer to it for the definitions of PREMIS terms.

These guidelines aim only to provide an endorsed expression of the PREMIS Data Dictionary version 3.0 and its data model in RDF, in order to facilitate interoperability between repositories or registries publishing or exchanging metadata about digital objects. Implementers are therefore encouraged to use the RDF constructs listed below and supplement them by creating subclasses and subproperties of their own where necessary.

Please also note that, because the RDF expression recommended in this document directly reuses some external vocabularies, this document should be read before using the PREMIS Ontology 3.0.

Implementation details like OWL version, OWL sublanguage, preferred prefix and namespace URI will be determined when the final version of the ontology gets published.

Differences Between v. 2 and v. 3 Ontology

This is a revision of an RDF ontology based on the PREMIS Data Dictionary version 2.2 available at: http://www.loc.gov/premis/rdf/v1. It is a substantial remodeling based on incorporating emerging Linked Data best practices, such as reusing and connecting to classes and properties from other related ontologies. The previous version of the ontology reflected an encoding of the PREMIS Data Dictionary in RDF, which was fairly faithful to the semantic units of the Data Dictionary, enhanced by value vocabularies, available at http://id.loc.gov/preservationdescriptions. This revision asserts relationships between PREMIS classes and properties and other vocabularies, and in some cases reuses external classes and properties. Generalized properties are used throughout the ontology, such as for identifier, note, and version.

The basic principles followed for the PREMIS OWL ontology 3.0 were:

- Make the ontology as simple as possible.
- Reuse existing elements from well-known ontologies where possible and where semantics agree.
- Make relationships between PREMIS elements and those from well-known ontologies (with skos:closeMatch, skos:exactMatch). These can be hierarchical relationships (e.g. class/subclass, property/subproperty) or equivalencies (skos:exactMatch, skos:closeMatch).
- Use URIs to identify things rather than strings. If there is an enumerable set, then create URIs for the items of the set. These are done in http://id.loc.gov/preservationdescriptions
- Follow RDF constructions, rather than XML constructions. Use data dictionary names where appropriate.
- Types / Categories for an object are classes, using rdf:type, and creating subClasses if necessary.
- Refinement of properties should be done by creating subproperties, if necessary,
- Extension is handled via RDF, by simply adding additional properties to the objects,
- Cardinalities are not useful to validate RDF data, like in the XML Schema implementations, but for the user to know which data structures he/she could be awaiting. The semantic units mandatory obligation in the Data Dictionary will not systematically be reflected as classes and properties cardinality.
- The names of the classes and predicates should follow best practice naming conventions. Classes should generally be initial upper case noun phrases (ClassOfThing), predicates should be initial lowercase verb phrases (hasSomeRelationship).
- RDF functionality that isn't very widely supported or has the potential to conflict with local implementation, such as named graphs and reification, should be avoided.

Preservation Vocabularies Integration

Preservation vocabularies listed at http://id.loc.gov/vocabulary/preservation.html were originally created as controlled vocabularies associated with specific PREMIS semantic units.[1]

Example: the values of the semantic unit *messageDigestAlgorithm* semantic unit[2] should be a controlled list. A controlled vocabulary proposed at http://id.loc.gov/vocabulary/preservation/cryptographicHashFunctions.html specifies a standard form for the most commonly expected values of this semantic unit.

The new version of the ontology assigns a key role to these vocabularies as they are intended to provide refinements of generic PREMIS elements defined in the ontology. Therefore members of each preservation vocabulary, which are now instances of SKOS Concept and MADS-RDF Topic, will be in a near future provided with additional assertions, making them either subproperties, subclasses or instances of elements declared in the PREMIS ontology.

Example: members of the 'cryptographic hash functions' vocabulary mentioned above will be declared subclasses of `premis:Fixity`.

Should implementers feel the need for new elements in the preservation vocabularies, they can create their own and relate it to the corresponding ontology elements.

Example: an implementer who would need to specify that a Bitstream is located in an XML File, at a specific place indicated by its XML ID could create a `my:xmlIDref` class, and declare it subclass of `premis:StorageLocation`.

Note that if implementers consider their new term to be generic enough to serve the community, they may as well suggest that the PREMIS Editorial Committee add it.

External vocabularies

Ontology	**Prefix used in this document**	**Namespace**
The following vocabularies are reused from existing ontologies		
Dublin Core Elements	dce	http://purl.org/dc/elements/1.1/

[1] Note that preservation vocabularies can be used in any expression of the PREMIS Data Dictionary, be it XML, RDF or relational database, and even by non-PREMIS implementations.
[2] See PREMIS Data Dictionary version 3.0, p. 61, http://www.loc.gov/standards/premis/v3/premis-3-0-final.pdf#page=71.

Dublin Core Terms	dct	http://purl.org/dc/terms/
EBU Core	ebucore	https://www.ebu.ch/metadata/ontologies/ebucore#
Friend Of A Friend	foaf	http://xmlns.com/foaf/spec/
Open Digital Rights Language	odrl	https://www.w3.org/ns/odrl/2/
OWL 2 Web Ontology Language	owl	http://www.w3.org/2002/07/owl#
Object Reuse and Exchange	ore	http://www.openarchives.org/ore/terms
PROV Ontology	prov	http://www.w3.org/ns/prov#
RDF Schema	rdfs	http://www.w3.org/TR/rdf-schema/
The following vocabularies are id.loc.gov preservation vocabularies		
Actions Granted	acGranted	id.loc.gov/vocabulary/preservation/actionsGranted
Agent Type	agType	id.loc.gov/vocabulary/preservation/agentType
Content Location Type	contLocType	id.loc.gov/vocabulary/preservation/contentLocationType
Copyright Status	copStatus	id.loc.gov/vocabulary/preservation/copyrightStatus/
Cryptographic Hash Functions	crypHashFunc	id.loc.gov/vocabulary/preservation/cryptographicHashFunctions
Environment Characteristic	envChar	id.loc.gov/vocabulary/preservation/environmentCharacteristic
Environment Function Type	envFuncType	id.loc.gov/vocabulary/preservation/environmentFunctionType
Environment Registry Role	envRegRole	id.loc.gov/vocabulary/preservation/environmentRegistryRole
Event Outcome	evOutcome	id.loc.gov/vocabulary/preservation/eventOutcome (not yet established)*
Event Related Agent Role	evRelAgRole	id.loc.gov/vocabulary/preservation/eventRelatedAgentRole

Event Related Object Role	evRelObjRole	id.loc.gov/vocabulary/preservation/eventRelatedObjectRole
Event Type	evType	id.loc.gov/vocabulary/preservation/eventType
Format Registry Role	forRegRole	id.loc.gov/vocabulary/preservation/formatRegistryRole
Functionality	func	id.loc.gov/vocabulary/preservation/functionality (not yet established)*
Inhibitor Type	inhibType	id.loc.gov/vocabulary/preservation/inhibitorType
Linking Environment Role	linkEnvRole	id.loc.gov/vocabulary/preservation/linkingEnvironmentRole
Preservation Level Role	presLevRole	id.loc.gov/vocabulary/preservation/preservationLevelRole
Preservation Level Type	presLevType	id.loc.gov/vocabulary/preservation/preservationLevelType (not yet established)*
Relationship Subtype	relSubType	id.loc.gov/vocabulary/preservation/relationshipSubType
Relationship Type	relType	id.loc.gov/vocabulary/preservation/relationshipType
Rights Basis	rightsBasis	id.loc.gov/vocabulary/preservation/rightsBasis
Rights Related Agent Role	rightsRelAgRole	id.loc.gov/vocabulary/preservation/rightsRelatedAgentRole
Signature Encoding	sigEncoding	id.loc.gov/vocabulary/preservation/signatureEncoding
Signature Method	sigMethod	id.loc.gov/vocabulary/preservation/signatureMethod
Storage Medium	storMedium	id.loc.gov/vocabulary/preservation/storageMedium
The following vocabularies are used in examples		
Bibframe	bf	http://id.loc.gov/ontologies/bibframe#
Portland Common Data Model	pcdm	http://pcdm.org/models#
Wikidata Entity	wd	http://www.wikidata.org/entity/

The prefix "my:" is used to introduce an example of locally defined class or property.
*See document Changes for id.loc.gov preservation voc

Conventions

Typographic conventions

The name of PREMIS semantic units is indicated using *italics*. RDF vocabularies are indicated using the `Courier New` font.

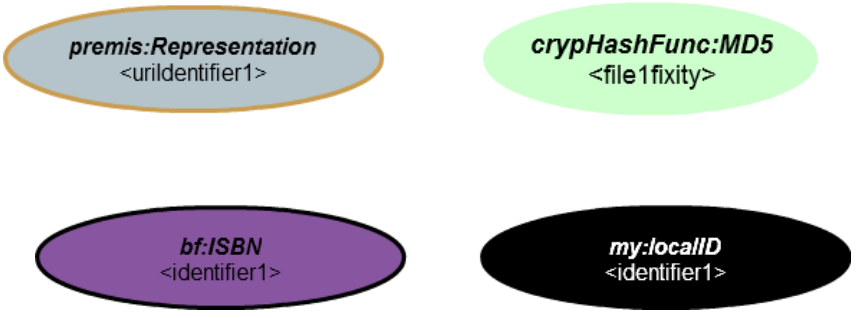
Example: "The *compositionLevel* semantic unit should be expressed through the `premis:hasCompositionLevel` property."

Diagram conventions

In the diagrams, resources are mentioned with a fictitious URI and the class they are instance of is indicated in ***bold and italics*** above the URI.

Instances of classes defined in the PREMIS ontology 3.0 are represented by gray ellipses, instances of locally defined classes are black ellipses, instances of classes or individuals defined at http://id.loc.gov/vocabulary/preservation are green ellipses, and individuals of classes defined in other ontologies are purple ellipses.

Examples:



Identifier

In RDF, defining an Identifier element is generally unnecessary, since the identifier of each Entity is the URI that is the subject of the triples. A URI being universally unique, its identifier type does not need to be specifically stated in a separate assertion.

Example:

Fig. 1: Identifier as URI

```
<uriIdentifier1> a premis:Representation .
```

Nevertheless, **if the identifier value is a literal and not a URI**, the simplest approach is to use `dct:identifier` (or subproperties of it to indicate the identifier type)

Example:

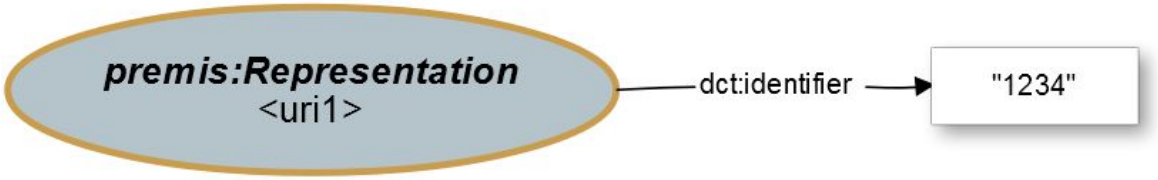


Fig. 2: Using dct:Identifier

```
<uri1> a premis:Representation ;
     dct:identifier "1234" .
```

Note: a `premis:Identifier` class and a `premis:hasIdentifier` property are maintained, but should be used **only**

- **if additional information is required**[3] (such as identifier status or administrative history);
- **or to point to instances of classes from existing vocabularies that would use classes to define identifier types**, e.g. http://id.loc.gov/ontologies/bibframe#c_ISBN.

Example 1: the identifier is a locally defined string, with additional assertions.

[3] Note that there is no equivalent semantic unit in the PREMIS Data DIctionary. Properties to be attached to a `premis:Identifier` class are therefore entirely up to implementers.

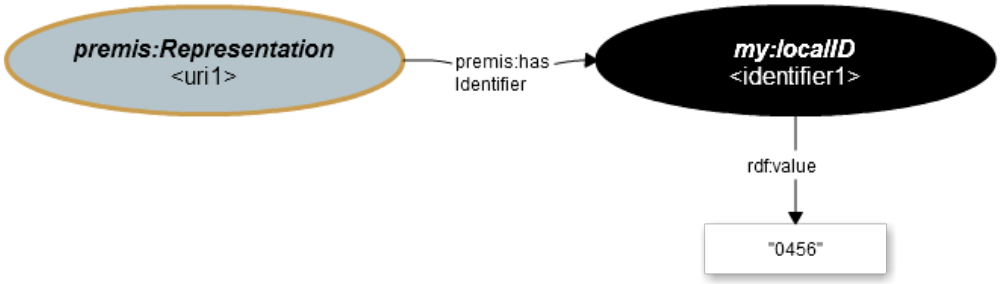


Fig. 3: Using locally defined identifiers with additional properties

```
<uri1> a premis:Representation ;
      premis:hasIdentifier <identifier1> .
<identifier1> a my:localID ;
      dct:created "2016-12-04" ;
      rdf:value "0456" .
```

Example 2: the identifier is an ISBN.

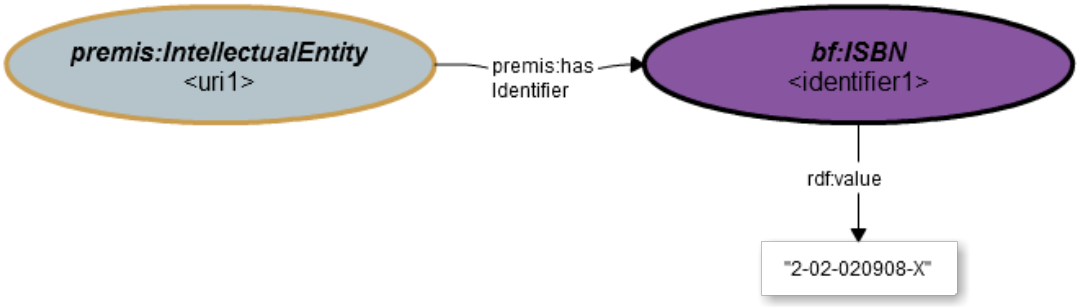


Fig. 4: Using externally defined identifiers with additional properties

```
@prefix bf: <http://id.loc.gov/ontologies/bibframe#> .
<uri1> a premis:IntellectualEntity ;
      premis:hasIdentifier <identifier1> .
<identifier1> a bf:ISBN ;
      rdf:value "2-02-020908-X".
```

Relationships Between PREMIS Entities

Relationships between the four different PREMIS Entities (Object, Event, Agent and Rights) are expressed by the following semantic units: *linkingEventIdentifier*, *linkingRightsStatementIdentifier*, *linkingAgentIdentifier*, *linkingObjectIdentifier* and

linkingEnvironmentIdentifier. Their mapping in RDF depends on the nature of the PREMIS entities they are binding.

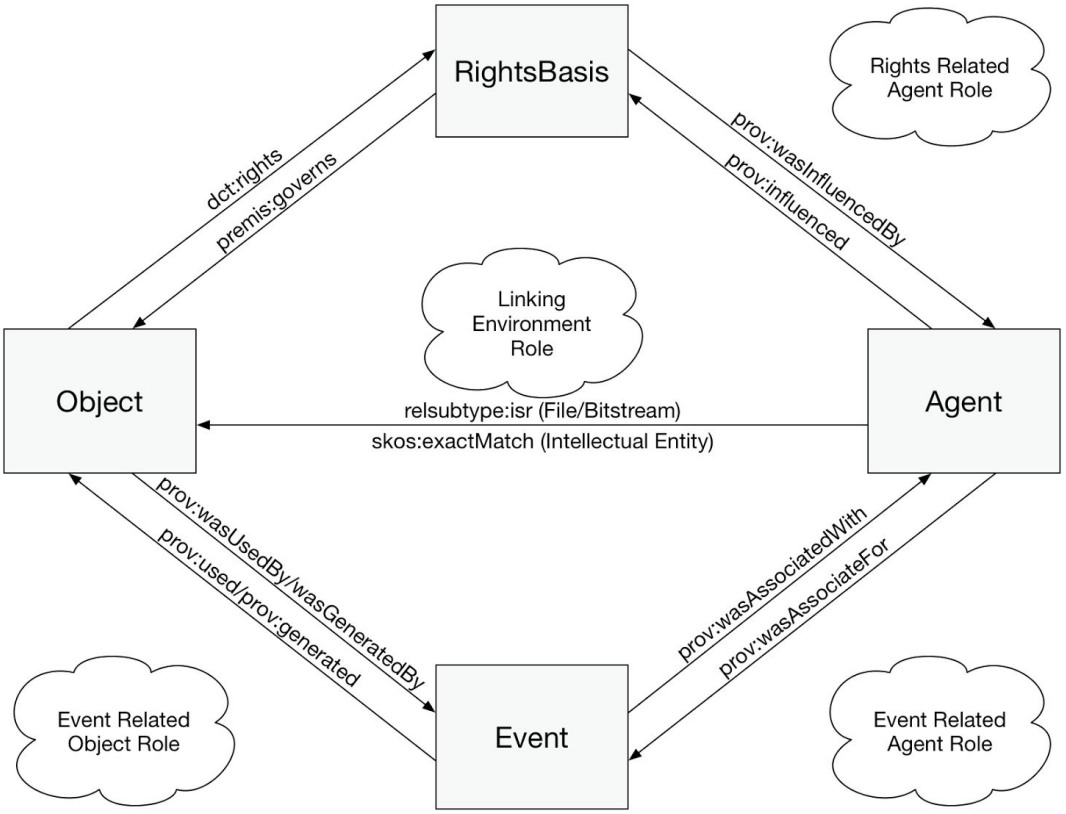


Fig. 5: Relationships between PREMIS entities

Object to Event

PROV-O properties should be used to point from an instance of `premis:Object` to an instance of `premis:Event`. The `prov:wasGeneratedBy` property should be used if the Object was created by the Event. The `prov:wasUsedBy` property should be used if the Object pre-existed the Event.

Note: the alternative construct used to associate an Event to an Object by means of the *relatedEventIdentifier*[4] semantic container should use the same construct in RDF, without regard to whether the Object is related to another Object through the Event or not.

Object to Rights

The `dct:rights` property should be used to point from an instance of `premis:Object` to an instance of `premis:RightsBasis`.

[4] See the PREMIS Data Dictionary v. 3.0 p. 15:
http://www.loc.gov/standards/premis/v3/premis-3-0-final.pdf#page=25

Event to Object

If the relationship from Event to Object must be stated, PROV-O properties `prov:generated` (inverse property of `prov:wasGeneratedBy`) and `prov:used` (inverse property of `prov:wasUsedBy`) should be used to point from an instance of `premis:Event` to an instance of `premis:Object`. If the Object role played in the Event is further specified by means of a *linkingObjectRole* semantic unit, locally defined subproperties of these PROV-O properties or those declared at id.loc.gov/vocabulary/preservation/eventRelatedObjectRole/ should be used.

Event to Agent

To express the relationship between Event and Agent the `prov:wasAssociatedWith` property should be used to point from an instance of `premis:Event` to an instance of `premis:Agent`. If the role played by the Agent in the Event is further specified by means of a *linkingAgentRole* semantic unit, subproperties of `prov:wasAssociatedWith` either declared at http://id.loc.gov/vocabulary/preservation/eventRelatedAgentRole or locally defined should be used.

Agent to Event

If the relationship from Agent to Event must be stated, the `prov:wasAssociateFor` PROV-O property (inverse property of `prov:wasAssociatedWith`) should be used.

Note that the relationship from an instance of `premis:Event` to an instance of `premis:Agent` should be preferred over this one, in particular because `prov:wasAssociateFor` is not an official PROV-O property.[5]

Rights to Agent

The relationship between Rights and Agent should be expressed through a `prov:wasInfluencedBy` property pointing from an instance of `premis:RightsBasis` to an instance of `premis:Agent`. If the role played in the definition of the rights basis by an Agent is further specified by means of a *linkingAgentRole* semantic unit, subproperties of `prov:wasInfluencedBy` either declared at http://id.loc.gov/vocabulary/preservation/rightsRelatedAgentRole or locally defined should be used.

Note: if users need to specify the Agent affected by the permission or prohibition expressed in the Rights Entity, a `prov:influenced` property may be used from an instance of `premis:Rule` to an instance of `premis:Agent`.

[5] See https://www.w3.org/TR/prov-o/#inverse-names for more information about PROV-O inverse properties.

Agent to Object

If the Object is a Representation, a File or a Bitstream, the `relSubType:isr` property (is Represented By) should be used; if the Object is an Intellectual Entity, one of the `skos:exactMatch / skos:closeMatch` properties should be used depending on whether the described Agent corresponds exactly or partially to the Environment Object described in an Environment registry. If the Environment Object role is further specified by means of a *linkingEnvironmentRole*, subproperties of `premis:hasRelationship` either declared at id.loc.gov/vocabulary/preservation/linkingEnvironmentRole/ or locally defined should be used.

Agent to Rights

If the relationship from Agent to Rights must be stated, a `prov:influenced` property should be used from an instance of `premis:Agent` to an instance of `premis:RightsBasis`.

Rights to Object

If the relationship from Rights to Object must be stated, the `premis:governs` property should be used to point from an instance of `premis:RightsBasis` to an instance of `premis:Object`.

Object

Object type (*objectCategory*)

Every PREMIS Object is an instance of one of the subclasses of the PREMIS Object abstract class. The class hierarchy is represented by the diagram below:

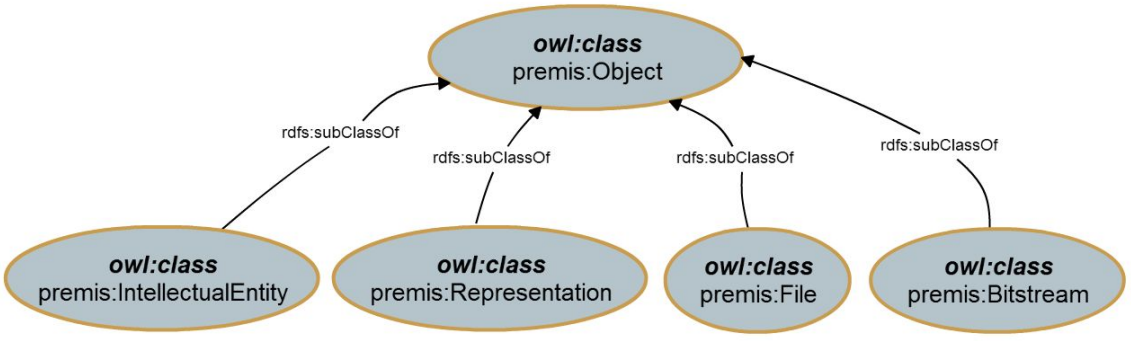


Fig. 6: The four PREMIS object types (objectCategory)

Hence, the *objectCategory* is specified using `rdf:type`.

Example:


```
<exampleTheses> a premis:IntellectualEntity .
```

Preservation Policy (*preservationLevel*)

PREMIS uses the *preservationLevel* semantic container to reference specific preservation policies applied to an Object or a set of Objects. Implementers should use the `premis:PreservationPolicy` class or one of its subclasses declared at http:id.loc.gov/preservation/preservationLevelType (not yet established), or create locally defined subclasses of these for every specific value of the *preservationLevelType* semantic unit; instances of such classes should then be created for every specific preservation policy and RDF constructs listed below should be used to describe them.

- The preservation policy applied to the Object should be related to it by means of a `premis:hasPolicy` property.
- As values of the *preservationLevelValue* semantic unit are meant to be specific to each implementer, it is recommended to create local instances of subclasses of `premis:PreservationPolicy` and attach to them a free-text description with a `premis:hasNote` property.
- To express the context in which the policy applies to the Object, e.g., to distinguish between the intended preservation level and the current achievable preservation level (*preservationLevelRole* semantic unit), implementers should use subproperties of `premis:hasPolicy`, either declared at http://id.loc.gov/vocabulary/preservation/preservationLevelRole or locally defined.
- The rationale for why the policy was assigned is expressed by the PREMIS semantic unit *preservationLevelRationale*. In RDF, this should be expressed through a `premis:hasRationale` property attached to the preservation policy, whose object would be a string literal.
- It is recommended that implementers use a PREMIS Event if they need to record information about the policy assignment. Hence, *preservationLevelDateAssigned* should be expressed through a `dct:date` property attached to an instance of a policy assignment Event (defined at http://id.loc.gov/vocabulary/preservation/eventType/poa).

Example:

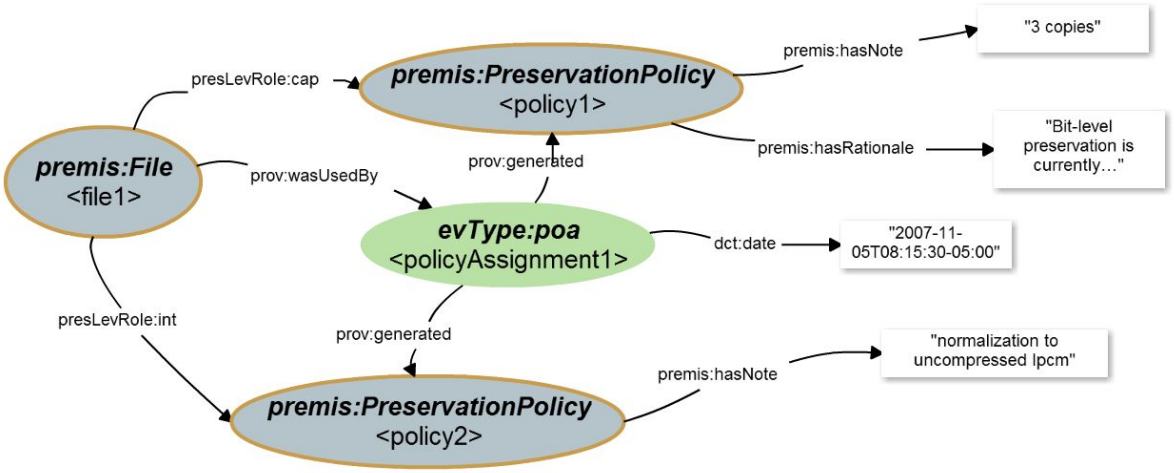


Fig. 7: An Object assigned with two policies

```
<file1> a premis:File;
      presLevRole:cap <policy1> ;
      presLevRole:int <policy2> ;
      prov:wasUsedBy <event1> .

<policy1> a presLevType:blp ;
      premis:hasNote "3 copies";
      premis:hasRationale "Bit-level preservation is currently…" .

<policy2> a presLevType:lop ;
      premis:hasNote "normalization to uncompressed lpcm" .

<event1> a evType:poa ;
      prov:generated <policy1> , <policy2> ;
      dct:date "2007-11-05T08:15:30-05:00".
```

Significant Properties

Significant properties are defined by the PREMIS Data Dictionary as "characteristics of a particular object subjectively determined to be important to maintain through preservation action". Thus, the `premis:SignificantProperties` class is declared a subclass of the `premis:PreservationPolicy` class. Instances of the `premis:SignificantProperties` class, attached to the Object thanks to a `premis:hasPolicy` property, are meant to point to one or several properties of the Object that should be considered significant, using the `rdfs:value` property. In the example below, <File1> and <File2> are formatted with bold and italics, and this property has been deemed significant. A Policy assignment Event records the date and time when a preservation policy, aiming at preserving the Objects formatting, was applied to these Objects. Note that <File1> has another property (frame rate) that was not deemed significant.

Significant properties can be shared and linked to from multiple objects when they are a general policy, or unique to a single object:

Example:

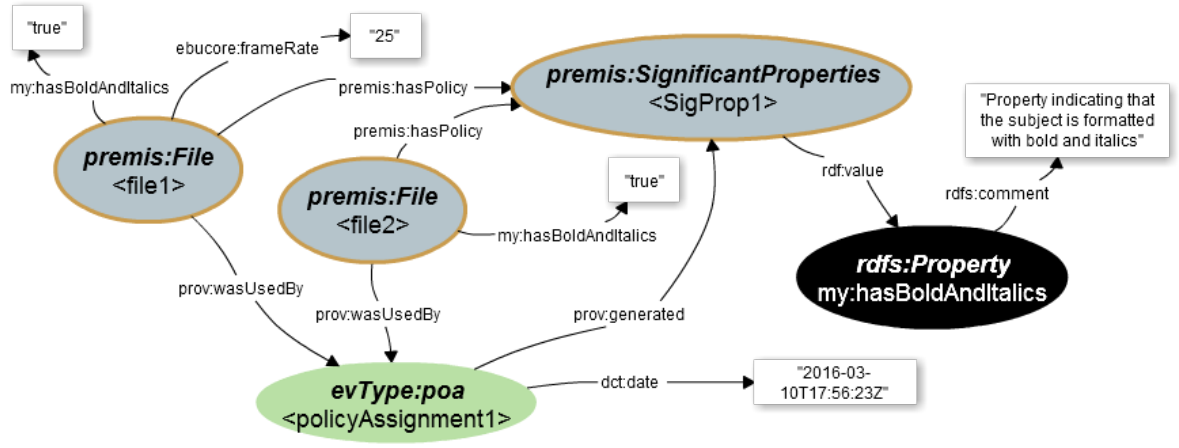


Fig. 8: Significant properties

```
<File1> a premis:File;
  premis:hasPolicy <sigProp1> ;
  prov:wasUsedBy <event1> ;
  ebucore:frameRate "25" ;
  my:hasBoldAndItalics "true" .

<File2> a premis:File ;
  premis:hasPolicy <SigProp1> ;
  prov:wasUsedBy <event1> ;
  my:hasBoldAndItalics "true" .

<SigProp1> a premis:SignificantProperties ;
  rdf:value <hasBoldAndItalics> .

<Event1> a evType:poa ;
  prov:generated <SigProp1> ;
  dct:date "2016-03-10T17:56:23Z" .

my:hasBoldAndItalics a rdfs:Property;
  rdfs:comment "Property indicating that the subject is formatted
with bold and italics" .
```

Compound Objects (*compositionLevel*)

When an Object is compressed, encrypted or just packaged into a container file, each "layer" is described as a different `premis:Object` related to the others by means of properties either declared at http://id.loc.gov/vocabulary/preservation/relationshipSubType or locally defined subproperties of `premis:hasRelationship`. For a comprehensive description of the way PREMIS defines compound Objects, see the "Object characteristics and composition level: the "onion" model" section page 256 of the Data Dictionary.[6]

- The *compositionLevel* semantic unit is expressed by a `premis:hasCompositionLevel` property.
- Indicate the relationship between these different Objects, such as
  - `relSubType:cot` (compressed to) / `relSubType:cof` (compressed from) for compressed Objects; `relSubType:ent` (encrypted to)/`relSubType:enf` (encrypted from)
  - `relSubType:hsp` (has part) / `relSubType:isp` (is part of) for packaged Objects.

Example:

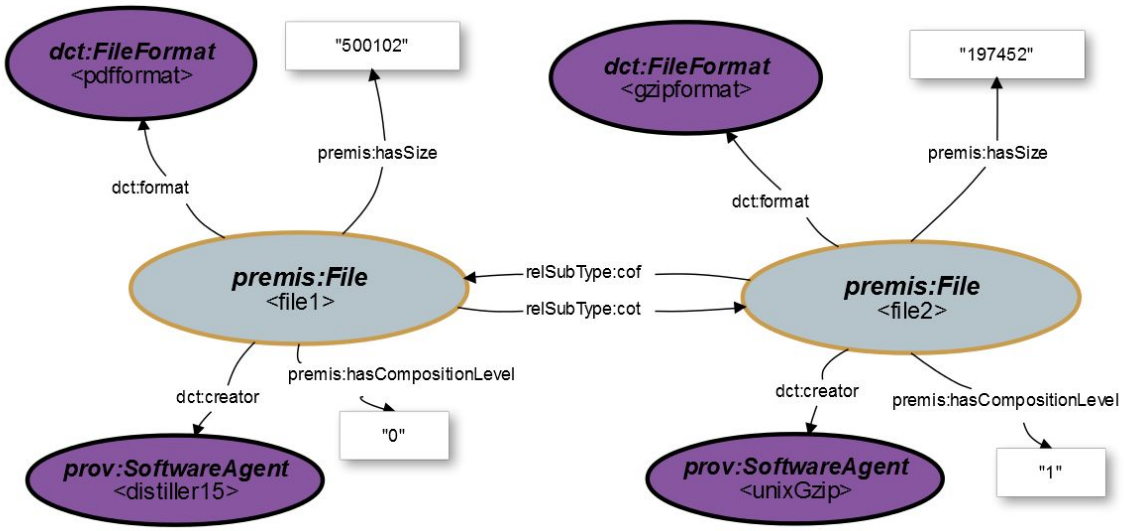


Fig. 9: Relationship between compressed and uncompressed versions of a File

```
<file1> a premis:File ;
  premis:hasCompositionLevel "0" ;
  dct:format <pdfformat> ;
  premis:hasSize "500102" ;
  dct:creator <Acrobat> ;
  relSubType:cot <file2> .
```

[6] Cf. http://www.loc.gov/standards/premis/v3/premis-3-0-final.pdf#page=266.

```
<file2> a premis:File ;
  premis:hasCompositionLevel "1" ;
  dct:format <gzipformat> ;
  premis:hasSize "197452" ;
  dct:creator <unixGzip> ;
  relSubType:cof <file1> .
```

Fixity

- Fixity information is attached to the Object with a `premis:hasFixity` property having a range of `premis:Fixity`.
- The algorithm used to produce the message digest (*messageDigestAlgorithm* semantic unit) should be expressed using a subclass of `premis:Fixity` either declared at http://id.loc.gov/vocabulary/preservation/cryptographicHashFunctions or locally defined.
- The *messageDigest* itself is attached to the instance of `premis:Fixity` using the `rdf:value` property.
- The agent that is the creator of the message digest (*messageDigestOriginator*) is expressed by an instance of `prov:SoftwareAgent` attached by a `dct:creator` property to the instance of `premis:Fixity`.
  Note: the *messageDigestOriginator* may also be declared as an implementer Agent (cf. http://id.loc.gov/vocabulary/preservation/linkingAgentRoleEvent/imp) involved in a message digest calculation Event (cf. http://id.loc.gov/vocabulary/preservation/eventType/mes).
  Note: implementers who would not want to specify URIs for the message digest originator may use `dce:creator` and mention it as a literal.

Example:

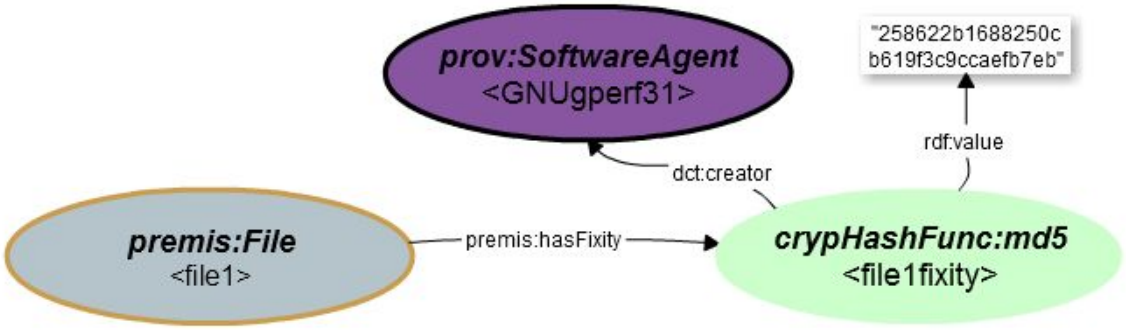


Fig. 10: Fixity information

```
<file1> a premis:File ;
premis:hasFixity <file1fixity> ;
<file1fixity> a crypHashFunc:md5 ;
  rdf:value "258622b1688250cb619f3c9ccaefb7eb" ;
  dct:creator <GNUgperf31> .
```

Size

To express the size of a File or a Bitstream, use `premis:hasSize`.

Example:

```
<file1> a premis:File ;
  premis:hasSize "478347923" .
```

Format

- A Bitstream or File format is expressed through a `dct:format` property pointing to an instance of the `dct:FileFormat` class. This construct may be repeated if the Object complies with multiple format definitions.
    - Note: if specifying the format by a MIME type is deemed sufficient by implementers, the `ebucore:hasMimeType` property may be used in addition to or in replacement of the mentioned construct.
- The *formatName* semantic unit should be expressed through a `rdfs:label` property attached to the instance of `dct:FileFormat`.
- The *formatVersion* semantic unit should be expressed through a `premis:hasVersion` property attached to the instance of `dct:FileFormat`
- The *formatNote* semantic unit should be expressed through a `premis:hasNote` property attached to the instance of `dct:FileFormat`.
- *formatRegistry* semantic units identify the same format in another registry, or a more generic definition of it. Depending on the degree of similarity between the two resources, either `skos:exactMatch` or `skos:closeMatch` should be used.
    - *formatRegistryRole*: if the registry role must be explicitly stated, properties defined at id.loc.gov/vocabulary/preservation/formatRegistryRole should be used, or subproperties of `skos:closeMatch` should be locally defined.
    - *formatRegistryName*: if the registry name must be explicitly stated, subproperties of `skos:exactMatch` or `skos:closeMatch` should be locally defined.

Example:[7]

[7] The PRONOM resource stands for PDF/A-1b, so `skos:exactMatch` is used, whereas the LoC registry describes the more generic PDF/A-1, so `skos:closeMatch` is used.

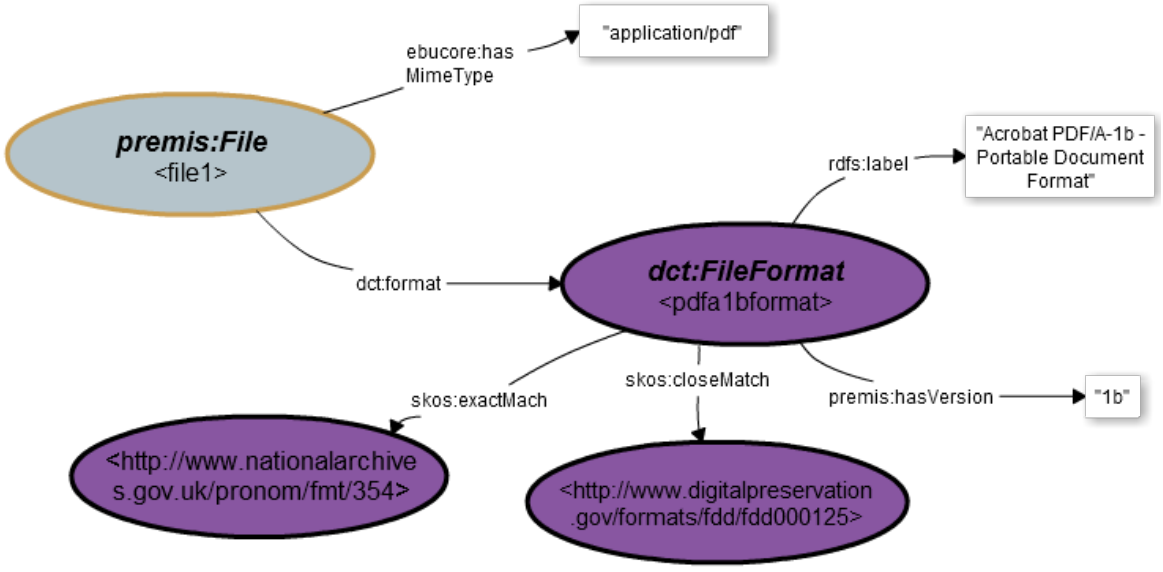


Fig. 11 File format information

```
<file1> a premis:File ;
  ebucore:hasMimeType "application/pdf" .
  dct:format <pdfa1bformat> .

<pdfa1bformat> a dct:FileFormat ;
  rdfs:label "Acrobat PDF/A-1b - Portable Document Format" ;
  premis:hasVersion "1b" ;
  premis:hasNote "Some note about PDF/A-1b" ;
  skos:exactMatch
<http://www.nationalarchives.gov.uk/pronom/fmt/354> ;
  skos:closeMatch
    <http://www.digitalpreservation.gov/formats/fdd/fdd000318> .
```

Creating Application

Though describing an Object's creating application is preferably done by declaring an Agent related to a creation Event, the following constructs may alternatively be used by implementers, namely if the Event and Agent Entities are not supported.

- Creating application is represented by an instance of `prov:SoftwareAgent` related to the created Object by a `dct:creator` property.
- The *dateCreatedByApplication* semantic unit should be expressed through a `prov:generatedAtTime` property attached to the Object.
- The *creatingApplicationName* semantic unit should be expressed through an `rdfs:label` property attached to the instance of `prov:SoftwareAgent`. Note: implementers who would not want to specify URIs for the creating application may use `dce:creator` and mention the creating application name as a literal.

- The *creatingApplicationVersion* semantic unit should be expressed through a `premis:hasVersion` property attached to the instance of `prov:SoftwareAgent`.

Example:

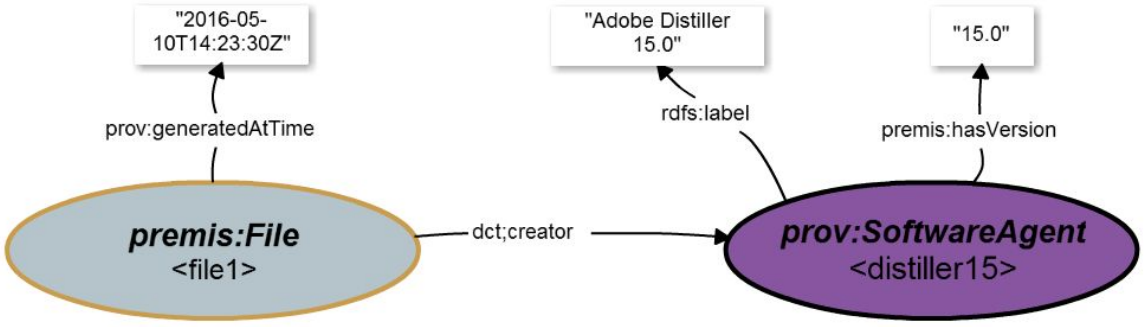


Fig. 12: Using dct:creator to link a File to its creating application

```
<file1> a premis:File ;
  dct:creator <distiller15> ;
  prov:generatedAtTime "2016-05-10T14:23:30Z" .

<distiller15> a prov:SoftwareAgent ;
  rdfs:label "Adobe Distiller 15.0" ;
  premis:hasVersion "15.0" .
```

Inhibitors

- Expressing features of the Object that could compromise preservation actions is done by declaring instances of subclasses of the abstract `premis:Inhibitor` class related to the Object by means of a `premis:inhibitedBy` property.
- Specifying the *inhibitorType* semantic unit is done by using subclasses of `premis:Inhibitor`, either declared at http://id.loc.gov/vocabulary/preservation/inhibitorType or locally defined.
- The *inhibitorTarget* semantic unit should be expressed by means of a `premis:inhibits` property related to the instance of `premis:Inhibitor` and pointing to an instance of `premis:Functionality`, either declared at http://id.loc.gov/vocabulary/preservation/functionality (not yet established) or locally defined.
- The *inhibitorKey* semantic unit is expressed by means of a `premis:hasKey` property related to the instance of `premis:Inhibitor` and pointing to a literal value of type `xsd:String`.

Note: Storing passwords or decryption keys is sometimes necessary, but raises security issues. Consider storing keys on offline media or printing them out, and referencing their location here.

Example:

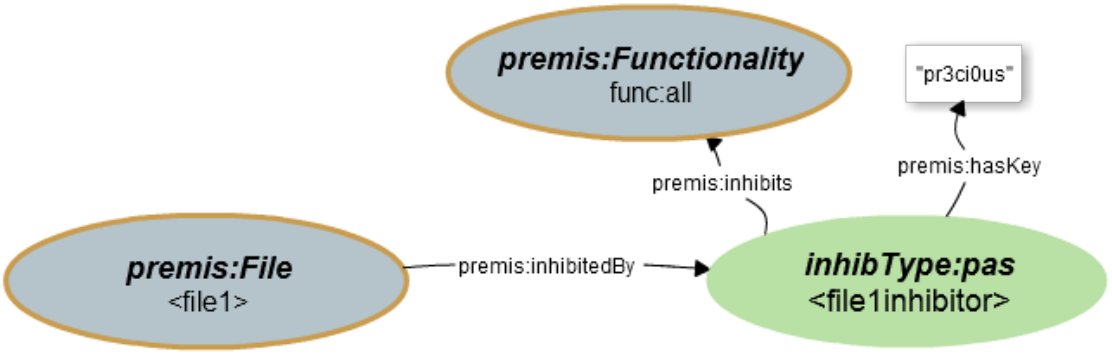


Fig. 13: A password-protected File

```
<file1> a premis:File ;
  premis:inhibitedBy <file1inhibitor> .

<file1inhibitor> a inhibType:pas ;
  premis:inhibits func:all ;
  premis:hasKey "pr3ci0us" .
```

Original Name

Expressing the name of the Object as submitted to or harvested by the repository is expressed by means of a `premis:hasOriginalName` related to the Object and pointing to a literal value of type `xsd:String`.

Example:

```
<uri1> a premis:File ;
  premis:hasOriginalName "thesis.final.20160509.docx" .
```

Storage

- Storage information about an Object (be it a Representation, either digital or physical, a File or a Bitstream) should be expressed by a `premis:storedAt` property pointing from the Object to an instance of `premis:StorageLocation`.
- The means to reference the location of the content (the *contentLocationType* semantic unit) should be expressed by subclasses of `premis:StorageLocation`,

either declared at http://id.loc.gov/vocabulary/preservation/contentLocationType or defined locally.

- The reference to the content location itself (the *contentLocationValue* semantic unit) should be expressed by the `rdf:value` property attached to the instance of `premis:StorageLocation` and pointing to a string (a filepath, a shelfmark, etc.).
- The storage medium on which the Object is preserved (the *storageMedium* semantic unit) is expressed by the `premis:hasMedium` property, pointing from the Object to an instance of `premis:StorageLocation` to an instance of `premis:StorageMedium`, either declared at http://id.loc.gov/vocabulary/preservation/storageMedium or defined locally.

Example:

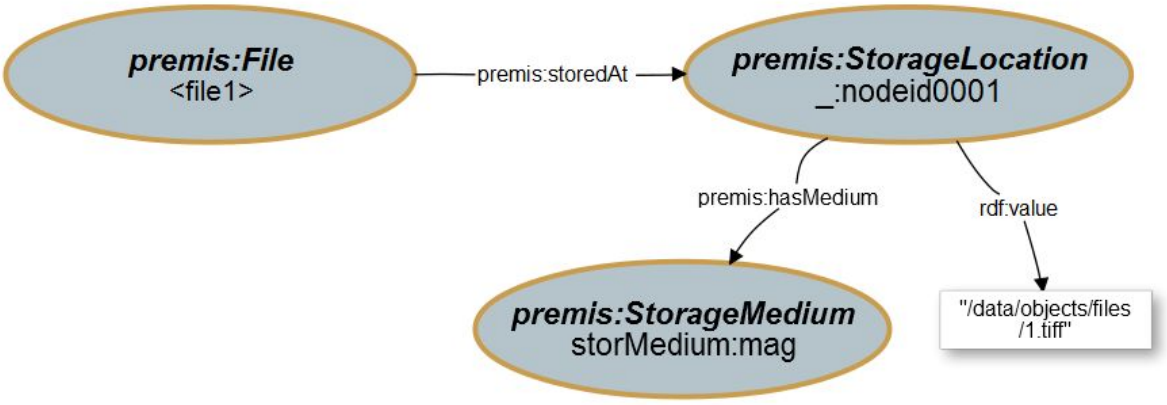


Fig. 14: Storage location information

```
<file1> a premis:File ;
      premis:storedAt [
            a premis:StorageLocation ;
            rdf:value "/data/objects/files/1.tiff" ;
            premis:hasMedium storMedium:mag . ] .
```

Signature

- Signature information, which is used to authenticate the signer of an object and/or the information contained in the object, should be expressed by a `premis:hasSignature` property attached to an instance of `premis:File` or `premis:Bitstream` pointing to an instance of `premis:Signature`.
- The authority responsible for generating the signature, mentioned by a *signer* semantic unit, should be expressed by an Agent involved in a digital signature generation Event by means of an `evRelAgRole:imp` property.
  Note: Alternatively, if implementers do not use the Agent and Event Entities, it may be expressed by a `dce:creator` property attached to the instance of `premis:Signature` and pointing to a literal.

- The encryption and hash algorithm used to generate the signature, mentioned in a *signatureMethod* semantic unit, should be expressed by subclasses of `premis:Signature` either declared at http://id.loc.gov/vocabulary/preservation/signatureMethod or locally defined.
- The value generated from the application of a private key to a message digest, mentioned in a *signatureValue* semantic unit, should be expressed by an `rdf:value` property attached to the instance of `premis:Signature` and pointing to a literal.
- The public key used to verify that the signature value is valid (*keyInformation* semantic unit) should be expressed by a `premis:hasKey` property attached to the instance of `premis:Signature` and pointing to a literal.
- Information conveyed by the *signatureEncoding* semantic unit, i.e. encoding used for the values of `rdf:value` (*signatureValue*) and `premis:hasKey` (*keyInformation*), should be expressed by a `premis:hasEncoding` property attached to the instance of `premis:Signature` and pointing to an instance of `premis:SignatureEncoding`, either declared at http://id.loc.gov/vocabulary/preservation/signatureEncoding or locally defined.
- Operations to be performed in order to validate the digital signature (*signatureValidationRules* semantic unit) should be expressed by a `premis:hasValidationRules` property attached to an instance of `premis:Signature`.
  Note: this property is intended to point to a free-text description of the validation rules. If implementers need to point to a resource documenting the validation rules, they may use `premis:hasDocumentation` instead.
- Additional information about the generation of the signature (*signatureProperties* semantic unit) should be expressed either by properties of a digital signature generation Event (date, Agents involved, etc.)
  Note: implementers who would not implement the Event entity may use a `premis:hasNote` property attached to the instance of `premis:Signature`.

Example:

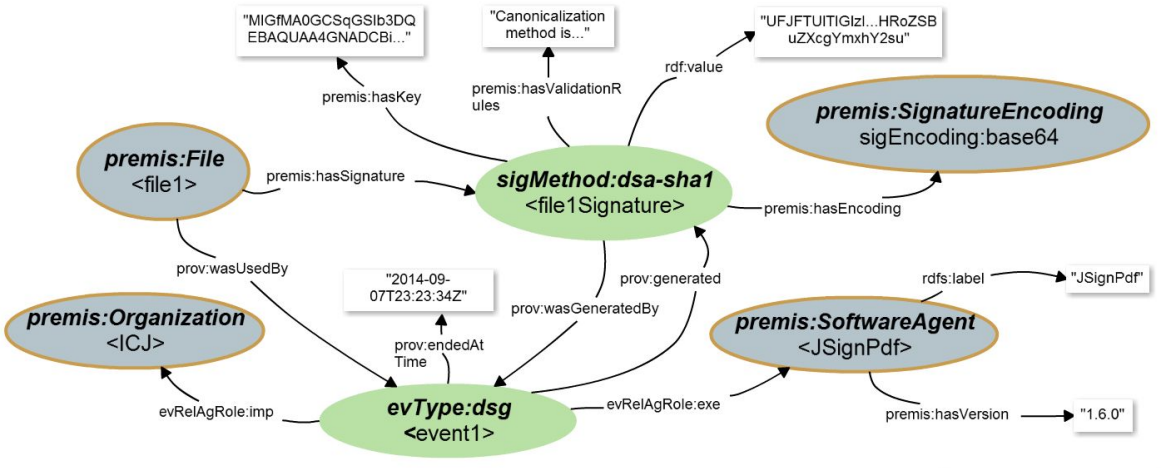


Fig. 15: Digital signature information

```
<file1> a premis:File ;
  premis:hasSignature <file1Signature> ;
  prov:wasUsedBy <event1> .

<file1Signature> a sigMethod:dsa-sha1 ;
  rdf:value "UFJFTUlTIGlzI...HRoZSBuZXcgYmxhY2su" ;
  premis:hasEncoding sigEncoding:base64 ;
  premis:hasKey "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBi..." ;
  premis:hasValidationRules "Canonicalization method is..." ;
  prov:wasGeneratedBy <event1>.

<event1> a evType:dsg ;
  prov:endedAtTime "2014-09-07T23:23:34Z" ;
  evRelAgRole:imp <ICJ> ;
  evRelAgRole:exe <JSignPdf> :
  prov:generated <file1Signature> .

<JSignPdf> a premis:SoftwareAgent ;
  rdfs:label "JSignPdf" ;
  premis:hasVersion "1.6.0" .

<ICJ> a premis:Organization .
```

Environment

An Environment is any kind of technology (software, hardware or a combination of both) supporting an Object in some way (rendering, executing it, etc.). Given that an Object can be stored by a repository as a historical asset but considered an Environment by another repository, being an Environment is not an intrinsic characteristic of the Object. Therefore, an Object is not explicitly declared as Environment, but is considered as so if an Object declares a dependency relationship to it. Any category of Object can be considered an Environment; however, the following constructs are applicable only to Environments described as Intellectual Entities.

- The *environmentFunction* semantic container and its semantic units are expressed through subclasses of `premis:IntellectualEntity`, defined at id.loc.gov/vocabulary/preservation/environmentFunctionType.
- *environmentName* is expressed by the `rdfs:label` property.
- *environmentVersion* is expressed by a `premis:hasVersion` property.
- *environmentOrigin* mentions the producer of the Environment. It is expressed by a `dct:creator` property. The object of such property is expected to be an instance of `foaf:Person` or `foaf:Organization`.
- *environmentDesignationNote* is considered a free-text note about the Environment and thus expressed by a `premis:hasNote` property.

- *environmentRegistry* semantic units identify the same Environment in another registry. Depending on the degree of similarity between the two resources, either `skos:exactMatch` or `skos:closeMatch` should be used.
    - *environmentRegistryRole*: if the registry role must be explicitly stated, properties defined at id.loc.gov/vocabulary/preservation/environmentRegistryRole should be used, or subproperties of `skos:closeMatch` should be locally defined.
    - *environmentRegistryName*: if the registry name must be explicitly stated, subproperties of `skos:exactMatch` or `skos:closeMatch` should be locally defined.

1. Example: WordPerfect 5.1 for DOS

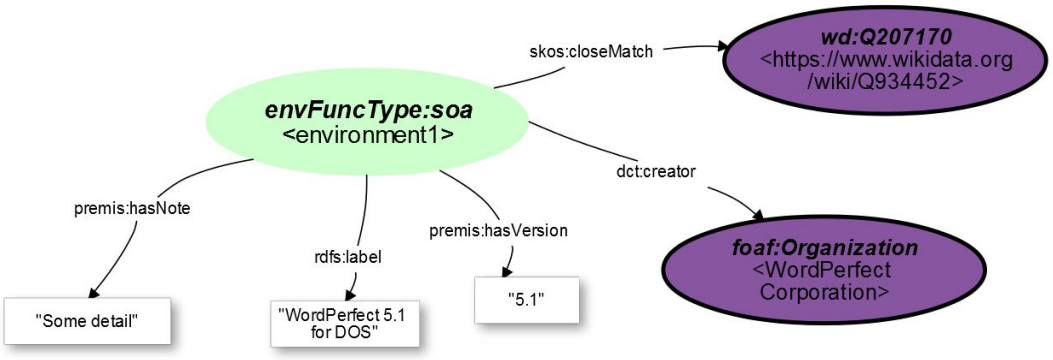


Fig. 16 A software application described as an Environment

```
<environment1> a envFuncType:soa ;
      rdfs:label "WordPerfect 5.1 for DOS" ;
      premis:hasVersion "5.1" ;
      premis:hasNote "Some detail" ;
      dct:creator <WordPerfect Corporation> ;
      skos:closeMatch <https://www.wikidata.org/wiki/Q934452>
.

<WordPerfect Corporation> a foaf:Organization .
```

Relationships between Objects

Relationships between PREMIS Objects should be expressed by subproperties of the `premis:hasRelationship` property either declared at http://id.loc.gov/vocabulary/preservation/relationshipSubType or locally defined. In most cases, one of these properties is sufficient to express the nature of the relationship between two PREMIS Objects.

Example: a set of Files represents a digitized book.

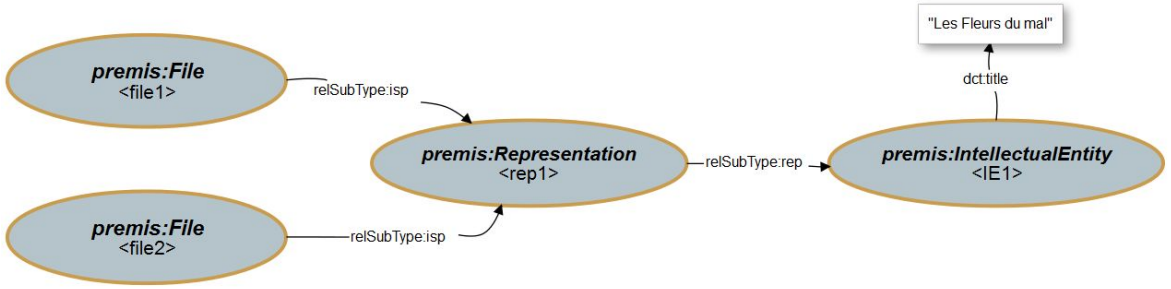


Fig. 17: Relationships between Intellectual Entities, Representations and Files

```
<file1> a premis:File ;
      relSubType:isp <rep1> .
<file2> a premis:File ;
      relSubType:isp <rep1> .

<rep1> a premis:Representation ;
      relSubType:rep <IE1> .

<IE1> a premis:IntellectualEntity ;
      dct:title "Les Fleurs du mal" .
```

Example: a repository has a file from which it creates a derivative.

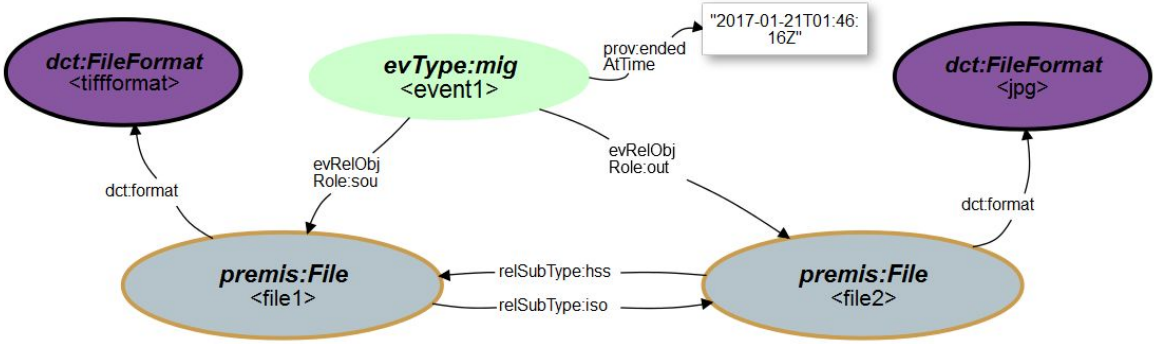


Fig. 18: Relationship between a File and its derivative

```
<file1> a premis:File ;
  dct:format <tiffformat> ;
  relSubType:iso <file2> .

<event1> a evType:mig ;
```

```
  prov:endedAtTime "2017-11-03T11:46:16+02:00" ;
  evRelObjRole:sou <file1> ;
  evRelObjRole:out <file2> .

<file2> a premis:File ;
  dct:format <jpgformat> ;
  relSubType:hss <file1> .
```

For complex relationships involving Environments (particularly dependency relationships), a more complex structure is needed to specify the Environment purpose (which action the Environment is meant to perform on the Object) and characteristic (the extent to which the Environment support its purpose). An instance of `premis:Dependency` should in this case be created to bear `premis:hasPurpose` and `premis:hasCharacteristic` properties.

Example: a repository preserves WordPerfect for DOS files and uses PREMIS to model an environment stack for rendering the files.

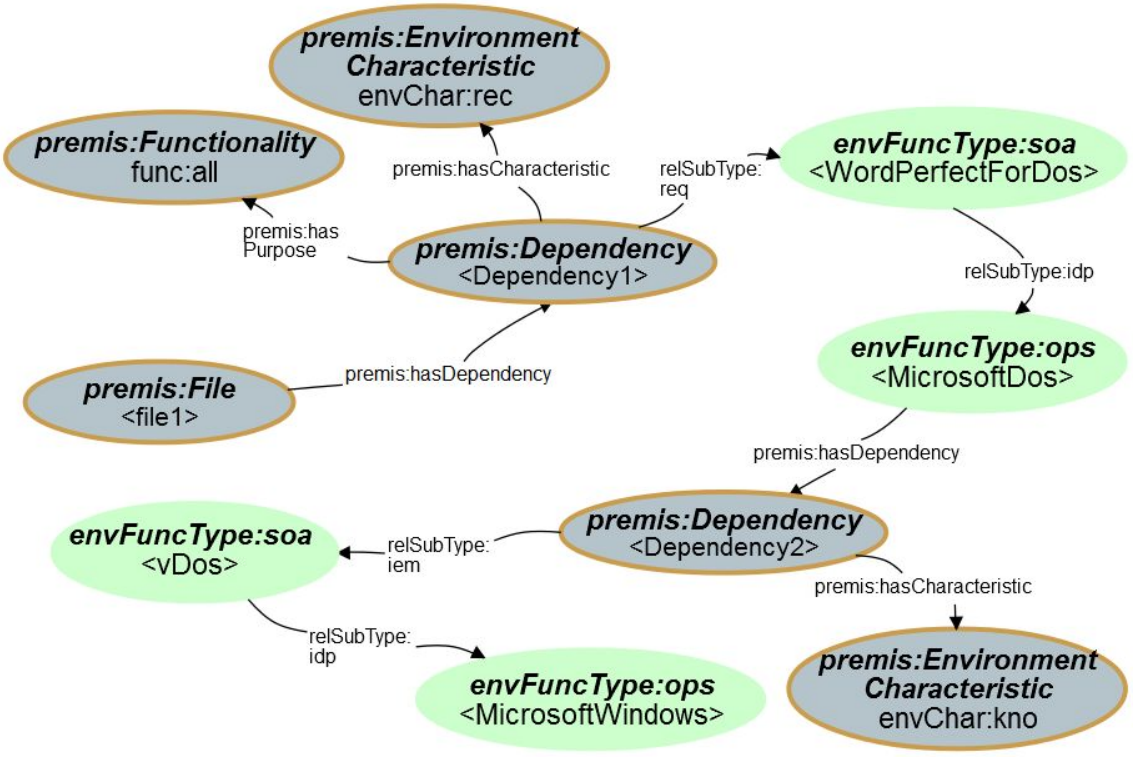


Fig. 19: A rendering environment for a File

```
<file1> a premis:File ;
    premis:hasDependency <Dependency1> .

<Dependency1> a premis:Dependency ;
    relSubType:req <WordPerfectForDos> ;
    premis:hasPurpose func:all ;
    premis:hasCharacteristic envChar:rec .
```

```
<WordPerfectForDos> a envFuncType:soa ;
    relSubType:idp <MicrosoftDos> .

<MicrosoftDos> a envFuncType:ops ;
    relSubType:req <dependency2> .

<dependency2> a premis:Dependency ;
    relSubType:iem <vDos> ;
    premis:hasCharacteristic envChar:kno .

<vDos> a envFuncType:soa ;
    relSubType:idp <MicrosoftWindows> .

<MicrosoftWindows> a envFuncType:ops .
```

Objects sequencing (*relatedObjectSequence*)

Use `edm:isNextInSequence`. (For aggregation-specific ordering, use `ore:Proxy`).

Event

Event is one of the four PREMIS Entities. The corresponding `premis:Event` class is a subclass of `prov:Activity`.

- The *eventType* semantic unit should be expressed by subclasses of `premis:Event`, either declared at http://id.loc.gov/vocabulary/preservation/eventType or locally defined.
- The *eventDateTime* semantic unit should be expressed by `prov:startedAtTime` or `prov:endedAtTime` properties or both, if a date and time range has to be specified.
  Note: in the case where legacy data must be transformed into RDF and if the *eventDateTime* value cannot be determined as the beginning or the end of an Event, a `dct:date` property may be used to express *eventDateTime* instead of PROV-O properties.
- The *eventDetail* semantic unit is used to mention a free-text note about the Event. The property `premis:hasNote` should be used.
- The *eventOutcome* semantic unit is used to mention the Event result in a coded way. The property `premis:hasOutcome` should be attached to the Event and point to an inidividual of a `premis:OutcomeStatus` class, either declared at http://id.loc.gov/vocabulary/preservation/eventOutcome (not yet established) or locally defined.
  Note: if the Event generated a non-PREMIS Object resource (a `premis:PreservationPolicy`, `premis:Signature`, `premis:Fixity`, etc.), implementers may use `prov:generated` to specify the relationship between the Event and this resource.[8]
- The *eventOutcomeDetailNote* semantic unit provides additional free-text information about the Event outcome. It should be expressed through a `premis:hasOutcomeNote` property attached to the Event and pointing to a string value.

Example 1: a compression Event generating a compressed File with the 7zip tool.

[8] See examples of this construct in sections on Policy, Signature and Significant properties above.

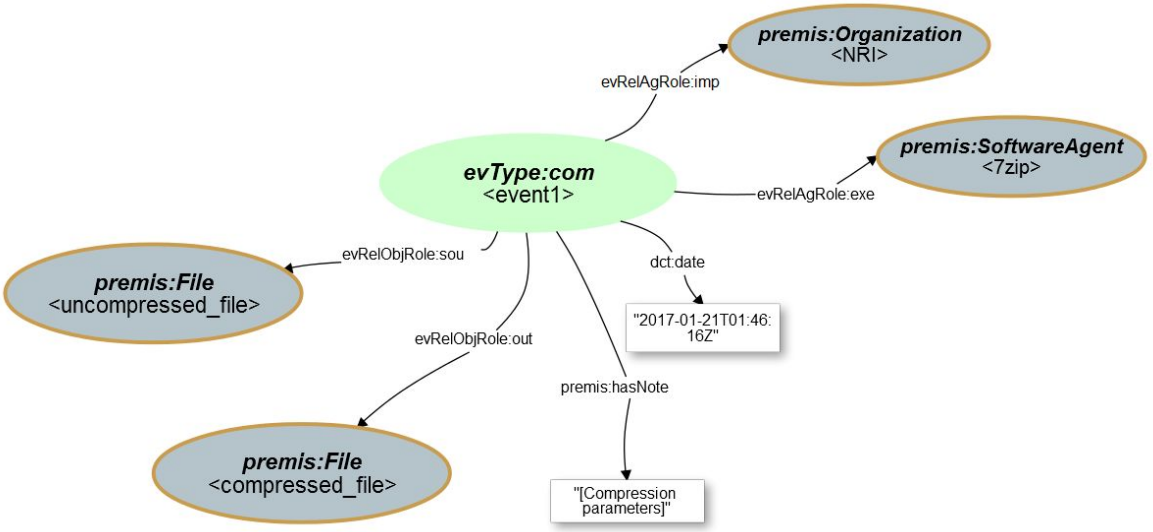


Fig. 20: A compression Event

```
<event1> rdf:type evType:com ;
      dct:date "2017-01-21T01:46:16Z" ;
      premis:hasNote "[Compression parameters]" ;
      evRelObjRole:sou <uncompressed_file> ;
      evRelObjRole:out <compressed_file> ;
      evRelAgRole:imp <NRI> ;
      evRelAgRole:exe <7zip> .

<NRI> a premis:Organization .

<7zip> a premis:SoftwareAgent .
```

Example 2: an Event tracking metadata extraction from a file by Jhove

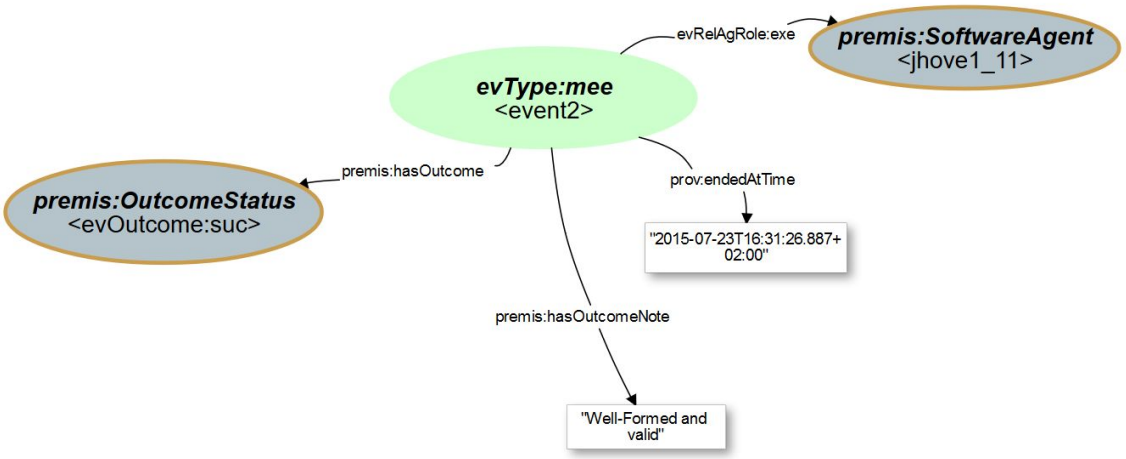


Fig. 21: A metadata extraction Event

```
<event2> rdf:type evType:mee ;
      prov:endedAtTime "2015-07-23T16:31:26.887+02:00" ;
      evRelAgRole:exe <jhove1_11> ;
      premis:hasOutcome evOutcome:com ;
      premis:hasOutcomeNote "Well-Formed and valid" .

<jhove1_11> a premis:SoftwareAgent .
```

Agent

Agent is one of the the four PREMIS Entities. Its corresponding class, `premis:Agent`, is an abstract class, subclass of both `foaf:Agent` and `prov:Agent`.

- The *agentType* semantic unit should be expressed by subclasses of `premis:Agent`: `premis:SoftwareAgent`, `premis:HardwareAgent`, `premis:Person` or `premis:Organization`.
- The *agentName* semantic unit should be expressed by a `foaf:name` property for instances of `premis:Organization` and `premis:Person`, or `rdfs:label` for instances of `premis:SoftwareAgent` and `premis:HardwareAgent`.
- The *agentVersion* semantic unit should be expressed by a `premis:hasVersion` property.
- The *agentNote* semantic unit should be expressed by a `premis:hasNote` property.
- Note that the premis namespace is used for the agentType only when the Agent is connected to `premis:Event`, `premis:RightsBasis` or `premis:Rule`. If the Agent is attached to an Object as a creating application or to `premis:Fixity` using `dct:creator`, `prov:SoftwareAgent` Is used instead.

Example:

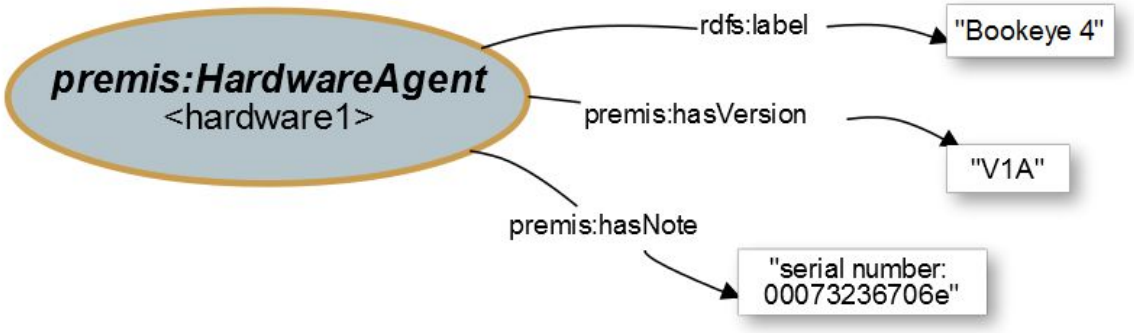


Fig. 22: A hardware Agent

```
<hardware1> a premis:HardwareAgent ;
      rdfs:label "Bookeye 4" ;
      premis:hasVersion "V1A" ;
      premis:hasNote "serial number: 00073236706e" .
```

Rights

Unlike the three other PREMIS Entities, the Rights Entity has no equivalent class in RDF. Instead, the Rights Entity is represented by an instance of the `premis:RightsBasis` class and, optionally, by an instance of `premis:RightsStatus` expressing the status of the Object regarding the rights basis it is (or was) governed by.

For the relationship between Rights and Agent entities, see section "Rights to Agent" above.

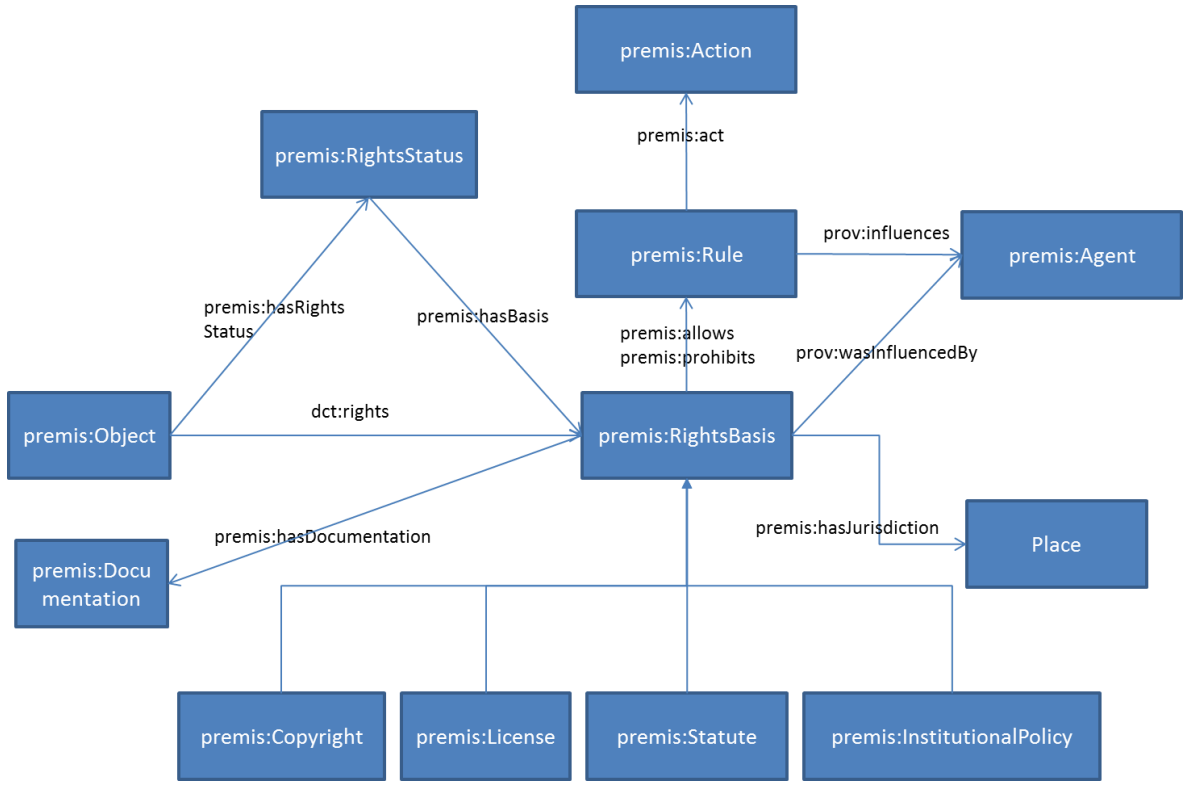


Fig. 23: A model for the PREMIS Rights entity

Rights Basis

Specifying the rights basis of any rights statement is mandatory. The rights basis governing the Object should be expressed by a `dct:rights` property pointing to an instance of one of the subclasses of the `premis:RightsBasis` abstract class (`premis:Copyright`, `premis:Statute`, `premis:License`, `premis:InstitutionalPolicy` or a locally defined class for other types of rights basis).

- Applicable dates are specific to the Object affected by the rights basis. See the section Rights status below.
- If a free-text note about the rights basis must be added, a `premis:hasNote` property should be attached either to the instance of a subclass of the `premis:RightsBasis` class or to the instance of the `premis:RightsStatus`

class, depending on whether the note applies to the rights basis or is specific to the Object.

- The jurisdiction to which the rights basis applies should be expressed by means of a `premis:hasJurisdiction` property attached to the instance of a subclass of the `premis:RightsBasis` class.
- Documentation concerning the rights basis should be expressed by means of a `premis:hasDocumentation` property pointing from an instance of `premis:RightsBasis` to an instance of `premis:Documentation`. If the documentation role has to be mentioned, implementers should define locally subproperties of `premis:hasDocumentation`.
- If the copyright status of the Object must be specified, subclasses of `premis:RightsStatus` should be used, either declared at http://id.loc.gov/vocabulary/preservation/copyrightStatus or locally defined.
- License terms should be expressed by means of a `premis:hasTerms` property attached to the instance of `premis:License`.
- Statute citation should be expressed by means of a `premis:hasCitation` property attached to the instance of `premis:Statute`.
- Implementers should define locally subclasses of the `premis:RightsBasis` class if the rights statement is not based on copyright, statute, license nor institutional policy.

Copyright example:

```
<rightsBasis1> a premis:Copyright ;
      premis:hasJurisdiction <http://ontologi.es/place/US> .
```

License example:

```
<rightsBasis2> a premis:Licence ;
      premis:hasDocumentation <documentation1> ;
      premis:hasTerms "Do not, under any circumstances, etc." .
```

Statute example:

```
<rightsBasis3> a premis:Statute ;
      premis:hasJurisdiction <http://ontologi.es/place/DE> ;
      premis:hasCitation "Gesetz über die deutsche
Nationalbibliothek vom 22. Juni 2006(DNBG)" ;
      premis:hasNote "Legal Deposit Law in Germany" .
```

Institutional policy example:

```
<rightsBasis4> a premis:InstitutionalPolicy ;
      premis:hasNote "80-year rule" ;
      prov:wasInfluencedBy <hul> .
```

Note: in the case a rights statement defined at rightsstatements.org/ applies to an Object, such rights statement may replace an instance of `premis:RightsBasis` and be the basis of any permission or prohibition.

Example:

```
<obj3> dct:rights <http://rightsstatements.org/vocab/InC/1.0/> .

<http://rightsstatements.org/vocab/InC/1.0/> a premis:Copyright .
```

Rights status

If the Object status regarding the rights basis has to be specified, an instance of `premis:RightsStatus` should be attached to the Object by means of a `premis:hasRightsStatus` property. It may be necessary to specify the relationship between the status and the rights basis, especially in the case of multiple rights basis affecting simultaneously or successively the Object; in such case, a `premis:hasBasis` property should point from the instance of `premis:RightsStatus` to the instance of `premis:RightsBasis`.

- Applicable dates should be mentioned by means of `premis:startDate` and `premis:endDate` properties attached to the instance of `premis:RightsStatus`. Note: though the use of standard conventions like ISO 8601 is recommended to express date and time values, the range of `premis:startDate` and `premis:endDate` is defined to be `rdfs:Literal` to accommodate uncertainty and open dates.[9]
- The status determination date should be specified by means of a `premis:hasDeterminationDate` attached to the instance of `premis:RightsStatus`.

Example:

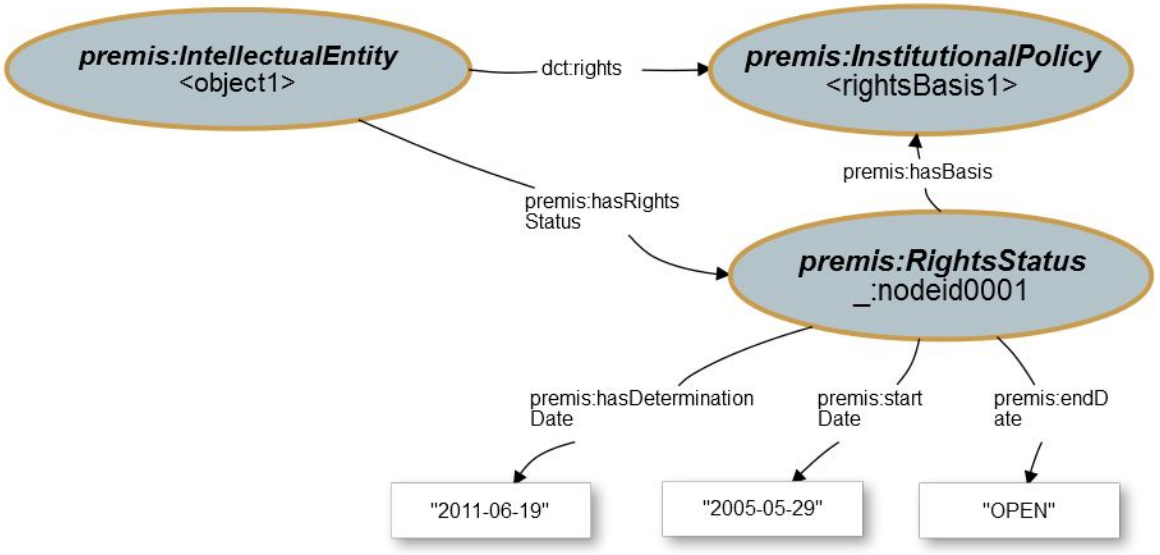


Fig. 24: The rights status of an Intellectual Entity

[9] For a standard expression of uncertain or open dates, see the Extended Date/Time Format (http://www.loc.gov/standards/datetime/).

```
<object1> a premis:IntellectualEntity ;
      dct:rights <rightsBasis1> ;
      premis:hasRightsStatus [
            a premis:RightsStatus ;
            premis:startDate "2005-05-29" ;
            premis:endDate "OPEN" ;
            premis:hasDeterminationDate "2011-06-19" ;
            premis:hasBasis <rightsBasis1> . ] .

<rightsBasis1> a premis:InstitutionalPolicy.
```

Rule (*rightsGranted*)

The `premis:Rule` class represents either a permission or a prohibition to perform an action. An instance of `premis:Rule` is attached to an instance of `premis:RightsBasis` by means of a `premis:allows` or `premis:prohibits` property, depending on whether the action is allowed or prohibited to the repository or to an Agent.

- The *rightsGrantedNote* semantic unit should be expressed by a `premis:hasNote` property attached to the instance of `premis:Rule`.
- The rule applicable dates (the *termOfGrant* semantic unit) should be expressed by means of `premis:startDate` and `premis:endDate` properties attached to the instance of `premis:Rule`.
- The action allowed or prohibited by the rule should be expressed by a `premis:act` property pointing to an instance of `premis:Action`, either declared at http://id.loc.gov/vocabulary/preservation/actionsGranted or locally defined.
- Other restrictions than the rule applicable dates applying to the permission or prohibition should be expressed by the `odrl:constraint` property pointing to an instance of `odrl:Constraint`.[10]

Note: however, if the restriction cannot be expressed by the ODRL vocabulary or in the case of a mapping from XML legacy data to RDF, a `premis:hasRestriction` property may be used to attach a free-text description of the restriction to the instance of `premis:Rule`.

Example 1 (an institutional policy allowing <agent1> to reproduce, modify and migrate an Object, but prohibiting its use for 80 years):

[10] See https://www.w3.org/TR/odrl-model/#constraint for further information about ODRL Constraints.

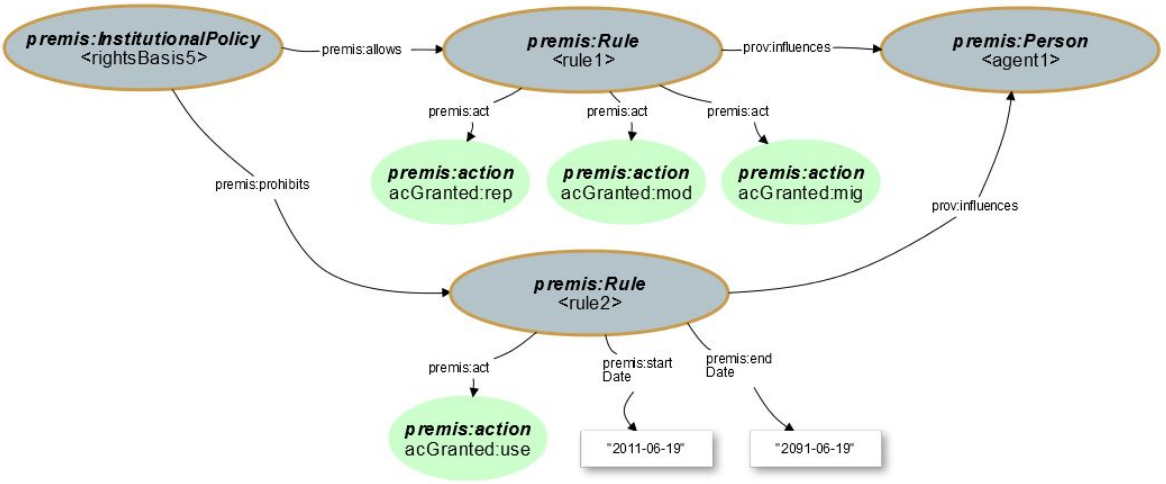


Fig. 25: Attaching Rules to a RightsBasis

```
<rightsBasis5> a premis:InstitutionalPolicy ;
     premis:allows <rule1> ;
     premis:prohibits <rule2> .

<rule1> a premis:Rule;
     premis:act acGranted:mig ;
     premis:act acGranted:mod ;
     premis:act acGranted:rep ;
     prov:influenced <agent1> .

<rule2> a premis:Rule
     premis:act acGranted:use ;
     premis:startDate "2011-06-19" ;
     premis:endDate "2091-06-19" ;
     prov:influenced <agent1> .

<agent1> a premis:Person .
```

Example 2 (a license granted by <agent1> allowing the repository to make only 3 copies of the Object for preservation purposes):

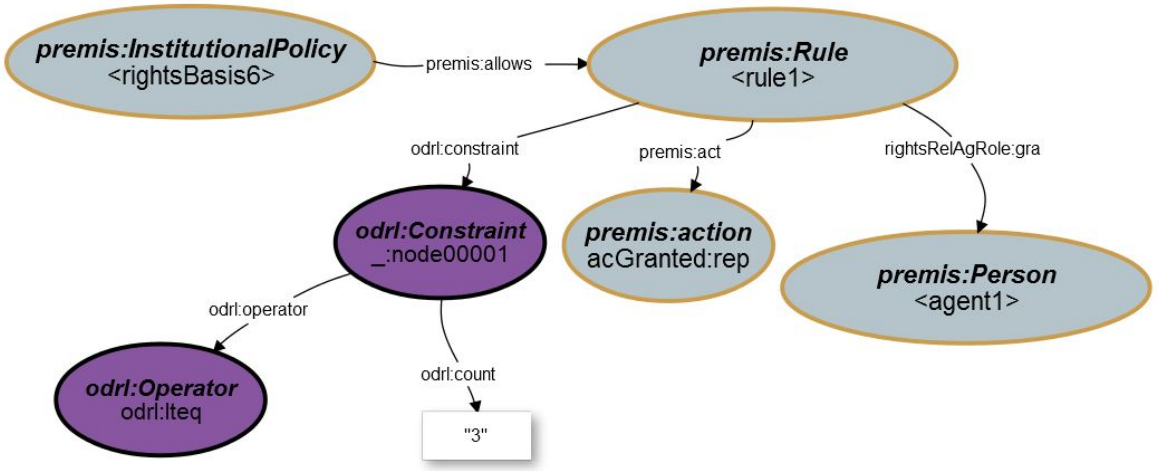


Fig. 26: Using Open Digital Rights Language (ODRL) to further refine a Rule

```
<rightsBasis6> a premis:License ;
     rightsRelAgRole:gra <agent1> ;
     premis:allows <rule1> .

<rule1> a premis:Rule ;
     premis:act acGranted:rep ;
     odrl:constraint [
          a odrl:Constraint ;
          odrl:count "3" ;
          odrl:operator odrl:lteq . ] .

<agent1> a premis:Person .
```