**Catalogue Service for API Consumers**

*Catalogue Service* lets NIMBLE users to make their products or services discoverable on NIMBLE. The term ***item*** is used to collectively refer to the products or services regardless of their nature. Items of a company are collected inside a ***catalogue.*** Driven by the data standard that is used in the scope of this service the term ***party*** is used to refer any company regardless of their business. So, within the documentations, the aforementioned terms might be used interchangeably.

**Data Model**

Although there are some extensions, we have adopted the Universal Business Language (UBL)[[1]](#footnote-1) standard to represent catalogues. In general, the following data types keep item information.

*CatalogueType -> CatalogueLineType -> GoodsItemType -> ItemType*

Up-to-date XSD representation of the data model can be used at: <https://github.com/nimble-platform/common/tree/staging/data-model/ubl-data-model/src/main/schema/NIMBLE-UBL-2.1>. Considering the way UBL uses to group data elements, while *CatalogueType* is defined in the *maindoc/UBL-Catalogue-2.1.xsd* file, the other data elements are defined in *maindoc/UBL-CommonAggregateComponents-2.1.xsd* file.

**CatalogueType**

****

* The ***id*** field is party-specific identifier for the catalogue and it must unique among the *ids* of other catalogues belonging to the party. The field must be provided while creating a new catalogue. A party might have multiple catalogues. Considering the front-end capabilities, as of writing this documentation, catalogues with ***id*** field set to ***default*** are displayed.
* The ***uuid*** field is a unique id generated by the system for each catalogue.
* Although not available in the data model XSDs explicitly, each complex data type included in the UBL data model, including the data types mentioned above, has an ***hjid*** field. This is an identifier generated by the database layer once the corresponding entity is persisted in the database. It is also unique among the entities persisted in the same database instance.
* ***issueDate*** is not considered, the field is kept as it is mandatory in the UBL standard.
* ***providerParty*** is the party owning the catalogue
* ***catalogueLine*** list contains catalogue lines describing items

**Constraints on CatalogueTypes**

* Each catalogue must have a filled ***id*** field i.e.
	+ *catalogue.id* must exist
* Each catalogue must have provider party
	+ catalogue.providerParty must exist

**CatalogueLineType**

****

* The ***id*** field of a catalogue line has the same reasoning with the one in the catalogue objects. It must be unique among the line ids considering the lines included in a specific catalogue.
* ***minimumOrderQuantity*** is a quantity indicating the minimum quantity for this specific item e.g. 5 metres
* ***freeOfChargeIndicator*** is a boolean value indicating the availability of a free sample for the associated item
* ***warrantyValidityPeriod*** is a period object that can be used to specify warranty information for the associated item. Currently, the ***durationMeasure*** field included in the period object is being used to indicate a quantity for the warranty validity period e.g. 2 years.
* The ***price*** object included in the ***requiredItemLocationQuantity*** field is used to specify the initial price for the associated item. Alternative prices in the form discounts of charges can be specified via the ***priceOption*** list.
* ***goodsItem*** is a data type containing information about the item characteristics via the ***item*** field as well as trading information including packaging information, delivery terms

**PriceOptionType**

****

Price options are used to specify alternative pricing on the initial price in the form of discount or charges. Multiple pricing options can be defined on a single item. All the pricing options are aggregated, and the final price is calculated with the aggregated value. For example, assuming that there is a 5% discount and a 3% charge defined for the item, a 2% discount is applied on the initial price.

* ***name*** can be used to specify a name for distinguishing the option
* ***itemLocationQuantity*** field is used to indicate the details of the discount or charge. The pricing option can be defined per unit basis or for complete order. For per unit case, itemLocationQuantity.allowanceCharge[0].perUnitAmount for complete order case itemLocationQuantity.allowanceCharge[0].amount fields are set.
* To specify a pricing option on the ordered amount itemLocationQuantity.minimumAmount is set. The specified values indicate the minimum order amounts so that the pricing option would be applied e.g. setting this value to 10 unit indicates that at least 10 units should be ordered.
* To specify a pricing option on the delivery location itemLocationQuantity.applicableTerritoryAddress list is filled. Multiple addresses can be specified.
* ***incoterms*** list can be used to specify specific pricing options various incoterms
* ***estimatedDeliveryPeriod*** field is used to specific pricing options based on the delivery period of the item. Specified values indicate the latest period so that the pricing option would be applied e.g. setting this value to 1 week means that the product would be delivered in 1 week at latest where the specified pricing option is applied.
* ***additionalItemProperty*** list contains specific values of product properties that are subject to varying pricing options. The item properties for the price option should be consistent with the item properties that are defined for the item in item.additionalItemProperty .
* ***paymentMeans*** field can be used to define price options based on varying payment means
* ***specialTerms*** field can be used to define price options based on unstructured conditions

**GoodsItemType**

****

Goods item entities contain information about items considering the item’s distribution in the supply chain i.e. package type and delivery terms.

* ***item*** is an entity containing information about the item’s characteristics i.e. master data.
* ***containingPackage*** is an entity where the quantity included in one package can be specified via containingPackage.quantity and packaging type can be specified via containingPackage.packagingTypeCode
* ***deliveryTerms*** can be used to specify *incoterms, delivery period* and other *special terms* concerning the delivery of the associated item via the deliveryTerms.incoterms, deliveryTerms.estimatedDeliveryPeriod and deliveryTerms.specialTerms fields.

**ItemType**

****

Item entities are used to specify characteristics of a product/service.

* ***name, description*** and ***productImage*** fields are self-explanatory
* ***manufactersItemIdentification*** is used to identify the item inside a particular catalogue.
* ***catalogueDocumentReference*** is used to give a reference to the catalogue in which the item is listed.
* ***itemSpecificationDocumentReference*** can be used to specify additional external documents for elaborating the item.
* ***commodityClassification*** list is a list of product categories. The categories are referred via the commodityClassification.itemClassificationCode field. Each category is a concept defined in an external taxonomy e.g. eClass or Furniture ontology.
	+ commodityClassification.itemClassificationCode.value is the identifier of the referred concept in the target taxonomy e.g. “0173-1#01-AKJ050#013”
	+ commodityClassification.itemClassificationCode.name is the name of the concept as defined in the target taxonomy e.g. “MDF raw”
	+ commodityClassification.itemClassificationCode.uri is the dereferencable uri of the concept returning the description of the concept as in the target taxonomy e.g. “http://www.nimble-project.org/resource/eclass/22292801”.
	+ commodityClassification.itemClassificationCode.listID is the identifier of the target taxonomy as managed in NIMBLE e.g. “eClass” or “FurnitureOntology”
* ***additionalItemProperty*** list is a list of item features. Additional item properties can either be driven by the product categories specified in the ***commodityClassification*** list or they can be custom properties.
	+ additionalItemProperty.id is the identifier of the property. It is unique among the identifiers of the all properties included in the ***additionalItemProperty*** list*.* For external-taxonomy driven item properties this field should be set as the identifier of the corresponding property (See ***ProductCategoryService*** below to get details of a product category and associated properties)
	+ additionalItemProperty.name is the name of the item property.
	+ additionalItemProperty.valueQualifier specifies the data type of the item property which can be TEXT, NUMBER, QUANTITY, BOOLEAN, FILE and IMAGE.
	+ According to the value qualifier of the property the value is set via an appropriate field
		- For TEXT and BOOLEAN cases, the ***value*** field should be used
		- For NUMBER cases, the ***valueDecimal*** field should be used
		- For QUANTITY case, the ***valueQuantity*** field should be used
		- For FILE and IMAGE cases, the ***valueBinary*** field should be used
	+ ***uri*** keeps the uri of the corresponding property referring to the target taxonomy if the property is driven from a taxonomy.
	+ ***itemClassificationCode*** is a reference to the corresponding ***commodityClassification*** where this property is driven from, specifically to the ***commodityClassification.itemClassificationCode***.
* ***manufacturerParty*** is the same party providing the catalogue
* ***certificate*** list keeps the certificates specified for this item
* ***dimension*** list keeps the dimensions of the item
* ***transportServiceDetails*** keeps the details specific to the transportation if the item is a transport service
* ***trackAndTraceDetails*** keeps the track and trace details of the item where each field in this entity
	+ ***eventUrl*** is a url returning the event information regarding the production of the item
	+ ***masterUrl*** is a url returning master data for this item as required by the track and trace service
	+ productionProcessTemplate is a url returning the production process template this item as required by the track and trace service

**Constraints on CatalogueLineTypes**

* Each ***CatalogueLineType*** must have a filled ***id*** field. This is a unique-identifier for the item considering all the items included in a “**specific catalogue**”. This value should be provided by the catalogue provider during the publishing.
	+ *catalogueLine.id* must exist
* Each item must have a ***ManufacturerParty*** with a filled ***id*** field i.e.
	+ catalogueLine.goodsItem.item.manufacturerParty.id must exist
* The value provided in the ***id*** field of a ***CatalogueLineType*** should be duplicated in the as the manufacturer item identifier in the corresponding ***ItemType*** i.e.
	+ catalogueLine.id must be equal to
	catalogueLine.goodsItem.item.manufacturersItemIdentification.id
* The identifier of the party providing the catalogue and the identifier of the manufacturer party of items must be equal for each item i.e.
	+ *catalogue.providerParty.id* must be equal to
	*catalogueLine.goodsItemType.itemType.manufacturerParty.id*
* Each ***ItemType*** must have a name
	+ catalogueLine.goodsItem.item.name must exist
* Items must have at least one commodity classification.
	+ catalogueLine.goodsItem.item.commodityClassification list must have at least one ***CommodityClassificationType***. See the
* Items must have a reference to the owning catalogue
	+ catalogueLine.goodsItem.item.catalogueDocumentReference.id must be equal to catalogue.uuid

**Product Categories**

NIMBLE benefits from external taxonomies and ontologies to offer standard-based product categories and associated features. In the UBL data model, while product categories are represented with ***CommodityClassificationType*** data types, the features associated to a category are represented with ***ItemPropertyType*** data types. Categories and features are associated to item objects via ***commodityClassification*** and ***additionalItemProperty*** fields.

An example category is “***MDF, melamine-resin faced****”* from the *eClass* taxonomy. This category, in turn, has properties associated to it including:

* Decorative Surface of Upper Side
* Decorative Surface of The Lower Side
* Surface Structure Pressed Top
* Surface Structure Pressed Bottom
* Layer Design of The Coating Top
* Layer Design of The Coating Bottom

Considering the example above, assuming “***MDF, melamine-resin faced****”* category is selected for annotating a product, a ***CommodityClassificationType*** instance would be created for the item inside the ***commodityClassification*** as follows:

 {

 "itemClassificationCode": {

 "value": "0173-1#01-AKJ051#013",

 "name": "MDF, melamine-resin faced",

 "uri": "http://www.nimble-project.org/resource/eclass/22292802",

 "listID": "eClass"

 }

Furthermore, if one of the properties driven by this category was populated for the product e.g. *“****Decorative Surface of Upper Side”***, an ***ItemPropertyType*** instance would be created in the ***additionalItemProperty*** array as follows:

{

 "id": "0173-1#02-BAF074#004",

 "name": "Decorative surface of upper side",

 "value": [

 "Melamine",

 "Veneer"

 ],

 "valueQualifier": "STRING",

 "itemClassificationCode": {

 "value": "0173-1#01-AKJ051#013",

 "name": "MDF, melamine-resin faced",

 "uri": "http://www.nimble-project.org/resource/eclass/22292802",

 "listID": "eClass"

 },

 "uri": "http://www.nimble-project.org/resource/eclass/property/BAF074004"

}

While retrieving categories from external taxonomies a simple data model is used. In this context, a ***Category[[2]](#footnote-2)*** entity represents a category, e.g. the melamine-resin faced category, retrieved from the taxonomy. Each property associated a category is represented with ***Property[[3]](#footnote-3)*** data type.

**REST API**

The service offers CRUD operations on the ***CatalogueType*** and ***CatalogueLineType*** entities. The service also provides specific operations to manage price options for catalogue lines. In addition to this REST services are available for import and export of catalogues, retrieving product categories, managing units and managing binary content data as described below.

The latest released version of the REST API of the catalogue service can also be found at: <http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-controller>

**General Remarks on REST API**

Although the reference data model, i.e. UBL, is an XML-based data model, REST services currently support JSON based content negotiation. In this respect, the UBL-driven Java Classes included in <https://github.com/nimble-platform/common/tree/staging/data-model/ubl-data-model/src/main/java/eu/nimble/service/model/ubl> can be used in combination with a ***Jackson ObjectMapper*** to get serialization of an object in JSON format. A NIMBLE-compliant serializer could be obtained with JsonSerializationUtility.getObjectMapper() method[[4]](#footnote-4).

As mentioned above, database identifiers (kept in ***hjid*** fields of entities) are created for any persisted entity. In ***update*** operations, i.e. ***PUT*** operations, the serialized form of the relevant entity should include *hjid* value that were generated after the first persistence of the entity.

Below, the REST API components are summarized along with the services included in them:

**Catalogue API (**[**http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#/catalogue-controller**](http://localhost:8095/swagger-ui.html#/catalogue-controller)**)**

Catalogue API provides simple CRUD functionalities for C***atalogueType*** entities. There are two ways of publishing items: one-by-one by providing item details on the NIMBLE UI or downloading an Excel-based template to provide details for multiple items at once. In both cases, the first step is to select one or more categories to annotate the item.

* Catalogue publishing: <http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-controller/addXMLCatalogueUsingPOST>
* Catalogue retrieval: <http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-controller/getCatalogueUsingGET>
* Catalogue update: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-controller/updateCatalogueUsingPUT](http://localhost:8095/swagger-ui.html#!/catalogue-controller/updateCatalogueUsingPUT)
* Catalogue deletion: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-controller/deleteCatalogueUsingDELETE](http://localhost:8095/swagger-ui.html#!/catalogue-controller/deleteCatalogueUsingDELETE)
* Downloading template: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-controller/downloadTemplateUsingGET](http://localhost:8095/swagger-ui.html#!/catalogue-controller/downloadTemplateUsingGET)
* Uploading template: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-controller/uploadTemplateUsingPOST](http://localhost:8095/swagger-ui.html#!/catalogue-controller/uploadTemplateUsingPOST)
* Uploading images for a catalogue: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-controller/uploadImagesUsingPOST](http://localhost:8095/swagger-ui.html#!/catalogue-controller/uploadImagesUsingPOST)

**Catalogue Line API (**[**http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#/catalogue-line-controller**](http://localhost:8095/swagger-ui.html#/catalogue-line-controller)**)**

Catalogue line API provides CRUD operations for catalogue lines. To be able to benefit from these operations, a catalogue should have been published already.

* Adding a catalogue line to a catalogue: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-line-controller/addCatalogueLineUsingPOST](http://localhost:8095/swagger-ui.html#!/catalogue-line-controller/addCatalogueLineUsingPOST)
* Updating a catalogue line: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-line-controller/updateCatalogueLineUsingPUT](http://localhost:8095/swagger-ui.html#!/catalogue-line-controller/updateCatalogueLineUsingPUT)
* Getting a catalogue line: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-line-controller/getCatalogueLineUsingGET](http://localhost:8095/swagger-ui.html#!/catalogue-line-controller/getCatalogueLineUsingGET)
* Deleting a catalogue line: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/catalogue-line-controller/deleteCatalogueLineUsingDELETE](http://localhost:8095/swagger-ui.html#!/catalogue-line-controller/deleteCatalogueLineUsingDELETE)

**Price Configuration Controller API (**[**http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#/price-configuration-controller**](http://localhost:8095/swagger-ui.html#/price-configuration-controller)**)**

This API provides functionalities create new prices options items.

* Adding a price option to a catalogue line: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/price-configuration-controller/addPricingOptionUsingPOST](http://localhost:8095/swagger-ui.html#!/price-configuration-controller/addPricingOptionUsingPOST)
* Updating a price option: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/price-configuration-controller/updatePricingOptionUsingPUT](http://localhost:8095/swagger-ui.html#!/price-configuration-controller/updatePricingOptionUsingPUT)
* Deleting a price option: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/price-configuration-controller/deletePricingOptionUsingDELETE](http://localhost:8095/swagger-ui.html#!/price-configuration-controller/deletePricingOptionUsingDELETE)

**Product Category API (**[**http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#/product-category-controller**](http://localhost:8095/swagger-ui.html#/product-category-controller)**)**

Category API provides services to retrieve categories as they are defined in external taxonomies.

* Getting specific categories: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/product-category-controller/getSpecificCategoriesUsingGET](http://localhost:8095/swagger-ui.html#!/product-category-controller/getSpecificCategoriesUsingGET)
* Getting categories by name: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/product-category-controller/getCategoriesByNameUsingGET](http://localhost:8095/swagger-ui.html#!/product-category-controller/getCategoriesByNameUsingGET)
* Getting available taxonomy ids: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/product-category-controller/getAvailableTaxonomyIdsUsingGET](http://localhost:8095/swagger-ui.html#!/product-category-controller/getAvailableTaxonomyIdsUsingGET)
* Getting root categories for a taxonomy: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/product-category-controller/getRootCategoriesUsingGET](http://localhost:8095/swagger-ui.html#!/product-category-controller/getRootCategoriesUsingGET)
* Getting child categories for a specified category: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/product-category-controller/getChildrenCategoriesUsingGET](http://localhost:8095/swagger-ui.html#!/product-category-controller/getChildrenCategoriesUsingGET)
* Getting parents categories for a specified category along with siblings of each parent and the category itself: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/product-category-controller/getCategoryTreeUsingGET](http://localhost:8095/swagger-ui.html#!/product-category-controller/getCategoryTreeUsingGET)

**Unit API (**[**http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#/unit-service-controller**](http://localhost:8095/swagger-ui.html#/unit-service-controller)**)**

Unit API provides services to manage units. NIMBLE maintains unit lists like *volume\_quantity* or *weight\_quantity* including a set of units. The examples contain respectively *L, m3* and *g, kg, ton* units.

* Getting all unit lists with the included values: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/unit-service-controller/getAllUnitListsUsingGET](http://localhost:8095/swagger-ui.html#!/unit-service-controller/getAllUnitListsUsingGET)
* Creating a unit list: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/unit-service-controller/addUnitListUsingPOST](http://localhost:8095/swagger-ui.html#!/unit-service-controller/addUnitListUsingPOST)
* Getting units inside a unit list: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/unit-service-controller/getValuesUsingGET](http://localhost:8095/swagger-ui.html#!/unit-service-controller/getValuesUsingGET)
* Adding unit to a unit list: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/unit-service-controller/addUnitToListUsingPATCH](http://localhost:8095/swagger-ui.html#!/unit-service-controller/addUnitToListUsingPATCH)
* Deleting a unit from a unit list: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/unit-service-controller/deleteUnitFromListUsingDELETE](http://localhost:8095/swagger-ui.html#!/unit-service-controller/deleteUnitFromListUsingDELETE)

**Binary Content API (**[**http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#/binary-content-controller**](http://localhost:8095/swagger-ui.html#/binary-content-controller)**)**

Binary content API provides retrieval services for the binary content managed in the binary content database.

* Getting binary content inside a ***BinaryObjectType***: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/binary-content-controller/getBinaryContentUsingGET](http://localhost:8095/swagger-ui.html#!/binary-content-controller/getBinaryContentUsingGET)
* Getting binary content in base64 format: [http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#!/binary-content-controller/getBase64BinaryContentUsingGET](http://localhost:8095/swagger-ui.html#!/binary-content-controller/getBase64BinaryContentUsingGET)

**Import Export Controller API (**[**http://nimble-staging.salzburgresearch.at/catalog/swagger-ui.html#/import-export-controller**](http://localhost:8095/swagger-ui.html#/import-export-controller)**)**

This API currently provides import service so that a catalogue retrieved from a particular NIMBLE instance can be imported into another instances.

**Catalogue Service for Platform Providers**

1. http://docs.oasis-open.org/ubl/UBL-2.1.html [↑](#footnote-ref-1)
2. https://github.com/nimble-platform/catalog-service/blob/master/catalogue-service/src/main/java/eu/nimble/service/catalogue/model/category/Category.java [↑](#footnote-ref-2)
3. https://github.com/nimble-platform/catalog-service/blob/master/catalogue-service/src/main/java/eu/nimble/service/catalogue/model/category/Property.java [↑](#footnote-ref-3)
4. https://github.com/nimble-platform/common/blob/master/utility/src/main/java/eu/nimble/utility/JsonSerializationUtility.java [↑](#footnote-ref-4)