

Appendix 5 DOI Resource Metadata Declaration

This appendix describes the DOI Resource Metadata Declaration (RMD) a non-mandatory model with comprehensive semantics and an XML schema for declaring metadata of any resource, of any complexity. This RMD is based on the indecs framework and development. Its principal function is for metadata interchange between RAs, but it may be used for any messaging purpose required by an RA (for example, for general metadata capture or delivery). A detailed understanding of the RMD is not required by DOI assigners or developers; mapping and related services may be made available through IDF or specialist services.

© International DOI Foundation 2004

The DMS offers RAs a standard form of extended Metadata Declaration, the **DOI Resource Metadata Declaration (RMD)**, whose primary purpose is for metadata interchange between RAs. This Declaration contains a comprehensive set of attribute semantics (drawn from the iDD), variant methods of attribution and syntax in the form of an XML Schema to provide a comprehensive method for making interoperable Resource description metadata to any level of complexity. The RMD is designed for the description of Resources: that is, creations which are identified with a DOI. However, the attribute structure can be easily adapted to provide a framework for standard Metadata Declarations for other kinds of entity (for example, Agents, Events or Places) if required in future.

The structure and Terms of the RMD are derived from the work of the indecs framework , the Contecs:DD consortium (the latter through the development of the MPEG-21 Rights Data Dictionary), and Ontologyx's OntologyX, and reflect the fundamental Terms of the iDD.

A5.1 Basic Attributes of the RMD

The RMD describes ten high-level types of Attributes for any Resource:

Basic Attributes of the RMD

| | Attribute Type | Description | Structure of basic Attribute Value |
|---|-----------------------|--|---|
| <i>Dependent Attributes (existing only in relation to the Resource)</i> | | | |
| 1 | Identifier | A unique label which makes the Resource referable. | Uncontrolled alphanumeric string. Structure may be governed by rules. |
| 2 | Name | A non-unique label which makes the Resource referable. | Uncontrolled alphanumeric string. Structure may be governed by rules. |
| 3 | Quantity | A numeric quantification of the Resource. | Number which may be supported by a UnitOfMeasure. |
| 4 | Annotation | A textual description of, or note or comment about the Resource. | Uncontrolled text string. |
| <i>Independent Attributes (existing independently of the Resource)</i> | | | |
| 5 | Category | A classification or type to which the Resource belongs, or a quality which it has. | Allowed value Code or Name. |
| 6 | Context | An event or situation affecting the Resource. | Identifier or Name of a related Context. |
| 7 | Agent | An entity acting in relation to the Resource. | Identifier or Name of a related Agent. |
| 8 | Time | A Time in relation to which | Identifier or Name of a related |

| | | | |
|----|-----------------|--|---|
| | | something happened to the Resource. | Time. |
| 9 | Place | A Place in relation to which something happened to the Resource. | Identifier or Name of a related Place. |
| 10 | Relative | Another Resource related to the Resource. | Identifier or Name of a related Resource. |

The basic RMD attributes may be simply illustrated like this:

Basic Resource Metadata Declaration model

| |
|---|
| <p><i>Resource</i></p> <p>Name [0-n] [Value]</p> <p>Identifier [1-n]* [Value]</p> <p>Quantity [0-n] [Value]</p> <p>Annotation [0-n][Value]</p> <p>Category [0-n] [Value]</p> <p>Context [0-n] [Value]</p> <p>Agent [0-n] [Value]</p> <p>Time [0-n] [Value]</p> <p>Place [0-n] [Value]</p> <p>Resource [0-n] [Value]</p> |
|---|

*The only mandatory attribute of a DOI RMD is (self-evidently) a DOI, which is a Subtype of Identifier.

The RMD is based on *Subtypes* and combinations of the ten basic attributes. Using these ten attributes, and subtypes from the iDD, the RMD model enables the building of Metadata Declarations (including the Kernel) going to any level of complexity within a relatively simple common framework. One of the benefits of structuring a Declaration according to the ten basic attributes is that each has consistent behaviour.

A5.2 Semantic engineering of RMDs

RMDs are developed through applying a number of basic processes to the ten basic attributes of a Term. These are listed in the table below, described in the rest of this section. Together they provide a set of tools for what might be termed the *semantic engineering* of metadata declarations. Note that these processes are not schema-dependent: they rely only on the semantic relationships explicitly mapped in the iDD. They may be represented in different ways according to the syntax and naming conventions of any given schema.

Processes of Semantic engineering for RMDs

| | Semantic Process | Description |
|---|---------------------------------|---|
| 1 | Subtyping Attributes | To generate new Attributes as specialized types of basic Attributes. |
| 2 | Substituting Types | To simplify a declaration by substituting a Type for one of its Archetypes (parents). |
| 3 | Chaining Attributes | To give Attributes to an Attribute. |
| 4 | Transferring Attributes | To create a new Attribute by compressing a Chain. |
| 5 | Reciprocating Attributes | To reverse the relationship of Entity and Attribute. |
| 6 | Compositing Attributes | To group two or more distinct elements as a single Attribute. |

| | | |
|---|---------------------------------|--|
| 7 | Allowed Values and Codes | To provide controlled Values and Codes for an Attribute. |
|---|---------------------------------|--|

A5.2.1 Subtyping Attributes

Basic RMD attributes are developed through **Subtypes**: (that is, by *specializing* some aspect of an attribute of a Term to give it a more precise meaning. For example, a **Title** is a Subtype of Name which has been specialized so that it is only valid for a Creation (and not, for example, to a plant or animal). In similar ways a **DOI** is a Subtype of Identifier, **Height** is a Subtype of Measure, **Description** is a Subtype of Annotation, **Creator** is a Subtype of Agent, **Translation** is a Subtype of Resource, and so on. Each Subtype is more limited in its scope than its parent or *Archetype*. Subtypes can be specialized to any level.

All Subtypes in the iDD may be used in RMDs, and others may be added to the iDD through the mapping of new Terms from other schemes. Subtyping is the most important mechanism of the iDD for defining new Terms.

Subtypes can be represented in the RMD as shown in this example:

RMD Subtyped Attributes - example

| |
|--|
| <pre> Resource Identifier ["10.9999/12345"] Subtype [DOI] Name ["My picture"] Subtype [Title] Quantity ["54"] Subtype [DigitalCapacity] Subtype [SizeInKB] Category [jpeg] Subtype [MimeType] </pre> |
|--|

In this example one *extended* Subtype hierarchy has also been shown: Quantity *hasSubtype* DigitalCapacity *hasSubtype* SizeInKB. Subtype hierarchies can go on to any required level of granularity, with each Type being progressively more specialized than its parent.

Types of attributes may in turn have their own Subtypes: for example, a Description might have a Subtype called a **SummaryDescription**, a Creator may have a Subtype called an **Author**, and so on. This "specialization" of Subtypes may go on through any number of levels. In principle there is no limit to the number of Subtypes of any attribute.

The use of Subtypes can be demonstrated in the DOI Kernel in figure below. Each of the Kernel elements is a Subtype of one of the RMD Basic Attributes (Kernel elements in **bold**):

RMD Subtyped Resource Attributes in the Kernel Metadata Declaration

| |
|--|
| <pre> Resource Identifier [Value] [0-n] Subtype [resourceIdentifier] Identifier [Value] [1] Subtype [DOI] Name [Value] [0-n] Subtype [resourceName] Category [Value] [1] Subtype [structuralType] Category [Value] [1-n] Subtype [resourceType] </pre> |
| <pre> Category [Value] [1-3] Subtype [mode] </pre> |

```
Agent [Value] [1-n]
Subtype [primaryAgent]
```

A5.2.2 Substituting Subtypes

Declarations can be simplified radically by substituting the name of the Subtype directly for its parent (or *Supertype*) within the RMD. This can be done from any point in the Subtype hierarchy to any point above it. In the RMD, Subtypes are substituted directly for one of the ten basic attributes. The example given in A5.2.1 can therefore be written much more simply as follows:

RMD Substituted Types - example

```
Resource
  DOI ["10.9999/12345"]
  Title ["My picture"]
  SizeInKB ["54"]
  MimeType [jpeg]
```

Because the Subtype hierarchies are all explicitly recorded in the iDD, the substitution of Subtypes for their Supertypes or (vice versa) can be managed in an automated fashion to support the expansion or contraction of attributes from one form of Declaration to another, and thereby transformation between schemes.

The figure below shows the Kernel elements with Subtypes (from A4.2.1) substituted as direct attributes of the Resource. In this figure (and those which follow) the parent basic attributes are given in italics to show how they fit into the now-hidden underlying "ten element" structure.

RMD Substituted Subtypes in the Kernel

```
Resource
[Identifier]
  resourceIdentifier [Value] [0-n]
  DOI [Value] [1]
[Name]
  resourceName [Value] [0-n]
[Category]
  resourceType [Value] [1-n]
  structuralType [Value] [1]
  mode [Value] [1-3]
[Agent]
  primaryAgent [Value] [1-n]
```

A5.2.3 Chaining Attributes

Each of the basic attributes may be an entity in its own right with attributes drawn from the same group of ten. For example, a *Category* may have a *Name*; an *Agent* may have a *Category*; an *Annotation* may have an *Agent*; or a *Measure* may have a related *Time*. This process may go on to any level of granularity, creating **Attribute chains**. For example, the Resource used in the example above may have Agent details in a simple Attribute chain as follows:

RMD Attribute Chain - example

```
Resource
  DOI ["10.9999/12345"]
  Title ["My picture"]
  SizeInKB ["54"]
  MimeType [Jpeg]
  Creator ["John Smith"]
```

```
Relative ["Photographers Inc"]
Subtype ["AffiliatedOrganization"]
```

This can be simplified in turn by substituting Subtypes:

RMD Attribute Chain with Substituted Subtype - example

```
Resource
  DOI ["10.9999/12345"]
  Title ["My picture"]
  SizeInKB ["54"]
  MimeType [Jpeg]
  Creator ["John Smith"]
  AffiliatedOrganization ["Photographers Inc"]
```

The eighth Kernel element (**AgentRole**) is an example of a chained attribute, as shown when it is added in the figure below:

RMD Attribute Chain in the Kernel

```
Resource
  [Identifier]
    resourceIdentifier [Value] [0-n]
    DOI [Value] [1]
  [Name]
    resourceName [Value] [0-n]
  [Category]
    resourceType [Value] [1-n]
    structuralType [Value] [1]
    mode [Value] [1-3]
  [Agent]
    primaryAgent [Value] [1-n]
      Category [Value] [1-n]
      Subtype [AgentRole]
```

This chain is then simplified by Substituting the Type for its parent, as shown in the figure below:

RMD Attribute Chain with Substituted Type in the Kernel

```
Resource
  [Identifier]
    resourceIdentifier [Value] [0-n]
    DOI [Value] [1]
  [Name]
    resourceName [Value] [0-n]
  [Category]
    resourceType [Value] [1-n]
    structuralType [Value] [1]
    mode [Value] [1-3]
  [Agent]
    primaryAgent [Value] [1-n]
      [Category]
```

Some of the other Kernel attributes will need further attributes, depending on how they are to be referenced. Some will require labels: the **primaryAgent** will be identified either by a **name** or **identifier**, as in the figure below:

Further RMD Attribute Chain in the Kernel

| |
|---|
| Resource |
| [Identifier] |
| resourceIdentifier [Value] [0-n] |
| DOI [Value] [1] |
| [Name] |
| resourceName [Value] [0-n] |
| [Category] |
| resourceType [Value] [1-n] |
| structuralType [Value] [1] |
| mode [Value] [1-3] |
| [Agent] |
| primaryAgent [Value] [1-n] |
| [Category] |
| agentRole [Value] [1-n] |
| [Name] |
| agentName [Value] [0-n] |
| [Identifier] |
| agentIdentifier [Value] [0-n] |

Some attributes (including those just added) may have their own attributes in the form of Subtypes, as shown in the figure below:

Further RMD Attribute Chain in the Kernel

| |
|---|
| Resource |
| [Identifier] |
| resourceIdentifier [Value] [0-n] |
| DOI [Value] [1] |
| [Name] |
| resourceName [Value] [0-n] |
| [Category] |
| resourceType [Value] [1-n] |
| structuralType [Value] [1] |
| mode [Value] [1-3] |
| [Agent] |
| primaryAgent [Value] [1-n] |
| [Category] |
| agentRole [Value] [1-n] |
| [Name] |
| agentName [Value] [0-n] |
| Subtype [Value] [0-1] |
| [Identifier] |
| agentIdentifier [Value] [0-n] |
| Subtype [Value] [0-1] |

A5.2.4 Transferring Attributes

Another very common form of simplification of Declarations is the compression of Attribute chains into a single **Transferred Attribute**. In this process, an attribute of one thing is transferred to another in order to “flatten out” the hierarchical structure. In the illustration above, the fact that “John Smith” is affiliated to “Photographers Inc” may be shown by compressing the chain into a single Attribute “CreatorAffiliatedOrganization”, which becomes an Attribute of the Resource. Attribution is thereby “displaced” from the Creator to the Resource. By the combination of substituting Types and Transferred Attributes, the “flattest” practical form of Declaration can be produced. For example:

RMD Transferred Attribute - example

```
Resource
  DOI ["10.9999/12345"]
  Title ["My picture"]
  SizeInKB ["54"]
  MimeType [Jpeg]
  Creator ["John Smith"]
  CreatorAffiliatedOrganization ["Photographers Inc"]
```

No semantic value is lost as the iDD can contain the mapping of the new Displaced Attribute to its chain. However, there are clear limitations to the compression of attribute chains. For example, if in this illustration there were more than one Creator and more than one AffiliatedOrganization, it would not be clear which one was Affiliated to which organization. Wherever a Resource's attributes have many-to-many Relationships among themselves, Transferred Attributes will have limited value.

A5.2.5 Reciprocating Attributes

The ten basic attributes all describe "Has..." relationships, although for simplicity they are shown here as (for example) "Title" rather than "hasTitle" etc. Sometimes it is necessary to show the reverse: that the Resource (or one of its attributes) being described is itself an attribute of something else: in other words, that something else "Has..." the Resource. This is called a *Reciprocal Attribute*. In the illustration above, for example, it may be that we wish to state that "John Smith" is a Member of an Organization called "Photographers Union" – in other words that "Photographers Union" *hasMember* "John Smith". Now, it is technically possible to do this within the RMD and iDD by giving "John Smith" an attribute such as *OrganizationOfWhichEntityIsMember*, but this is cumbersome and unintuitive. Instead, all entity-attribute relationships can be reversed by using a reciprocal Term from the iDD, which normally adds *Is...Of* to the attribute name. The reciprocal of *Member* is therefore *IsMemberOf*. In our example:

RMD Reciprocated Attribute - example

```
Resource
  DOI ["10.9999/12345"]
  Title ["My picture"]
  SizeInKB ["54"]
  MimeType [Jpeg]
  Creator ["John Smith"]
  IsMemberOf ["Photographers Union"]
  CreatorAffiliatedOrganization ["Photographers Inc"]
```

A5.2.6 Compositing Attributes

Attributes are often represented by two or more elements in a structured group, known in RMD (as in ONIX) as a **Composite Attribute** or *Composite* for short. Composite Attributes are defined as Terms in their own right in the iDD. RAs may define their own Composites which can be mapped to iDD like any other Term (for example, the Crossref AP has Composites for **doi_data**, **article**, **journal** and **author** among others). ONIX employs a large number of Composites. Composites may also contain other Composites.

For example, a MeasureComposite may be described in a Composite comprised of these basic attributes and Subtypes, as shown in the figure below:

Composite Attribute with unsubstituted Subtypes

```
MeasureComposite
  Quantity [Value] [1]
  Subtype [AllowedValue] [1]
  Category [AllowedValue] [1]
  Subtype [UnitOfMeasure]
  Category [AllowedValue] [1]
  Subtype [Precision]
```

which after substituting Subtypes looks like this:

Composite Attribute with Substituted Subtypes

```
MeasureComposite
[Quantity]
    Quantity [Value] [1]
[Category]
    Subtype [AllowedValue] [1]
    UnitOfMeasure [AllowedValue] [1]
    Precision [AllowedValue] [1]
```

For example:

Composite Attribute example

```
MeasureComposite
[Quantity]
    Quantity ["54"]
[Category]
    Type [DigitalCapacity]
    UnitOfMeasure [KB]
    Precision [Exactly]
```

This can be shown in place within the example given in A5.2.5:

RMD Composite Attribute - example

```
Resource
    DOI ["10.9999/12345"]
    Title ["My picture"]
    QuantityComposite
        Quantity ["54"]
        Subtype [DigitalCapacity]
        UnitOfMeasure [KB]
    MimeType [Jpeg]
    Creator ["John Smith"]
        IsMemberOf ["Photographers Union"]
    CreatorAffiliatedOrganization ["Photographers Inc"]
```

A5.2.7 Allowed Values and Codes

Some attribute values are drawn from sets of *allowed* or *controlled* values, usually with representative *Codes*. This is always true for Subtypes, and usually for Categories, but is not uncommon for Places and Agents. All AllowedValues are mapped as Terms in iDD. Allowed values (often presented as *code lists*) are valuable for interoperability as they may be drawn from widely-used standards (such as ISO Currency Codes), or they may be mappable to other code lists through the iDD.

Allowed values are shown in the RMD using the iDD Term *AllowedValue*. In turn these Values will be drawn from a set of Terms which is known as an *AllowedValueSet* or (when they have codes) a *CodeSet*. Again this can be illustrated in the Kernel, which can now be shown in full as an RMD in the figure below, with Subtypes substituted as appropriate. Note that in some cases an RA may specify any AllowedValueSet, while in others (*structuralType* and *mode*) the AllowedValues are mandated in the Kernel and provided from iDD.

Complete Kernel, showing addition of AllowedValues and AllowedValueSets

| |
|--|
| <i>Resource</i> |
| <p><i>[Identifier]</i></p> <p>resourceIdentifier [Value] [0-n] Subtype [AllowedValue] [1] AllowedValueSet [Value] [1]</p> <p>DOI [Value] [1]</p> <p><i>[Name]</i></p> <p>resourceName [Value] [0-n]</p> <p><i>[Category]</i></p> <p>resourceType [Value] [1-n] structuralType [AllowedValue] [1] mode [AllowedValue] [1-3]</p> <p><i>[Agent]</i></p> <p>primaryAgent [Value] [1-n]</p> <p><i>[Category]</i></p> <p>agentRole [Value] [1-n]</p> <p><i>[Name]</i></p> <p>agentName [Value] [0-n] Subtype [AllowedValue] [0-1]</p> <p><i>[Identifier]</i></p> <p>agentIdentifier [Value] [0-n] Subtype [AllowedValue] [0-1]</p> |

The standard iDD AllowedValues for the Kernel are as follows:

AllowedValues for **structuralType**:

PhysicalFixation
 DigitalFixation
 Performance
 Abstraction
 RestrictedType

AllowedValues for **mode**:

Audio
 Visual
 Audiovisual
 Abstract
 RestrictedMode

A5.3 Contexts as Attributes

Four of the ten basic attributes shown have a special relationship: three of them (**Agent**, **Time**, **Place**) are attributes of the first (**Context**), as is **Resource** itself, according to the <indec> Context Model, as shown in the figure above:

Context Attributes

Context [Value]
 Resource [Value] [0-n]
 Agent [Value] [0-n]
 Time [Value] [1-n]
 Place [Value] [1-n]

(Note that Agent is actually a Subtype of Resource in iDD: it has been substituted here). The Context is at the heart of the <indec> Data Dictionary model where it is the key to defining and mapping many Terms. This section explains and illustrates how Contextual attributes may be used in the RMD.

A Context is either an **Event** in which something changes, or a **Situation** in which something remains the same. For example, if someone translates a text, then there is or was a Context in which an Agent (the **Translator**) created the new Resource (the **Translation**) from the original Resource (the **Source**) at a certain **Time** and **Place**. It is through this Context of a **TranslatingEvent** that the Resource acquires its attribute relationship with the other elements. There are many circumstances where it is useful or essential to identify a Context explicitly (for example, in workflow management or rights management).

However, in Resource Metadata it is common for the Context to be ignored as an entity, and the activity (the “verb” – in this example, **Translate**) at the heart of a Context to be “bundled” along with one or more of its attributes into a single Term. For example, the Event of Publishing is often shown through a number of different attributes such as **Publisher** (Agent), **DateOfPublication** (Time) and **PlaceOfPublication** (Place), rather than through an explicit description of the Event itself. This may be described as a **Resource-based view** rather than a **Context-based view**.

Both views are widely used and valid. Each has advantages and disadvantages. The two different approaches are illustrated below using a fictitious example of book publication. For simplicity the full attribute Subtype hierarchy is *not* shown but is shown in footnotes. First, an example where the publication Event is made explicit¹:

```
Resource
  PublishingEvent ["#1"]
    Publisher ["IDG Books Worldwide, Inc"]
    Date ["1996"]
    Place [California, USA]
```

Secondly, where the Event is implicit²:

```
Resource
  Publisher ["IDG Books Worldwide, Inc"]
  DateOfPublication ["1996"]
  PlaceOfPublication [California, USA ]
```

Either of these approaches is supported by the RMD.

In this example the second approach is more compact, as all three elements are direct attributes of the Resource, whereas the first requires the extra step of indirection and some means of identifying or naming a Context (shown here as “#1”). For this reason the second approach is the most common in simple Resource description schemes.

However, in cases where the metadata becomes more complicated, the Contextual approach can become more efficient. If, for example, we need to show that the book had been published on different dates in two different countries, a level of indirection would have to be introduced, perhaps like this:

¹ Showing the Subtype hierarchy, this would look as follows:

```
Resource
  Context ["#1"]
    Subtype [PublishingEvent]
      [Agent]
        Publisher [IDG Books Worldwide, Inc"]
          [Time]
            Date ["1996"]
          Place ["California, USA"]
```

² Showing the Subtype hierarchy, this would look as follows:

```
Resource
  [HasAgent]
    Publisher ["IDG Books Worldwide, Inc"]
  [HasTime]
    DateOfPublication ["1996"]
  [HasPlace]
    PlaceOfPublication [California, USA]
```

Resource
 Publisher ["IDG Books Worldwide, Inc"]
 DateOfPublication ["1996"]
 PlaceOfPublication ["California, USA"]
 DateOfPublication ["1998"]
 PlaceOfPublication [Paris, France]
 DateOfPublication ["1999"]
 PlaceOfPublication [Moscow, Russia]

(Note that in this example the *PlaceOfPublication* is an attribute of the *DateOfPublication*, but it could just as easily have been done the other way round).

Our example now contains a level of indirect attribution. Then suppose that the book has different publishers in one of the countries:

Resource
 Publisher ["IDG Books Worldwide, Inc"]
 DateOfPublication ["1996"]
 PlaceOfPublication [California, USA]
 DateOfPublication ["1998"]
 PlaceOfPublication ["Paris, France"]
 Publisher ["XYZ Books"]
 DateOfPublication ["1999"]
 PlaceOfPublication [Moscow, Russia]

We must now have *two* levels of indirect attribution in order to ensure that the metadata is unambiguous. However, if we say the same thing with explicit Contexts, we need only a single level of indirection:

Resource
 PublishingEvent ["#1"]
 Publisher ["IDG Books Worldwide, Inc"]
 DateOfPublication ["1996"]
 PlaceOfPublication [California, USA]
 PublishingEvent ["#2"]
 Publisher ["IDG Books Worldwide, Inc"]
 DateOfPublication ["1998"]
 PlaceOfPublication [Paris, France]
 PublishingEvent ["#3"]
 Publisher ["XYZ Books"]
 DateOfPublication ["1999"]
 PlaceOfPublication [Moscow, Russia]

Where Contexts have multiple Agents, Times or Places, or involve multiple Resources, it is often more efficient to use the explicit Context approach to organize Resource metadata. In the IDD, the Context structure is one of the keys to interoperability.

The RMD supports either approach, and importantly provides (through the IDD) an internal mapping between them, so that it can be possible to transform metadata from one basis to the other automatically without loss of semantic value.

A5.4 Journal-RMD XML schema

The first RMD to be expressed in the form of an XML schema, called Journal-RMD and dealing with Journal metadata, is under development.