TM Forum Specification

REST API Design Guidelines Part 7

JSON Schema patterns

TMF630

Team Approved Date: 08-May-2020

Release Status: Production	**Status: TM Forum Approved**
Version 4.0.1	**IPR Mode: RAND**

Notice

Direct inquiries to the TM Forum office:

4 Century Drive, Suite 100
Parsippany, NJ 07054, USA
Tel No. +1 973 944 5100
Fax No. +1 973 998 7916
TM Forum Web Page: www.tmforum.org

Table of Contents

Document objective

This document describes patterns to be applied for JSON schema definition. Our objective is to build consistent schema in order to have consistent APIs more easily useable and understandable for the TMF community.

JSON schemas are defined for all API classes. TMF tooling will build API swagger & Specification from these schemas (and YAML file).

It is important to consider that TMF schema JSON built are based on JSON draft 7.

A complete 'dummy' API example is provided with this document. This dummy API 'Schema Ordering' is featuring all schema patterns.

Reader notice:

The reader should take note that

- When we use the notation **<<Entity>>**, it represents a place holder to insert the name of the class described in this schema. If we have to use following **Related<<Entity>>** schema pattern for Trouble Ticket then the schema's name will be **RelatedTroubleTicket.**
- All provided examples are illustrative and not systematically reflect existing TMF schema.

Schema structure definition

A set of attributes MUST be present in all schemas and CONSISTENTLY valued:

Schema attribute	
`"$schema": "http://json-schema.org/draft-07/schema#",`	MUST always valued like this – no <<Entity>> dependent.
`"$id":"http://datamodel.tmforum.org/`**`<<Entity>>`**`.schema.json",`	MUST be same than schema name file MUST use <<Entity>>.
`"$version":"4.0.0",`	3-digits version: major/minor/patch As starting point to align swagger with schema proposal it to begin with 4.0.0
`"title": "`**`<<Entity>>`**`",`	MUST be same than schema name file MUST use <<Entity>>.
`"definitions": {`	
`"`**`<<Entity>>`**`": {`	Only one entity MUST be described in one JSON schema file.

Schema attribute	
	MUST use <<Entity>>.
"$id": "#<<**Entity**>>",	Entity identifier – This is used to refer this Entity from other. MUST be valued with <<Entity>>
"description": "bla bla …",	Description of the <<Entity>> - must be a definition of the entity. **This is copied in the Specification**
"type": "object",	
"required": ["attribute1", "attribute2"],	Optionally mandatory attribute could be specified
"properties": {	distinct cases are described below:
When the property is defined in this schema (basic schema type)	
"<<**entityAttributeName**>>": {	Attribute name
"type": "string",	The type could be string, , integer, boolean, number see array below
"description":	Description of the attribute - must be a definition of the attribute. **This is copied in the Specification**
"format":	Optionally the format could be provided – example "format": "date-time" for a date (the type is defined as a string)
"examples" : ["red", "blue"]	Optionally attribute value example(s) could be provided.
When the property is referred in another schema	
"<<**entityAttributeName**>>": {	Attribute name
"description": "",	Specific description for this attribute in this schema Note: The swagger editor will complaint about this description (via warning) – The description is coming for the schema description referred.
"$ref": "../<<**ReferredEntityDomain**>>/<<**ReferredEntityFileName**>>.schema.json#<<**ReferredEntityIdentifier**>>"	<<ReferredEntityDomain>> is the repository name where the schema is stored in the Schema git hub repo. <<ReferredEntityFileName>> is the referred file name – Must be the referred entity name <<ReferredEntityIdentifer>>: identifier of the referred entity in the file.

Schema attribute	
When the property is defined as an array	
"<<**entityAttributeName**>>": {	Attribute name
"type": "array",	
"items": {	
"$ref": "../<<**ReferredEntityDomain**>>/<<**ReferredEntityFileName**>>.schema.json#<<**ReferredEntityIdentifier**	
}	
"minItems": 1	Optionally minimum occurrence of item could be provided

Addressable schema

Purpose

This schema is used for any entity that could be addressable by an id and a href.

Definition

Addressable schema features only 2 attributes

- id – unique identifier of a resource
- href – hyperlink (uri) to access the resource

Rules

- Addressable is not used directly in <<Entity>> schema but used as part of Entity or EntityRef schemas.
- This schema is already defined in Common repo and MUST not be changed in any API.

Extensible schema

Purpose

This schema is used for any entity that could leverage polymorphism pattern to be extended

Definition

Extensible schema features attributes

- @type - When sub-classing, this defines the sub-class Extensible name
- @baseType - When sub-classing, this defines the super-class
- @schemaLocation - A URI to a JSON-Schema file that defines additional attributes and relationships

Rules

- Extensible is not used directly in <<Entity>> schema but used as part of Entity or EntityRef schemas.
- This schema is already defined in Common repo and MUST not be changed in any API.

Entity schema

Purpose

Entity schema MUST be used to any schema describing a resource addressable with an id and a href and extensible though TMF polymorphic pattern (example: ProductOrder, TroubleTicket, Party, etc…)

Definition

Entity schema is a merge of Addressable + Extensible schemas

Rule

- This schema is already defined and presents in common repo. It MUST not be changed in any API.
- Entity schema must be used for <<Entity>> schema via allOf (see <<Entity>> part)

EntityRef schemas

Purpose

EntityRef schema MUST be used to any schema describing a reference to an entity (example: ProductOrderRef, TroubleTicketRef, PartyRef, etc…)

Definition

EntityRef schema is a merge of Addressable + Extensible schemas plus @referredType and name attribute.

Rule

- This schema is already defined and presents in common repo. It MUST not be changed in any API.
- EntityRef schema must be used for <<Entity>>Ref schema via allOf (see <<Entity>>Ref part)

<<Entity>> schema

Purpose

The <<Entity>> schema is used to describe all API resource representation. In most case an <<Entity>> is (or will be) managed as a resource is one of the TMF API.

Examples: TroubleTicket, QueryServiceQualification, CustomerBill, Quote, etc…

Rules

- MUST include a reference to **Entity** subschema (use of allOf keyword in Swagger 2.0).
- MUST contain specific attributes (‘basic datatype like string, Boolean, etc…or subresources) used to describe the <<Entity>> class.
- If the resource need a type attribute the attribute name MUST not be type but <<entity>>Type.
- The resource may require a ‘timestamp’ date to manage date time when resource is created – in this case the attribute should be named <<entity>>Date.

Example

ShoppingCart schema

```
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "$id": " http://datamodel.tmforum.org/ShoppingCart.schema.json",
   "title": "ShoppingCart",
   "definitions": {
      "ShoppingCart": {
         "$id": "#ShoppingCart",
         "description": "Shopping Cart resource is used for the temporarily selection and reservation of product offerings in e-commerce, call center and retail purchase. Shopping cart supports purchase of both physical and digital goods and service (e.g. handset, telecom network service). Shopping Cart contain list of cart items, a reference to customer (partyRole) or contact medium in case customer not exist, and the total items price including promotions",
         "type": "object",
         "properties": {
            "validFor": {
               "$ref": "../Common/TimePeriod.schema.json#TimePeriod",
               "description": "The period for which the shopping cart is valid (e.g. 90 if no activity or 7 days if cart is empty)"
            },
            "contactMedium": {
               "type": "array",
               "items": {
                  "$ref": "../Common/ContactMedium.schema.json#ContactMedium"
               }
            },
            "cartTotalPrice": {
               "type": "array",
               "items": {
                  "$ref": "CartPrice.schema.json#CartPrice"
               },
               "description": "Total amount of the shopping cart, usually of money, that represents the actual price paid by the Customer for cart (considering only \"Active\" cart items)"
            },
            "cartItem": {
               "type": "array",
```

```
            "items": {
              "$ref": "CartItem.schema.json#CartItem"
            }
          },
          "relatedParty": {
            "type": "array",
            "items": {
              "$ref": "../EngagedParty/RelatedParty.schema.json#RelatedParty"
            }
          }
        },
        "allOf": [
          {
            "$ref": "../Common/Entity.schema.json#Entity"
          }
        ]
      }
    }
}
```

<<Entity>>Ref schema

Purpose

<<Entity>>Ref schema is used to describe a reference to an existing entity. This referred entity must be accessible via GET operation from an existing (or future) TMF API.

Example: In the Product schema for example the following entities are referred: Billing Account, Product Offering, Product specification, Service, etc…. this allow to – if required – refer these entities from a product instance.

Rules

- MUST include a reference to **EntityRef** schema (use of allOf keyword in Swagger 2.0).
- MAY include additional attributes from the <<Entity>> that are considered important – illustration. Example: ResourceRef schema features a 'value' attribute. This represents the human-readable value associated with the resource, it might be any of:
    - MSISDN
    - Serial number
    - IMSI
    - Etc
- The associated <<Entity>> schema MUST contain the id attribute

Examples

ServiceSpecificationRef schema – please note presence of additional version attribute:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": " http://datamodel.tmforum.org/ServiceSpecificationRef.schema.json",
  "title": "ServiceSpecificationRef",
  "definitions": {
```

```
    "ServiceSpecificationRef": {
      "$id": "#ServiceSpecificationRef",
      "description": "Service specification reference: ServiceSpecification(s) required to realize a ProductSpecification.",
      "type": "object",
      "properties": {
        "version": {
          "type": "string",
          "description": "Service specification version"
        },
      }
      "allOf": [
        {
          "$ref": "../Common/EntityRef.schema.json#EntityRef"
        }
      ]
    }
  }
}
```

<<Entity>>ItemRef schema

Purpose

<<Entity>>Item Ref schema is used to describe a reference to an existing item of an existing entity. This schema should be used in cases where referred item is multiple depths below the managed entity. In this case, we need to provide entity id (and href) plus item identifier.

It is recommended to use this schema to refer a distinct entity type (Product resource target a service resource using this schema).

When same entity type should be linked we recommend using <<EntityRelationship>> pattern (but it enforces a relationship type use).

Example: in resource inventory, we target an array of {#resource order + #resource order item} that created/modified this resource instance.

Rules

- MUST include a reference to **Entity** schema (use of allOf keyword in Swagger 2.0)
- Additional attributes MUST be added in the schema:
    - <<entity>>Id
    - <<entity>>Href
    - itemId
    - @referredType
- Additional attributes could be added (optional) – like name.
- It should be noted than @referredType is relevant for the targeted entity owning the targeted item and not this item itself.

Examples

ContainedItemRef schema is defined to describe any <<Entity>>Item reference:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://datamodel.tmforum.org/AgreementItemRef.schema.json",
  "title": "AgreementItemRef",
  "definitions": {
    "AgreementItemRef": {
      "type": "object",
      "description": "Agreement reference. ….",
      "properties": {
        "@referredType": {
          "type": "string",
          "description": "The actual type of the target instance when needed for disambiguation."
        },
        "agreementHref": {
          "type": "string",
           "format": "uri",
          "description": "Reference of the related entity."
        },
        "agreementId": {
          "type": "string",
          "description": "Unique identifier of a related entity."
        },
        "name": {
          "type": "string",
          "description": "Name of the related entity."
        },
        "itemId": {
          "type": "string",
          "description": "Identifier of the agreement"
        }
      },
      "allOf": [
        {
          "$ref": "../Common/Entity.schema.json#/Entity"
        }
      ]
    }
  }
}
```

Related<<Entity>> schema

Purpose

The Related<<Entity>>Schema is used when it is necessary to refer another resource type from a resource representation **with a role**. This is not used to describe relation between same resource type. The most common example is relatedParty – in a lot of API like all order API, trouble ticket, quote, etc… we link related party(ies) to these resources but also indicate the role played by the party within the resource.

Rules:

- MUST include a reference to **EntityRef** schema (use of allOf keyword in Swagger 2.0). It is important to consider that the information provided in the @type, @baseType, @schemaLocation and @referredType are relevant to the targeted entity and not for the related<<Entity>> schema itself.
- The role attribute MUST be mandatory

- **Exception**: In RelatedParty schema, the role is not mandatory because provided id could refer to a PartyRole id – in this case the Role could be blank. Please note that for RelatedParty the @referredType MUST be mandatory

- **Related<<Entity>>Ref** schema MUST not exist!

Examples

RelatedPlace – Not as of today in our repository but must be fixed to be like:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "RelatedPlace.schema.json",
  "title": "RelatedPlace",
  "definitions": {
    "RelatedPlace": {
      "$id": "#RelatedPlace",
      "description": "A  Place and an associated role as installation address, delivery address, etc....",
      "type": "object",
      "required" : [
        "role"
              ],
      "properties": {
        "role": {
          "type": "string",
          "description": "Role of the place, such as: [home delivery], [shop retrieval])"
        },
      },
      "allOf": [
        {
          "$ref": "EntityRef.schema.json#EntityRef"
        }
      ]
    }
  }
}
```

RelatedParty:

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "$id": "RelatedParty.schema.json",
 "title": "RelatedParty",
 "definitions": {
    "RelatedParty": {
      "$id": "#RelatedParty",
      "description": "Related Entity reference. A related party defines party or party role linked to a specific entity.",
      "type": "object",
       "required" : [
        "id",
        "@referredType"
              ],
      "properties": {
        "role": {
          "type": "string",
          "description": "Role played by the related party"
        },
          "allOf": [
       {
          "$ref": "../Common/EntityRef.schema.json#EntityRef"
       }
```

```
      ]
    }
  }
}
```

Related<<Entity>>Item schema

Purpose

Globally this is the same concept than Related<<entity>> but to an item from another resource and not to a complete resource. The targeted resource (owning the item) should be different than the source resource (if same, relationship pattern must be used). For example, in product inventory we describe a list of related item of product order used to modify this product. This should be used when we want to be precise at item level.

Rules:

- A Related<<Entity>>Item **MUST** include a reference to **Entity** subschema (use of allOf keyword in Swagger 2.0).
- Additional attributes MUST be added in the schema :
    - itemId (mandatory)
    - role (mandatory)
    - <<entity>>Id (mandatory)
    - <<entity>>Href
    - @referredType
- Additional attributes could be added (optional) – like name.

Example

RelatedProductOrderItem schema - Not as of today in repository but must be fixed to be like:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "RelatedProductOrderItem.schema.json",
  "title": "RelatedProductOrderItem",
  "definitions": {
    "RelatedProductOrderItem": {
      "type": "object",
      "description": "RelatedProductOrderItem (ProductOrder item) .The product order item which triggered product
creation/change/termination.",
      "required": ["itemId","productOrderId","role"],
      "properties": {
        "@referredType": {
          "type": "string",
          "description": "The actual type of the target instance when needed for disambiguation."
        },
        "productOrderHref": {
          "type": "string",
          "format": "uri",
          "description": "Reference of the related entity."
        },
        "productOrderId": {
```

```
            "type": "string",
            "description": "Unique identifier of a related entity."
          },
          "orderItemAction": {
            "type": "string",
            "description": "Action of the order item for this product"
          },
          "itemId": {
            "type": "string",
            "description": "Identifier of the order item where the product was managed"
          },
          "role": {
            "type": "string",
            "description": "role of the product order item for this product"
          }
        },
        "allOf": [
          {
            "$ref": "Entity.schema.json#/Entity"
          }
        ]
      }
    }
}
```

<<Entity>>RefOrValue

Purpose

This schema is used to describe an existing id via its reference (by ref) **or** to describe literally an existing/notExisting Entity (by value). It should be considered that OrValue feature MUST be added only when value pattern is relevant in **providing** information **via POST** (and/or PATCH) operation. This pattern MUST not be used only for retrieving information in GET response (where EXPAND & DEPTH keywords could be leveraged). Good example is provided in productOrder API where product could provide by ref (to change existing product) or by value (a new one, to provide product intend).

Rule

- MUST be built from <<Entity>> schema + <<Entity>>Ref schema (use of allOf keyword in Swagger 2.0).
- isRef attribute MUST NOT be present in the schema.
- Attributes @type, @schemaLocation & @referredType **are related to the contained entity** and not the <<Entity>>RefOrValue itself.

Example

ProductRefOrValue:

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "$id": "ProductRefOrValue.schema.json",
```

```
 "title": "ProductRefOrValue",
 "definitions": {
    "ProductRefOrValue": {
       "$id": "#ProductRefOrValue",
       "description": "A product to be created defined by value or existing defined by reference. The polymorphic attributes @type,
@schemaLocation & @referredType are related to the product entity and not the RelatedProductRefOrValue class itself",
       "type": "object",
       "properties": {
       },
       "allOf": [
          {
            "$ref": "ProductRef.schema.json#ProductRef"
          },
          {
            "$ref": "Product.schema.json#Product"
          }

       ]
    }
 }
}
```

Instantiation examples:

For Ref:

```
"place" : {
   "@type" : "GeographicAddressRef",
   "@referredType" : "GeographicAddress",
   "id" : "17",
   "href": "www...../geographicAdress/17",
   "name" : "Name of the referred address 17"
}
```

For Value:

```
"place" : {
   "@type" : "GeographicAddress"
   "@schemaLocation" : "url where the geographic site schema is described...not the placeRefOrValue schema"
   "name": "Jacob House"
   "GeographicAddress": {
       "streetNr": 4,
       "streetNane":  "Boisseau",
       "streetType":  "rue",
...
   }
}
```

Related<<Entity>>RefOrValue

Purpose

This schema has same purpose than <<Entity>>RefOrValue where entity targeted could be only provided by reference or literally but with the additional requirement to have an associated role provided.

A good example is RelatedPlaceRefOrValue in a service qualification request where the place to perform the qualification could be an address reference or a valued geographic location for example and additionally a role must be specified to define place role as installation site, covering zone, etc.

Rules

- MUST be built from <<Entity>> schema + <<Entity>>Ref schema (use of allOf keyword in Swagger 2.0)
- MUST have a role attribute (mandatory)
- isRef attribute MUST NOT be present in the schema.
- Attributes @type, @schemaLocation & @referredType **are related to the contained entity** and not the Related<<Entity>>RefOrValue class itself.

Example

Example for RelatedPlaceRefOrValue

```
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "$id": "RelatedPlaceRefOrValue.schema.json",
   "title": "RelatedPlaceRefOrValue",
   "definitions": {
      "RelatedPlaceRefOrValue": {
         "$id": "#RelatedPlaceRefOrValue",
         "description": "Related Entity reference. A related place defines a place described by reference or by value linked to a specific entity. The polymorphic attributes @type, @schemaLocation & @referredType are related to the place entity and not the RelatedPlaceRefOrValue class itself",
         "type": "object",
         "properties": {
          "role":{
            "type": "string"
          }
         },
         "required": [
          "role"
         ],
         "allOf": [
           {
              "$ref": "PlaceRef.schema.json#PlaceRef"
           },
           {
              "$ref": "Place.schema.json#Place"
           }

         ]
      }
   }
}
```

Instantiation examples:

For Ref:

```
"relatedPlace" : {
```

```
   "role" : "shipping address for the cheese",
   "@type" : "GeographicAddressRef",
   "@referredType" : "GeographicAddress",
   "id" : "17",
   "href": "www...../geographicAdress/17",
   "name" : "Jacob House"
}
```

For Value:

```
"relatedPlace" : {
   "role" : "shipping address for the cheese",
   "@type" : "GeographicAddress"
   "@schemaLocation" : "url where the geographic site schema is described...not the placeRefOrValue schema"
   "name": "Jacob House"
   "GeographicAddress": {
      "streetNr": 4,
      "streetNane":  "Boisseau",
      "streetType":  "rue",
...
   }
}
```

<<Entity>>Relationship

Purpose

This schema MUST be used to describe relationship between **same** base entity type for example to describe relationship between Trouble Ticket. Additionally, this schema is also used within one resource instance to describe relationship between sub elements – for example relationship between order items within an order instance. This relationship is always typed.

Rules

- <<Entity>> must be built from **Entity** schema (allOf).
- Add the @referredType attribute
- Additional attributes could be added in the schema :
    - relationshipType (mandatory)
    - <<Entity>>RelationshipCharacteristic (using Characteristic.schema.json)
    - Targeted entity could be either described with (one – and only one - mandatory)
        - id and href attribute, or
        - a relation to a <<Entity>>Ref, or
        - a relation to a <<Entity>>ItemRef.

Examples

Example for ResourceRelationship:

```
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "$id": "ResourceRelationship.schema.json",
```

```
  "title": "ResourceRelationship",
  "definitions": {
    "ResourceRelationship": {
      "type": "object",
      "properties": {
        "href": {
          "type": "string",
           "format": "uri",
          "description": "href of the related resource."
        },
        "relationshipType": {
          "type": "string"
        },
        "id": {
          "type": "string",
          "description": "id of the related resource."
        },
        "resourceRelationshipCharacteristic": {
          "type": "array",
          "items": {
            "$ref": "Characteristic.schema.json#/definitions/Characteristic"
          }
        }
      },
      "allOf": [
        {
          "$ref": "../Common/Entity.schema.json#Entity"
        }
      ],
      "required": [
        "id", "relationshipType"
      ]
    }
  }
}
```

Example of use of this schema for distinct resource type (a logical resource specification targeting a physical resource specification) but same base type (resource specification):

```
{
            "@type": "LogicalResourceSpecification",
            "resourceSpecRelationship": [
                {
                "id": "42",
                "href": "/resourceCatalog/v4/resourceSpecification/42",
                "@referredType": "PhysicalResourceSpecification”,
                "relationshipType": "dependency"
                }
            ]
}
```

Characteristic

Purpose

This schema MUST be used when a resource need to manage characteristic. It allows defining relationship characteristic. The ‘value’ attribute is defined with an ‘any’ value type to indicate that depending on characteristic it could be a string, a Boolean, an integer, an object, etc…

Rules

- This schema is already defined and present in TMF schema tooling and MUST not be changed for any API.

TMF Forum will provide a recommendation to manage the 'any' data type in implementation.

Examples

Characteristic schema:

```
{
 "$schema": "http://json-schema.org/draft-07/schema#",
 "$id": "Characteristic.schema.json",
 "title": "Characteristic",
 "definitions": {
  "Characteristic": {
   "$id": "#Characteristic",
   "description": "Describes a given characteristic of an object or entity through a name/value pair.",
   "type": "object",
   "required": [
    "name",
    "value"
   ],
   "properties": {
    "id": {
     "type": "string",
     "description": "Unique identifier of the characteristic"
    },
    "name": {
     "type": "string",
     "description": "Name of the characteristic"
    },
    "valueType": {
     "type": "string",
     "description": "Data type of the value of the characteristic"
    },
    "value": {
     "$ref": "../Common/Any.schema.json#Any",
     "description": "The value of the characteristic"
    },
    "characteristicRelationship": {
     "type": "array",
     "items": {
      "$ref": "../Common/CharacteristicRelationship.schema.json#CharacteristicRelationship"
     }
    }
   },
   "allOf": [
    {
     "$ref": "../Common/Extensible.schema.json#Extensible"
    }
   ]
  }
 }
}
```

Instantiation examples

Example for a string characteristic:


```
{
  "productCharacteristic": [
    {
      "id": "1gh-5gg",
      "@type": "Characteristic",
      "name": "TEL_MSISDN",
      "valueType": "string",
      "value": "415 279 7439"
    }
  ]
}
```


Example for an object:


```
{
  "productCharacteristic": [
    {
      "id": "zzd-722",
      "@type": "Characteristic",
      "name": "ElectricBikeConfiguration",
      "valueType": "object",
      "value": {
        "@type": "ElectricBikeConfig",
        "@schemaLocation": "http://serverlocation:port/electicBikeConfig.json",
        "Color": "red",
        "Motor": "CX Gen 4",
        "size": "large",
        "wheel size": 27.5
      }
    }
  ]
}
```


Characteristic Specification

Purpose

This set of schemas MUST be used when an entity specification defines characteristics for instantiation in an inventory. For example, product specification defines characteristics that will be populated when a product is instantiated in the product inventory. The set of schemas includes:

- **CharacteristicSpecificationBase** – abstract base class for all characteristic specifications; contains all the common attributes relevant for a characteristic definition
- **CharacteristicSpecification** – concrete characteristic specification for Entity Specification. Does allOf for the base class together with the list of values and list of characteristic relationships.
- **<xxxx>CharacteristicSpecification** – concrete characteristic specification for entity specification <xxxx>, e.g. ProductSpecificationCharacteristic. Does allOf for the base class, together with parts of the definition that are specific to the entity.

Note: These <xxxx> concrete classes exist to preserve backward compatibility of the existing attribute names (e.g. productCharacteristicValue). Any new implementation of a characteristic specification should use the generic CharacteristicSpecification

- These schemas are already defined and present in TMF schema tooling and MUST not be changed for any API.

Examples

ProductSpecificationCharacteristic schema:


```
{
   "$schema": "http://json-schema.org/draft-07/schema#",
   "$id": "ProductSpecificationCharacteristic.schema.json",
   "title": "ProductSpecificationCharacteristic",
   "definitions": {
      "ProductSpecificationCharacteristic": {
         "$id": "#ProductSpecificationCharacteristic",
         "description": "A characteristic quality or distinctive feature of a ProductSpecification.  The characteristic can be take on a discrete value, such as color, can take on a range of values, (for example, sensitivity of 100-240 mV), or can be derived from a formula (for example, usage time (hrs) = 30 - talk time *3). Certain characteristics, such as color, may be configured during the ordering or some other process.",
         "type": "object",
         "properties": {
            "productSpecCharacteristicValue": {
               "type": "array",
               "items": {
                  "$ref": "../Common/CharacteristicValueSpecification.schema.json#CharacteristicValueSpecification"
               },
               "description": "A ProductSpecificationCharacteristicValue object is used to define a set of attributes, each of which can be assigned to a corresponding set of attributes in a ProductSpecificationCharacteristic object. The values of the attributes in the ProductSpecificationCharacteristicValue object describe the values of the attributes that a corresponding ProductSpecificationCharacteristic object can take on."
            },
            "productSpecCharRelationship": {
               "type": "array",
               "items": {
                  "$ref": "ProductSpecificationCharacteristicRelationship.schema.json#ProductSpecificationCharacteristicRelationship"
               },
               "description": "An aggregation, migration, substitution, dependency or exclusivity relationship between/among Specification Characteristics."
            }
         },
         "allOf": [
            {
               "$ref": "../Common/CharacteristicSpecificationBase.schema.json#CharacteristicSpecificationBase"
            }
         ]
      }
   }
}
```


Example json payload of ProductSpecificationCharacteristic

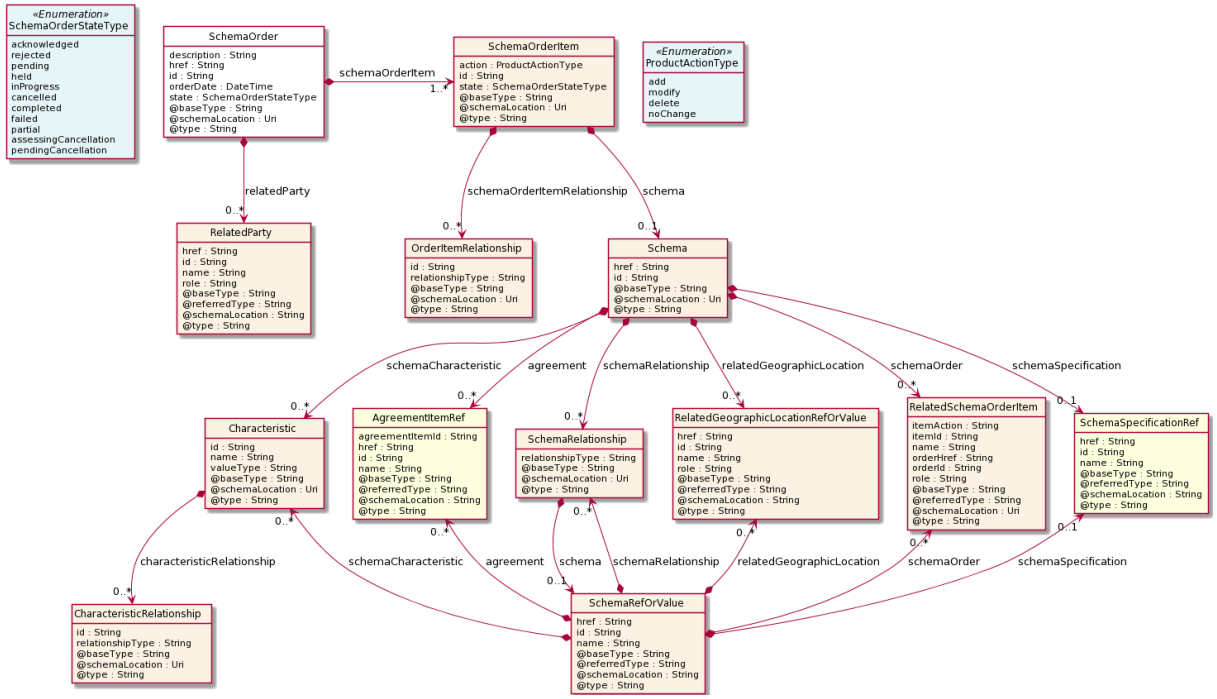

```
"productSpecCharacteristic": [
   {
      "name": "Number of Ports",
      "description": "The total Number of Ports for this product",
      "valueType": "number",
      "configurable": true,
```

```
      "minCardinality": 1,
      "maxCardinality": 1,
      "isUnique": true,
      "productSpecCharRelationship": [
        {
          "id": "43",
          "href": "https://mycsp.com:8080/tmf-api/productCatalogManagement/v4/productSpecification/43",
          "relationshipType": "Dependency",
          "name": "Bandwidth",
          "validFor": {
            "startDateTime": "2017-04-19T16:42:23-04:00"
          }
        }
      ],
      "productSpecCharacteristicValue": [
        {
          "isDefault": true,
          "validFor": {
            "startDateTime": "2017-06-16T00:00",
            "endDateTime": "2018-01-13T00:00"
          },
          "@type": " ProductSpecCharacteristicValue ",
          "valueType": "number",
          "value": 8
        },
        {
          "isDefault": false,
          "validFor": {
            "startDateTime": "2017-06-16T00:00",
            "endDateTime": "2018-01-13T00:00"
          },
          "@type": "ProductSpecCharacteristicValue ",
          "valueType": "number",
          "value": 16
        },
        {
          "isDefault": false,
          "validFor": {
            "startDateTime": "2017-06-16T00:00",
            "endDateTime": "2018-01-13T00:00"
          },
          "@type": "ProductSpecCharacteristicValue ",
          "valueType": "number",
          "value": 24
        }
      ],
      "validFor": {"startDateTime": "2017-04-19T16:42:23.0Z"}
    },
    {
      "name": "Color",
      "description": "Color of the Firewall housing",
      "valueType": "string",
      "configurable": true,
      "minCardinality": 1,
      "maxCardinality": 1,
      "extensible": true,
      "isUnique": true,
      "productSpecCharacteristicValue": [
        {
          "isDefault": true,
          "@type": "ProductSpecCharacteristicValue ",
          "valueType": "string",
          "value": "Black"
        },
        {
          "isDefault": false,
```

```
            "@type": "ProductSpecCharacteristicValue ",
            "valueType": "string",
            "value": "White"
          }
        ],
        "validFor": {
          "startDateTime": "2017-04-19T16:42:23.0Z"
        }
      }
  ]
```

TMFZZZ API Schema demo:

This diagram illustrates the resource model of the TMF ZZZ API. This API is used to describe all schema patterns:



The API swagger is available under TMFZZZ repository.

Administrative Appendix

Document History

Version History

Version Number	Date Modified	Modified by:	Description of changes
4.0.0	14-May-2020	Alan Pope	Minor edits prior to publication
4.0.1	20-Jul-2020	Adrienne Walcott	Updated to reflect TM Forum Approved Status

Release History

This section records the changes between this and the previous Official document release. The release number is the 'Marketing' number which this version of the document is first being assigned to.

Release Number	Date Modified	Modified by:	Description of changes
Pre-production	14-May-2020	Alan Pope	Updated to v4.0
Production	20-Jul-2020	Adrienne Walcott	Updated to reflect TM Forum Approved Status

Acknowledgments

This document was prepared by the members of the TM Forum team:

- Pierre Gauthier, TM Forum, Editor and Team Leader
- Ludovic Robert, Orange, Author