Product Definition Resource Formats

This note describes the resources contributed to the index by the RELM Product Definition capability through a Tracked Resource Set.

These resources may be read in RDF/XML, Turtle, and JSON formats using the appropriate request headers. This note does not define the semantics of PUT, POST, or DELETE requests; in many cases such requests will fail.

## Namespaces

Examples in this document use [Turtle](http://www.w3.org/TeamSubmission/turtle/), with the following namespaces:

|  |  |
| --- | --- |
| *Prefix* | *Namespace URI* |
| rdf | [http://www.w3.org/1999/02/22-rdf-syntax-ns#](http://www.w3.org/1999/02/22-rdf-syntax-ns) |
| rdfs | [http://www.w3.org/2000/01/rdf-schema#](http://www.w3.org/2000/01/rdf-schema) |
| dcterms | <http://purl.org/dc/terms/> |
| xsd | [http://www.w3.org/2001/XMLSchema#](http://www.w3.org/2001/XMLSchema) |
| vvc | TBD – Rational’s version, variant and configuration namespace |
| pd | [http://jazz.net/ns/pd#](http://jazz.net/ns/pd) |

## Common Properties

RELM PD resources follow the recommendations in the W3C [Linked Data Platform](http://www.w3.org/2012/ldp/wiki/Main_Page), [OSLC Core 2.0](http://open-services.net/bin/view/Main/OslcCore), and Rational best practices.

All RELM resources may have the common properties indicated in the table below; the specification of each individual resource type omits these common properties. A resource may have additional properties not described in this note.

The Range column of this and subsequent tables indicate the likely type of the object of a triple. However, in accordance with the general principles of linked data, clients should not assume an object is of that type, but should handle the actual data type.

|  |
| --- |
|  **Common Properties**  |
| *Prefixed Name* | *Occurs* | *Value type* | *Range* | *Description* |
| rdf:type | one or many | Resource | RDF type URI | The RDF type of this resource |
| dcterms:contributor | zero or many | Resource | foaf:Person | Owner of, or contributor to, this resource. |
| dcterms:creator | zero or many | Resource | foaf:Person | Creator or creators of resource. |
| dcterms:created | zero or one | DateTime | n/a | Timestamp of resource creation |
| dcterms:description | zero or one | XMLLiteral | Rich text in XHTML content valid inside a <div> element | Descriptive text about this resource. |
| dcterms:identifier | exactly one | String | n/a | A unique identifier for this resource. Not intended for end-user display. |
| dcterms:isVersionOf | exactly one | Resource | pd:Item | For a resource that is a version of some other resource: a reference to the concept resource of which this is a version. The subject of this triple is the version resource, not the concept resource. The subject of almost all other triples is the concept resource, representing the idea that a version resource captures the concept resource at some state. |
| dcterms:modified | zero or one | DateTime | n/a | Timestamp of latest resource modification |
| dcterms:title | zero or one | XMLLiteral | Rich text in XHTML content valid inside a <span> element | Human-readable name or title of the resource. |

## Products and their parts: Items

The product definition tool allows the user to define products and their parts, providing structures similar to a bill of materials. These structures are directed acyclic graphs (DAGs), though for simplicity they are represented as trees in the user interface.

Each product line, or product, or part thereof is represented by a concept resource with rdf:type = pd:Item, where pd:Item is a subclass of rdfs:Class. The term **Item** is borrowed from the STEP (ISO 10303) model used in PDM tools.

Since the product definition graphs are always presented in the context of a configuration, each node in these graphs corresponds to a version of one of these Item concept resources. These resources have no pre-defined properties beyond the standard ones described above, but may have arbitrary user-defined properties. These user-defined properties are not used in a strictly correct manner; the properties are not properties of the class, but of all instances of that class (for example, we allow the user to define a wheelSize property on the wheel class, where that property applies to all instances of the class). Although incorrect, it is felt this will not cause confusion, and results in simpler queries.

|  |
| --- |
|  **Properties of Item resources** |
| *Prefixed Name* | *Occurs* | *Value type* | *Range* | *Description* |
| *any* | zero or more | *any* | *any* | Custom properties other than links allow user-selectable URIs for the predicates. In RELM 1.0, link properties are not user-definable but use the fixed predicate dcterms:references.Custom link properties to other resources (typically but not necessarily those found in the index) may be used to relate this Item to its associated requirements, designs, etc. |

## Configurations

The RELM product definition application uses the Versions, Variants, and Configuration Management Service (VVC) to manage configurations, and determine the versions of each concept resource used in those configurations. RELM posts dimension point mappings to VVC, and can retrieve version information and mappings from VVC.

Since RELM creates variants of product definition resources, and allows fine-grained reuse of the variants, VVC resources and predicates must be read in conjunction with RELM resources to understand the nature of the resulting structures. For this reason, some of the examples at the end of this note include some possible VVC resources. However, the VVC data is not part of the RELM product definition resources, so is not defined here, and may vary from the examples shown here.

## Views

In Product Data Management (PDM) or Product Line Engineering (PLE), it is common to define hierarchies of products and their constituent parts – like a Bill of Materials. A single product or constituent part may have more than one view, where each view is a different hierarchy or filtered view of a single combined hierarchy. For example, a car might have a mechanical view and an electrical view: the mechanical parts and their structures are not the same as the electrical parts and structures.

A **View** is a concept resource with rdf:type=pd:View that defines the parts or members of a particular view of some version of a parent Item. In RELM 1.0, there is likely to be only one view per Item version.

View resources are versioned; different versions of a view may be used in different VVC configurations, allowing controlled reuse of views, and allowing different versions and variants of the view in those configurations.

A View is also a [Linked Data Platform Container](http://www.w3.org/TR/ldp/#linked-data-platform-container), specifying the membership and ordering predicates used for the members of this view.

|  |
| --- |
|  **Properties of a View** |
| *Prefixed Name* | *Occurs* | *Value type* | *Range* | *Description* |
| pd:isViewOf | exactly one | Resource | any | A reference to the parent resource of which this is a view.  |
| bp:membershipPredicate | zero or one | URI | Predicate | The predicate used to define membership in this view. |
| bp:containerSortPredicates | zero or one | Resource | rdf:List | A list of predicates that a client can use for sorting the members of this view |
| rdfs:member,dcterms:hasPart, oras defined by bp:membershipPredicate | zero or more | Resource | pd:Part | A reference to a member of the view (a constituent part). The recommended predicate for RELM views is dcterms:hasPart. |

## Parts

Each member of a view is represented as a non-versioned resource with rdf:type=pd:Part (in addition to the rdf:type predicate used to reference the concept resource for this part. Note that each resource of this type is specific to some context, and is not shared between multiple contexts; it therefore represents a part in context, where a context might be a View, or some other structure such as a requirements collection. Part resources may have additional properties used for ordering the members of the View containers.

|  |
| --- |
|  **Properties of a Part** |
| *Prefixed Name* | *Occurs* | *Value type* | *Range* | *Description* |
| dcterms:title | zero or one | XMLLiteral | Rich text in XHTML content valid inside a <span> element | Optional human-readable label showing the usage of this part. |
| rdf:type  | typically two | Resource | any | One type is pd:Part.; the other type is the concept resource reference for the part. This could be a pd:Item, but could be of any type; clients and applications must delegate handling of the relevant resources to the appropriate application or capability. |
| vvc:foreignConfiguration | zero or one | Resource | vvc:Configuration | The VVC configuration to be used for the member resource; if not specified, the child inherits the same configuration as the parent or current context. |
| *any* | zero or more | *any* | *any* | Instantiation and ordering properties may be present – that is, properties that apply to this specific usage of the constituent part, such as the inflation pressure for this usage of a tire, or properties that are used for ordering of view members. |

## RDF Vocabulary

Each type and predicate defined in this document with the pd namespace prefix shall be a readable URI. A GET on such a URI shall respond with an RDF vocabulary document; both HTML and RDF formats shall be supported.

## Examples

@prefix dcterms: <http://purl.org/dc/terms/>.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

@prefix bp: <http://open-services.net/ns/basicProfile#>.

@prefix pd: <http://jazz.net/ns/pd#>.

@prefix : <http://example.org/toybike/>.

pd:Item rdfs:subClassOf rdfs:Class.

pd:View rdfs:subClassOf bp:Container.

# We start with a version-unaware look at a bike and its parts

# First, we define a bike item.

:bike

 a pd:Item.

# Now, we define a compositional view of that bike;

# in this compositional view, the bike has two wheels,

# with no distinction between front and back wheels.

:viewOfBike

 a pd:View;

 pd:isViewOf :bike;

 bp:membershipPredicate dcterms:hasPart;

 dcterms:hasPart :frontWheel, :backWheel .

:frontWheel

 a pd:Part, :wheel.

:backWheel

 a pd:Part, :wheel.

# Define the wheel item, with a user-defined wheel size property.

:wheel

 a pd:Item;

 :wheelSize 15 .

# And a compositional view of the wheel, with tire and hubcap.

:viewOfWheel

 a pd:View;

 pd:isViewOf :wheel;

 bp:membershipPredicate dcterms:hasPart;

 dcterms:hasPart :tirePart, :hubcapPart.

:tirePart

 a pd:Part, :tire.

:hubcapPart

 a pd:Part, :hubcap.

# The tire and hubcaps are leaf nodes in this example,

# so they have no views, just items. The hubcap is colored silver.

:tire a pd:Item.

:hubcap a pd:Item; :color :silver.

# We associate the bike with a requirements module in DOORS,

# a design in Rhapsody DM, and to a relevant wikipedia article.

:bike

 dcterms:references <http://doors.example.com/bikeModule/bikeRequirements>,

 <http://dm.example.com/dm/bikeDesign>,

 <http://en.wikipedia.org/wiki/Bicycle>.

# In this example, we introduce different versions of the bike parts.

# We use TriG rather than Turtle, with a separate graph for each resource

# in the TRS/index.

# For simplicity of exposition, the following example elides the View,

# and pretends that an Item owns its constituent parts directly.

# Note that all the resources and triples in these examples are created

# and updated by RELM, except for the triples in *blue italic* which may be

# contributed by VVC, given suitable version-to-point mappings.

@prefix vvc: <http://jazz.net/ns/vvc#>.

# Define the initial concept resources: bike, wheel, tire, hubcap

# by defining a first version of each.

:bike-1 = {

 :bike-1

 dcterms:isVersionOf :bike.

 :bike

 a pd:Item;

 dcterms:hasPart [a :wheel], [a :wheel].

}

:wheel-15inch = {

 :wheel-15inch

 dcterms:isVersionOf :wheel.

 :wheel

 a pd:Item;

 :wheelSize 15;

 dcterms:hasPart [a :tire], [a :hubcap].

}

:tire-15inch = {

 :tire-15inch

 dcterms:isVersionOf :tire.

 :tire

 a pd:Item;

 :wheelSize 15 .

}

:hubcap-silver = {

 :hubcap-silver

 dcterms:isVersionOf :hubcap.

 :hubcap

 a pd:Item;

 :color :silver.

}

# Construct the first bicycle configuration that selects

# these initial versions. Note that RELM may create the configuration,

# but VVC decides the vvc:selects triples using the configuration function.

# In bike configuration 5\_15\_silver:

# the timeOrder is 5, the wheels are 15" and the hubcap is silver.

:bikeConfig\_5\_15\_silver = {

 :bikeConfig\_5\_15\_silver

 a vvc:Configuration;

 vvc:point [vvc:timeOrder 5; :wheelSize 15; :color :silver];

 *vvc:selects*

 *vvc:mapping1,*

 *vvc:mapping2,*

 *vvc:mapping3,*

 *vvc:mapping4.*

}

# Still continuing with the same example,

# we introduce a 17" wheel and tire, and a red hubcap.

:wheel-17inch = {

 :wheel-17inch

 dcterms:isVersionOf :wheel.

 :wheel

 a pd:Item;

 :wheelSize 17;

 dcterms:hasPart [a :tire];

 dcterms:hasPart [a :hubcap].

}

:tire-17inch = {

 :tire-17inch

 dcterms:isVersionOf :tire.

 :tire

 a pd:Item;

 :wheelSize 17 .

}

:hubcap-red = {

 :hubcap-red

 dcterms:isVersionOf :hubcap.

 :hubcap

 a pd:Item;

 :color :red.

}

# In bike configuration 9\_17\_red:

# the wheels are 17", the hubcap is red.

# Note that the bike itself did not need to change.

:bikeConfig\_9\_17\_red = {

 :bikeConfig\_9\_17\_red

 a vvc:Configuration;

 vvc:point [vvc:timeOrder 9; :wheelSize 17; :color :red];

 *vvc:selects*

 *vvc:mapping1,*

 *vvc:mapping6,*

 *vvc:mapping7,*

 *vvc:mapping8.*

}

# In the following example, we build a penny-farthing, which is

# a bike with a large front wheel and a small back wheel.

# For completeness, we redefine everything from scratch, and include the views.

@prefix dcterms: <http://purl.org/dc/terms/>.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

@prefix bp: <http://open-services.net/ns/basicProfile#>.

@prefix vvc: <http://jazz.net/ns/vvc#>.

@prefix pd: <http://jazz.net/ns/pd#>.

@prefix : <http://example.org/toybike/>.

pd:Item rdfs:subClassOf rdfs:Class.

pd:View rdfs:subClassOf bp:Container.

:wheel-15inch = {

 :wheel-15inch dcterms:isVersionOf :wheel.

 :wheel

 a pd:Item;

 :wheelSize 15 .

}

:viewOfWheel-15inch-v1 = {

 :viewOfWheel-15inch-v1 dcterms:isVersionOf :viewOfWheel-15inch.

 :viewOfWheel-15inch

 a pd:View;

 bp:membershipPredicate dcterms:hasPart;

 pd:isViewOf :wheel-15inch;

 dcterms:hasPart [a :tire], [a :hubcap].

}

:tire-15inch= {

 :tire-15inch dcterms:isVersionOf :tire.

 :tire

 a pd:Item;

 :wheelSize 15 .

}

:wheel-50inch = {

 :wheel-50inch :isVersionOf :wheel.

 :wheel

 a pd:Item;

 :wheelSize 50 .

}

:viewOfWheel-50inch-v1 = {

 :viewOfWheel-50inch-v1 dcterms:isVersionOf :viewOfWheel-50inch.

 :viewOfWheel-50inch

 a pd:View;

 bp:membershipPredicate dcterms:hasPart;

 pd:isViewOf :wheel-50inch;

 dcterms:hasPart [a :tire], [a :hubcap].

}

:tire-50inch = {

 :tire-50inch dcterms:isVersionOf :tire.

 :tire

 a pd:Item;

 :wheelSize 50 .

}

:hubcap-silver = {

 :hubcap-silver dcterms:isVersionOf :hubcap.

 :hubcap

 a pd:Item;

 :color :silver.

}

# And build the penny-farthing. This time, since the wheels are different,

# we drop the use of pd:occurrences and use two different sub-configurations.

:bike-pF = {

 :bike-pF dcterms:isVersionOf :bike.

 :bike a pd:Item.

}

:viewOfBike-pF-v1 = {

 :viewOfBike-pF-v1 dcterms:isVersionOf :viewOfBike.

 :viewOfBike-pF

 a pd:View;

 bp:membershipPredicate dcterms:hasPart;

 pd:isViewOf :bike;

 dcterms:hasPart [

 a :wheel; vvc:foreignConfiguration vvc:largeWheelConfig];

 dcterms:hasPart [

 a :wheel; vvc:foreignConfiguration vvc:smallWheelConfig].

}

# Finally, show the penny-farthing configuration

# and its two different sub-configurations

:bikeConfigPF = {

 :bikeConfigPF

 a vvc:Configuration;

 vvc:point [vvc:timeOrder :now; :color :silver];

 vvc:selects

 vvc:mapping110,

 vvc:mapping111.

}

vvc:largeWheelConfig = {

 vvc:largeWheelConfig

 a vvc:Configuration;

 vvc:point [vvc:timeOrder :now; :wheelSize 50; :color :silver];

 *vvc:selects*

 *vvc:mapping102,*

 *vvc:mapping103,*

 *vvc:mapping104,*

 *vvc:mapping106.*

}

vvc:smallWheelConfig = {

 vvc:smallWheelConfig

 a vvc:Configuration;

 vvc:point [vvc:timeOrder :now; :wheelSize 15; :color :silver];

 *vvc:selects*

 *vvc:mapping100,*

 *vvc:mapping101,*

 *vvc:mapping104,*

 *vvc:mapping105.*

}