BIBFRAME 2.0

RDF Conventions

The following lists some of the RDF conventions used in developing the BIBFRAME 2.0 vocabulary. It is not an exhaustive list but tries to cover the most important conventions.

1) Datatype and Object Property

Any given BIBFRAME property is either a datatype property or an object property.

A *datatype property* is one whose object is always a literal. An example is bf:version.

<http://example.org/work/workX> bf:version "final cut" .

The object of any triple with property bf:version is always a literal (string) as in this example, and therefore the property is a datatype property.

An *object property* is one whose object is always a resource (and in particular a resource identified by a URI, or by a node-id in the case of a blank node). An example is bf:instanceOf.

```
<http://example.org/instance/instanceY>   bf:instanceOf
                                          <http://example.org/work/workX> .
```

In BIBFRAME every property is one or the other: for any given BIBFRAME property, the object should not be a literal in one triple and a resource in another. The object should always be a literal, or it should always be a resource. The motivation for this rule is the added complexity which would be imposed on a system, consuming BIBFRAME, if it needed to be prepared to receive both types of objects for a given property.

2) URIs and Labels

When referencing a resource, provide the URI, label, or both.

BIBFRAME defines many properties to be object properties with the intention that either the resource, or a label in lieu of the resource, or both, can be supplied. BIBFRAME and RDF syntax enable the inclusion of these reference methods. If the link is not known then only the label might be supplied, and if only the link is known it can be supplied. By supplying both the link and label, the label may be displayed to a user, who may then decide to follow the link for additional information about the resource, or may decide that the label is sufficient information and that it is not necessary to follow the link, and retrieval will thus be avoided.

3) URIs and Blank Nodes

BIBFRAME takes no position on the issue of URI vs. blank node.

While it is recognized that URIs are linked-data friendly and blank nodes are not, both are acceptable in BIBFRAME and the choice is an implementation decision.

4) Classes and Types

Classes are generally used to indicate type.

There are several categories of BIBFRAME resource that have types. Identifiers, for example, have types such as ISBN, ISSN, LCCN, etc. and variant titles have types such as abbreviated title, key title, etc.

In BIBFRAME 2.0, there is a single identifier property, bf:identifiedBy, and different classes defined for the different identifier types: bf:Isbn, bf:Lccn, and so on.

Some advantages of representing type as class rather than property are:

- **Reusability**. Consider identifiers for example. For every identifier expressed in BIBFRAME, a bf:Identifier resource is created. If it is created as a linked data resource (assigned a URI) then it may be accessed and reused outside of BIBFRAME. Allowing the class to reflect the identifier source means that the source will be known when it is used as such. If the source is conveyed only by the BIBFRAME property then that source will be known only when accessed in the BIBFRAME context.
- **Query Efficiency**. Expressing types as classes often makes the data more easily queried. "Find things of type X", for example, is simpler when X is a class rather than a property.
- **Graceful degradation**. Suppose a new note type is created, in some external namespace (ex:). If the new type were to be expressed by property, that might look like:

  ex:typeOfNote "note content"

  On the other hand if the type is expressed by class it might look like:

  bf:note [a ex:TypeOfNote ;
  rdfs:label "note content"]

If the receiving system does not recognize the namespace "ex", then in the first case, the statement will not make any sense at all. In the second case, the system will at least be able to recognize that it is a note (even though it may not understand the note type).

5) Reciprocal Properties

For any given BIBFRAME property, a reciprocal property should be defined, if appropriate.

This guideline recommends only that reciprocal properties be *defined,* not that they necessarily be *used.* Thus for example, if a Work points to one of its Instances (via bf:hasInstance), BIBFRAME takes no position on whether that Instance should point back to the Work (via bf:instanceOf); that would be an implementation decision. The guideline merely recommends that the reciprocal property be defined, when logical, so that it may be used, if desired.

6) Metadata about the Description

Do not represent metadata about a description of a resource as a property of the resource itself.

For any BIBFRAME statement or description, there may be metadata which pertains to that statement or description, rather than to the resource which is the subject of the statement/description: rules used, metadata creation date or date last revised, etc. Suppose for example the resource is a BIBFRAME Work. An RDF description of that Work might include bf:creationDate, intended to convey the date the description was created, rather than the date that the Work was created, and it should be clearly distinguished from statements describing the Work.

7) Proliferation of Properties

Avoid proliferation of properties by defining a single general property when multiple potential properties have the same meaning.

8) rdfs: and rdf: Properties

Use rdf:value and rdfs:label as appropriate when supplying the value of a resource.

9) Formal constraints

Explicit domains and ranges for a property are generally not specified.

BIBFRAME practice in general is to not define a domain or range for a property. There are some obvious exceptions, for example, for property bf:hasInstance, the domain is Work and the range is Instance, because clearly, these constraints are appropriate. But

in general, explicitly defined domains and ranges can have unintended, over-constraining effects.

When defining a property, the class of resources expected to be subjects of that property, as well as the class of expected values of that property, should be well-document -- what the domain and range of the property would likely be if the domain and range were formally specified. Thus for documentation purposes, properties are noted as “property of” and have “expected value” to express the usual domain and range, but these are not intended as domains and ranges to be enforced.

10) Naming Properties and Classes

Class names are nouns and property names suggest verbs.

A Class name should always be a noun. A property name should suggest a verb. It need not actually be a verb, for example, the (hypothetical) property “age” might indicate the age of a person. In this case the prefix “has” is implied, so the meaning would be as if the name were hasAge.