Rainbow OF – Rainbow order fulfilment

Active document used for design ideas. Will be merged into a single document for documentation.

# Main screen:

Contains Details on the Right hand side and a collapsible menu on the left

## Main Menu:

* Dashboard?
  + Order Status
  + Feed Status
  + Preparation
  + Packing
* Contacts
  + All
  + By Type
  + Preferences
    - Equipment owned?
* Items
  + Items
    - Active
  + Item Groups
  + Categories
  + Attributes
    - Varieties
  + Price Lists?
  + Preferences
    - Item Types
      * Should this be simple / Variety? What about equipment?
    - Packaging
    - Varieties
* Orders
  + New
  + View
    - Active
    - By Day
    - All, Today, Tomorrow, This Week, Next Week
    - By Status
      * Ready, Invoiced, Done, Pending
    - By Type?
    - Packing
  + Recurring
  + Returns
  + Preferences
* Notifications
  + Send notifications
  + Preferences
    - Email template, order, ship, done, pack, repair
* 3rd Party (xRepairs)
  + Active
  + By Status
  + Preferences
    - Swop out equipment
* Integration
  + Import
    - From CSV
    - From Woo
    - View Log
  + Export
  + Live integration
    - Woo settings
  + External forms? Requests?
* Reports
  + Preparation Report
  + Packing Report
* System
  + Users/Parties
    - All
    - By type (Driver, Courier, admin, user)
  + Tools
* Settings
  + Days and dates (weekdays/closure dates)
  + Company Details
  + Preferences
    - Regional Settings
    - TAX / VAT
    - Email settings
    - Import Export
  + Shipping Zones / Areas?
  + Accounting
    - Price levels
    - Invoice Types

# Woo Integration Support

## WooSync Table

Stores the status of the synchronisation of Woo. Initially use for the Import but also fired by the WooTriggers that we load. Used by the WooSyncRepository

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| WooSyncLogId | Int | PK |
| WooSyncDateTime | DateTime | The Date and time the sync was done |
| Section | Enum | Which section: Coupons, Customers, Order, OrderRefunds, Product Variant, Product VariantAttributes, Product VariantAttributeTerms, Product VariantCategories, Product VariantTags, WebHooks, Taxes or System |
| SectionID | Int | Id if queried otherwise “0”. |
| Result | Enum | Success, Error, Timeout |
| Parameters | String | What was sent |
| Notes | String | Any notes |

## Woo Rest Repositories

We use WooCommerce.NET nuGet package. The Git repo is here: <https://github.com/XiaoFaye/WooCommerce.NET>

Each section of the Rest Api has its own calls, and any child items reside within the parents class. We have a Interface for each REST API section we called defined interfaces are:

* IWooBase
* IWooProduct
* IWooProductAttribue
* IWooProductAttributeTerm
* IWooProductCategory
* IWooProductVariation

Defined Classes that use these are

* WooBase
* WooProduct
* WooProductAttribute
* WooProductAttributeTerm
* WooProductCategory
* WooProductVariation

WooBase is used by each to set the basic platform, the others will have CRUD type calls. These could have been Generic with WooEntity as the Generic. However, these were created early on and the advantage of creating Generics had I did not fully understand yet. There is an inconsistency with the implementation of id’s in the WooCommerce.net Api where some id’s are uint and so are ints. Where possible units are used.

There are two types of Rest API calls ones that use the root Rest like url/<WooEntity> and ones that returns children or variants url/<WooEntity>/<id>/<WooEntityVariant>. The Interfaces that are root based do not need an id to work, the others need the root id to work as reflected in the interface definitions. Some of the root base Woo Entities are returned in pages (like Categories, Product, Customers, orders. This would mean there would need three generic Interfaces (besides the Base).

* IWooRootEntity
* IWooRootEnityPaged or Large (may be able to live without this, if we just implement the Get/Retrieve as paged).
* IWooVariantEnity

We can look at this for new interfaces, or convert the current ones.

All Interfaces use:

* WooAPISettings
* ILoggerManager

## Woo Sync Log

All interactions with Woo should be logged. They will be logged in the WooSyncLog Table which will look like this.

## Woo Web Hooks

After the migration has happened, we need to ass web hooks, or modify them if they exist.

Hooks that need to be created:

* CRUD – Product Variants, and Product Variant settings like attributes and categories
* CRUD – orders and supporting tables
* CRUD – users/customers

# Woo Integration Import and Lookups

Used for importing:

* items or products,
* item or product variants,
* product categories,
* product attributes and
* possible attribute terms to the system.
* customers
* orders

This is done using the WooSettings Table.

Options:

* API settings (URL/IS HTTPS/QueryPrefixes/CustomerKey/CustomerSecret
* Import of:
  + Categories (**true**/false)
  + Attributes (**true**/false)
  + Map Varieties (**true**/false)
  + Only in stock (**true**/false)
  + Include item quantities (true/**false**)

## Import Interface for Woo

The Woo Import implements the following per imported section. This interface design can be used as a base for other imports, but it has been made specific to Woo due to the settings record and allows for any unique methods and variables to be included.

IWooImportBase

IAppUnitOfWork \_AppUnitOfWork

ILoggerManager \_Logger

WooSettings \_AppWooSettings

ImportCounters CurrImportCounters

Task<Guid> GetWooMappedEntityById(int sourceWooEntityId);

IWooImport: IWooImportBase

Task<List<TWooEntity>> GetWooEntityData();

Task<Guid> AddWooEntity(TWooEntity newEntity);

Task<Guid> AddOrGetEntityID(TWooEntity sourceEntity);

Task<Guid> AddOrUpdateEntity(TWooEntity sourceEntity, Guid sourceWooMappedEntityId);

Task<Guid> ImportAndMapWooEntityData(TWooEntity sourceEntity);

Task<Guid> UpdateWooEntity(TWooEntity updatedPC, TWooMapEntity targetWooMap);

IWooImportWithAParent : IWooImportBase

Task<List<TWooEntity>> GetWooEntityData(unit wooSourceId);

Task<Guid> AddWooEntity(TWooEntity newEntity, Guid sourceParentAttributeId);

Task<Guid> AddOrGetEntityID(TWooEntity sourceEntity, Guid sourceParentAttributeId);

Task<Guid> AddOrUpdateEntity(TWooEntity sourceEntity, Guid sourceWooMappedEntityId,  
Guid sourceParentAttributeId);

Task<bool> FindAndSetParentEntity(WooItemWithParent sourceWooEntityWithParent);

Task<Guid> ImportAndMapWooEntityData(TWooEntity sourceEntity, Guid sourceParentAttributeId);

Task<bool> SetWooEntityParent(Guid sourceChildWooEntityId, Guid sourceParentWooEntityId);

Task<Guid> UpdateWooEntity(TWooEntity updatedPC, TWooMapEntity targetWooMap, Guid sourceParentAttributeId);

IWooImportWithParents : IWooImportBase

Task<List<TWooEntity>> GetWooEntityData(bool? onlyInStock = true);

Task<Guid> AddWooEntity(TWooEntity newEntity, List<WooItemWithParent> newWooEntityWithParents);

Task<Guid> AddOrGetEntityID(TWooEntity sourceEntity, List<WooItemWithParent> sourceWooEntityWithParents);

Task<Guid> AddOrUpdateEntity(TWooEntity sourceEntity, Guid sourceWooMappedEntityId,   
List<WooItemWithParent> sourceWooEntityWithParents);

Task<bool> FindAndSetParentEntity(WooItemWithParent sourceWooEntityWithParent);

Task<Guid> ImportAndMapWooEntityData(TWooEntity sourceEntity,   
List<WooItemWithParent> sourceWooEntityWithParents);

Task<bool> SetWooEntityParent(Guid sourceChildWooEntityId, Guid sourceParentWooEntityId);

Task<Guid> UpdateWooEntity(TWooEntity updatedPC, TWooMapEntity targetWooMap,   
List<WooItemWithParent> sourceWooEntityWithParents);

## Categories

First need to import categories that will map to our category lookups. This Import uses the Interface IWooImportWithParents to be implemented, since Categories can have Parents.

Rest URL used:

[https://www.quaffee.com/wp-json/wc/v3/products/categories](https://www.quaffee.com/wp-json/wc/v3/products/categories?parent%3e0)

From there we can create a list of possible item categories. These are used in the items Mapping. Only those that are imported can be used…..

#### What Woo Gives Us:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| id | integer | Unique identifier for the resource. (read-only) |
| name | String | Category name. (mandatory) |
| slug | String | An alphanumeric identifier for the resource unique to its type. |
| parent | integer | The ID for the parent of the resource. |
| description | String | HTML description of the resource. |
| display | String | Category archive display type. Options: default, products, subcategories and both. Defaul1t is default. |
| image | object | Image data. See [Product category - Image properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-category-image-properties) |
| menu\_order | integer | Menu order, used to custom sort the resource. |
| count | integer | Number of published products for the resource. (read-only) |

#### Our Lookup class

|  |  |  |
| --- | --- | --- |
| Field name | Type | Notes |
| ItemCategoryLookupId | Guid |  |
| CategoryName | string(255) | item category |
| ParentCategoryId | Guid? | [foreignkey("parentcategoryid ")] |
| UsedForPrediction | Bool | Is this category used for prediction – will need to be user editted |
| UoMBaseId | Guid? | Links to BaseUoM on ItemUoMLookup – will need to be user editted |
| Notes | String |  |
| Rowversion | byte[] | [timestamp] |

#### Mapping:

The Table WooCategoryMaps stores the category information.

|  |  |  |
| --- | --- | --- |
| Field | Type | Used For |
| WooCategoryMapId | Int | Id field |
| WooCategoryID | Int | To store the CategoryID that Woo returns |
| ~~WooCategoryName~~ | string (255) | To store Categories.name that Woo returns |
| ~~WooCategorySlug~~ | string (255) | To store Categories.slug that Woo returns |
| WooCategoryParentID | Int | Can be null if not will point to a WooCategoryID |
| ItemCategoryLookupID | Guid | This links to the Category Lookup in the system. Allows us to transfer the data. |
| CanUpdate [bool] | BOOL | Set true by default – false if we do not want the Product to be updated |

Item categories can be linked to tracking / predictability. So, items in the same category are tracked similarly. The Item in a category is associated to other items and all of the items have a common denominator set via the Bases Unit of Measure of the category. These settings must be changed by the user later. Perhaps after the Import a wizard should guide them through.

Parent id is stored. ParentId=0 means no parent (like with woo)

Logic:

* Retrieve all the woo Categories
* If the IDs exists in the mapping update if category can be updated (if missing from the item table add)
* otherwise, if a category of the same name exists get that ID and update
  + otherwise add the category
  + if the ParentId > 0 add this category id to a list of categories to scan later to check if the parent has been imported.

Once the Import is done then loop through the list of attributes that had parents and see if they exists if so set the id otherwise set parent id to Guid.empty

##### Note about Prediction and Unit of Measure

Predictability and Unit of measure are associated to a category and an item is associated to a category. This has to work this way since items that are part of the same category can then be predicted together based on both the unit of measure and category that links them to other items. For Item specific prediction a category will need to be created specific to that prediction. Can an item be part of two predictive categories? We cannot think of one, but I am sure there is one, we do not block this.

### Display of Category Lookups

Categories are displayed using a DataGrid. The DataGrid will allow you to edit the category and all data related to it. Like all the other Data Grids sorting and filtering of critical fields is supported.

#### Default Sort:

Sorted by Parent Name(s)+CategoryName

#### Columns:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Column Name | Displays | E | F | S | Format | Validation |
| Category Name | CategoryName prefixed with “—” per parent | Y | Y | Y |  |  |
| Parent Category | If there is a parent category otherwise n/a | Y | Y | Y |  |  |
| Prediction | Is this category used for prediction | Y | Y | Y | Combo |  |
| Base UoM | If there is a base Unit of Measure | Y | Y | Y | Combo |  |
| Woo Linked | Is it linked to Woo, can Woo update this? And must we update it in Woo | Y | ? | ? | Y/N |  |
| Notes | Any notes about the category | Y | Y | Y |  |  |

**Legend***: E= Editable; F= Filterable; S = Sortable*

### Editing

Editing will occur via the standard DataGrid modal popup menu. With all values besides the current available in the dropdown list linked to the current parent.

If modified, then Woo item is updated only if woo items are affected like Category Name and Parent Category, and it is allowed to be updated.

## Attributes/ Variations

Attributes are used but Woo to create Variations of a products, and to give ectra information of the Product. A product can have normal attribute or attributes are used to create variations.

Variable Terms are product Attribute Terms that are used as variations in a product. If a product item is of type “variable” then it has attribute terms that make up the varieties. There are groups of varieties, like colour (Green, Red, White, Blue) and size (S,M,L) or packaging (box, bottle, packet bag) and preparation (beans, ground espresso, ground filter).

Uses IWooImport to import the attributes.

To retrieve attributes, REST call:

<https://www.quaffee.com/wp-json/wc/v3/products/attributes>

This returns the attribute names not the types. We still need to link the attribute to the attribute terms or types. This means parent -> child tables

Parent Table is ItemAttributes, Child is ItemAttributeVarieties. Each Item will also need a table linking the ItemAttributes that Item has to the Item

Note for future reference: For the user entering in a line item in the order / invoice, just enters in the SKU or can search on Item name (auto complete).

#### What Woo Gives Us:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| Id | integer | Unique identifier for the resource. (read-only) |
| Name | string | Attribute name. (mandatory) |
| Slug | string | An alphanumeric identifier for the resource unique to its type. |
| Type | string | Type of attribute. By default, only select is supported. |
| order\_by | string | Default sort order. Options: menu\_order, name, name\_num and id. Default is menu\_order. |
| has\_archives | boolean | Enable/Disable attribute archives. Default is false. |

### Our class Item Attribute Lookup

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeLookupID | Int | Pk |
| AttributeName | String(100) | Required, indexed |
| OrderBy | Enum ? (custom, name, num and id) | Should this be an enum? |
| Notes | String |  |

#### Mapping:

The Table WooVariationMaps stores the category mapping information.

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeLookupId | GUID | PK (ItemAttributeId+WooProductAttribID) |
| WooProductAttributeId | Int |
| CanUpdate | Bool | Set true by default – false if we do not want the Product to be updated |

All id fields that are to be converted to GUID – except for preferences

### Attribute Terms (or variations)

Here is a list of the terms that an attribute has or the list of labels/names an attribute can have. For example. Packaging can be an attribute which has Terms: 250, 275, 500 , 750, 200 etc.

This is a list that can allocated to a product that has that attribute. A product can have any number of these terms. For our purposes we will call the variety. Since we mainly use this to determine the variety of the Product selected.

Uses IWooImportWithAParent for the attribute variations, since there is a single parent per Attribute term or variant.

These have to also be retrieved. You request a list of an attribute’s terms using the Woo Attribute’s Id. Retrieved using Rest:

/wp-json/wc/v3/products/attributes/<attribute\_id>/terms

#### What Woo Gives Us:

Woo gives us this per attribute:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| id | integer | Unique identifier for the resource. |
| name | string | Term name. (mandatory) |
| slug | string | An alphanumeric identifier for the resource unique to its type. |
| description | string | HTML description of the resource. |
| menu\_order | integer | Menu order, used to custom sort the resource. |
| count | integer | Number of published products for the resource. (read-only) |

##### Item Attribute Variety

We store this data is a table ItemAttributeVariety. Based on the Woo Attribute Varieties or what they call terms, but added to that the fields for displaying

|  |  |  |
| --- | --- | --- |
| Field name | Type | Comments |
| ItemAttributeVarietyID | GUID | Pk |
| ItemAttributeID | GUID | Parent or null |
| VarietyName | String(100) | Required, indexed |
| UoMID? | GUID | Optional (set by user) linked to UoM |
| UoMQtyPerItem | Double | If UOM is applicable what is the factor to the base |
| DefaultSKUSuffix | String(10) | Default Suffix for variant's SKU |
| SortOrder | Int | Number for order |
| Symbol | String(2) | For display |
| FGColour | String(11) | For display |
| BGColour | String(11) | For display |
| Notes | String |  |

#### Woo Link

Use these Table to link, and if the WooSettings is set to Import Attributes

##### WooProductAttributeMapping

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeID | Int | Pk with WooProductAttribID |
| WooProductAttributeID | Int | Links to woo attribute |
| CanUpdate | Bool | Set true by default – false if we do not want the Product to be updated |

##### WooProductAttributeTermMapping

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeVarietyID | Int | Pk with WooProductAttribTermID |
| WooProductAttributeTermID | Int | Links to woo attribute |
| CanUpdate | Bool | Set true by default – false if we do not want the Product to be updated |

To view list of attributes terms

https://www.quaffee.com/wp-json/wc/v3/products/attributes/14/terms

These can map to variations.

For each Product we allow people to select the “parent” and then the child, making it easier and supporting more. Then we need to do variation report linking to UOM or each.

<https://woocommerce.github.io/woocommerce-rest-api-docs>. We use the WooRestAPI NuGet Package - WooCommerceNET by JamesYang@NZ.

Example screens:

Product

# 

Item Variant

# 

# Items

Items are what is offered to the client, what can be ordered and delivered. Will use the items from WooCommerce as a guide. Basic breakdown in the images:

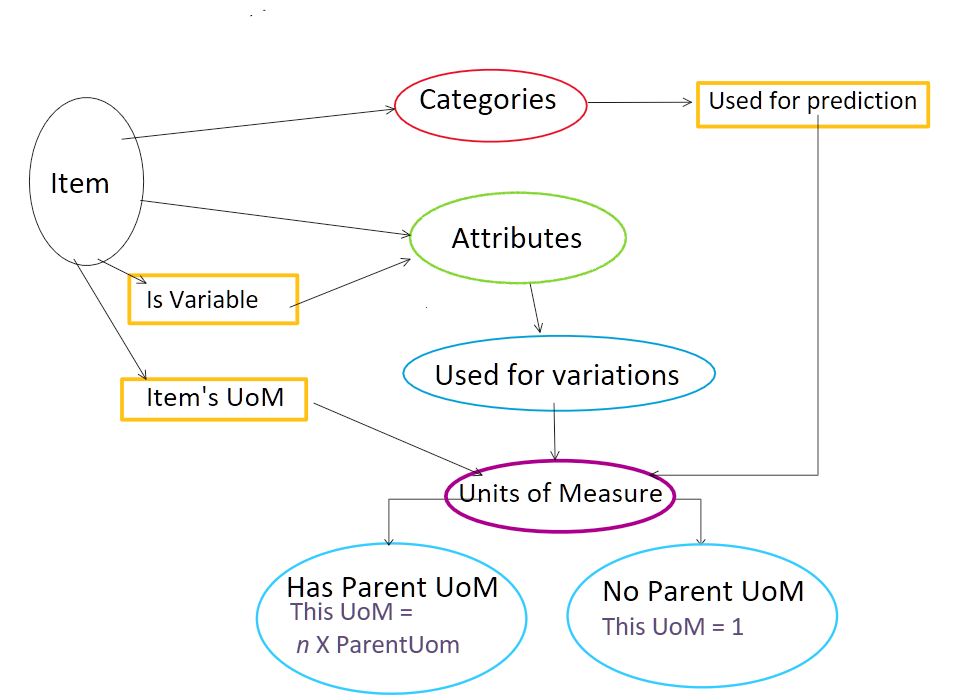


Figure 1: Items with Variations

Besides fields that are listed in the Items Table below this is a methodology on how the Item (or Product in Woo) works.

An Item can be of types as below, but the figure is how the variables (and predictability works, if required.)

## Item Category

Each Item has a at least one category. Only one of which can be predictable (cannot current identify if you would need more than one). When the Item is ordered that is part of a predictable category that is, that order (once fulfilled) will be used to determine when a contact needs a notification about the possibility that they will be running low. When a predictive category is used, along with Unit Of Measure (UoM), the system can see how many of the Item of type category x where consumed per date period. As the number of date periods between orders of the items of this type increase the prediction can be moved to the last X orders. Prediction will be covered later in the document, but this is how the data is stored.

## Item Attributes

As can be seen from the image Attributes can be used to determine the variations of the Item, should an item is of type variable. Some attributes can also just be used for information about a product. However, when and attribute marked as used for variation, then an item has many child items. Item variations are mapped the prediction and also the Item they are associated with.

A Woo Item Rest call returns any variations that where created. Each of these is a new Product Variation, that returns only limited data specific to the variation. Each variation can also be retrieved.

Item Variations should be synced during the major import and also per Item

### Example of implementation

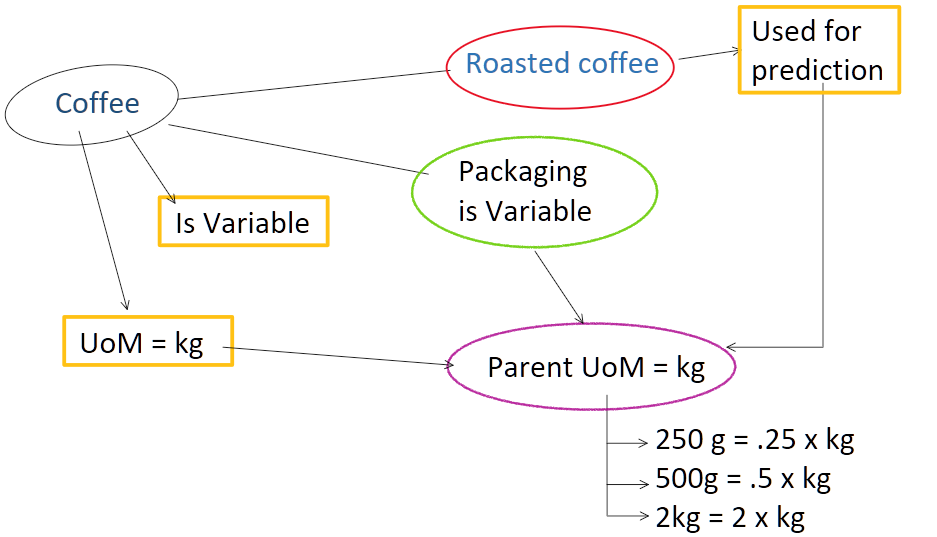


Table version

Lookup

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | **Item Table**  Name: BrazilianCoffee  Parent UoM: kg  Id: XYZ |  | Lookup |  |  |
|  |  |  |  |
|  |  |  |  |  |

**Attribute Vars Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **ItemId** | **AttId** | **AttVarLookupId** | **Name** |
| XYZ | ABC | A001 | 1kg Bag |
| XYZ | ABC | A002 | 250g Box |
| XYZ | ABC | A003 | 500g Bag |

**Attribute Var Lookups Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AttVarLookupId** | **AttVarLookupName** | **UoM** | **ParentUoM** | **PUoMQty** |
| A001 | Kilogram | kg |  |  |
| A002 | 250 gram | g | kg | .25 |
| A003 | 500 gram | g | kg | .5 |

## Import Item/Product form Woo

The Items are imported from Woo, using the Woo Settings.

Each Woo Item has:

* Item Detail stuff (like name, SKU etc)
* Can have varieties via the attribute that are marked as variables.
* Product Category (at least one). Even though Woo allows you to select a primary category this is not communicated via Woo’s REST API. This will need to be set globally. A product can have a category that will be predictable / used for trackability – this has to be set via the Category Lookup table.
* Pricing details.

The system will import all items or just those in stock. All items that are marked a draft are not imported. Only stock status private and publish should be included. This will be done by the user selecting options.

### What Woo Gives Us:

Items are called Products in Woo. Woo gives us a lot per Item the full list can be seen here (https://woocommerce.github.io/woocommerce-rest-api-docs/#products):

###### To pull all product: /wp-json/wc/v3/products

This pulls 20 products per page. What we are interested in (column I / X is used to determine if we import it or eXport it. Legend Y is yes, N is no and ? means perhaps in the future)

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Type | Description | I/X |
| id | integer | Unique identifier for the resource. read-only | Y/N |
| name | string | Product name. | Y/Y |
| permalink | string | Product URL. read-only | N/N |
| type | string | Product type. Options: simple, grouped, external and variable. Default is simple. | Y/Y |
| status | string | Product status (post status). Options: draft, pending, private and publish. Default is publish. | N/Y |
| featured | boolean | Featured Product. Default is false. | N/Y |
| catalog\_visibility | string | Catalog visibility. Options: visible, catalog, search and hidden. Default is visible. | N/Y |
| description | string | Product description. | N/N |
| short\_description | string | Product short description. | Y/Y |
| sku | string | Unique identifier. | Y/Y |
| price | string | Current product price. read-only | N/N |
| regular\_price | string | Product regular price. | Y/Y |
| sale\_price | string | Product sale price. | ?/? |
| on\_sale | boolean | Shows if the Product is on sale. read-only | ?/? |
| purchasable | boolean | Shows if the Product can be bought. read-only | N/? |
| virtual | boolean | If the Product is virtual. Default is false. | ?/? |
| downloadable | boolean | If the Product is downloadable. Default is false. | ?/? |
| manage\_stock | boolean | Stock management at product level. Default is false. | Y/Y |
| stock\_quantity | integer | Stock quantity. | Y/Y |
| stock\_status | string | Controls the stock status of the Product. Options: instock, outofstock, onbackorder. Default is instock. | Y/Y |
| parent\_id | integer | Product parent ID. | Y/Y |
| categories | array | List of categories. See [Product - Categories properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-categories-properties) | Y/Y |
| tags | array | List of tags. See [Product - Tag’s properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-tags-properties) | ?/? |
| images | array | List of images. See [Product - Image’s properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-images-properties) | Y/N? |
| attributes | array | List of attributes. See [Product - Attribute’s properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-attributes-properties) | Y/Y |
| default\_attributes | array | Default variation attributes. See [Product - Default attributes properties](https://woocommerce.github.io/woocommerce-rest-api-docs/#product-default-attributes-properties) | N/N |
| variations | array | List of variations IDs. read-only – used for child items | Y/Y |
| menu\_order | integer | Menu order, used to custom sort products. | ?/? |

When we add items, we include:

* Using the variety, we can link to UoM base
* As part of the Import, we select if we only import Item in stock if so we only pull those. We should also consider hidden products.

#### Item Images

The Item URL is added as string, this then can be used to display the Item (if online)

*Status: Import and display Done*

#### Item Attributes

In the Product->attributes [] array is:

|  |  |  |
| --- | --- | --- |
| Attribute | Type | Description |
| Id | integer | Attribute ID. |
| Name | string | Attribute name. |
| position | integer | Attribute position. |
| Visible | boolean | Define if the attribute is visible on the “Additional information” tab in the Product’s page. Default is false. |
| variation | boolean | Define if the attribute can be used as variation. Default is false. |
| options | array | List of available term names of the attribute. |

We use the variation setting to then set the Attribute Varieties for the Item, these are found in **options**, which lists the Attribute Terms (what we call varieties) we need to map the terms to the variety. This array is imported

*Status: Import and display not complete*

### WooProduct -> Item Table Mapping

Mapping the Products is required for continued updating. Once we know which Product is what we can update it. So we need to Link the WooProduct.Id to Item.Id

##### WooProductToItemMapping

Used to store the link between our Item and the Woo Product.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Comments** |
| WooProductVariantMapId | Guid | DatabaseGeneratedOption.Identity Primary Key |
| ItemID | Guid | **Links to our Item Id** |
| WooProductID | int | **Links to woo attribute** |
| CanUpdate | bool | **Set true by default – false if we do not want the Product to be updated** |

*Status: Import done and display in items Table done*

### Item Table

Here we look at our Item Table and see what we map.

| **Field** | **Type** | **Comments** |
| --- | --- | --- |
| ItemId | Guid | Unique to IT not imported |
| ItemName | string (100) | Item Name |
| SKU | string (50) | The Item’s Stock Unit code |
| IsEnabled | bool | Is it enabled |
| ItemDetail | string (500) | Links to WooProducts.short\_description |
| PrimaryCategoryId | Guid? | What is this items primary category. Used for ? |
|  |  |  |
| ReplacementItemId | Guid? | leave |
| ItemAbbreviatedName | string (10) | Abbreviated name – not imported |
| BasePrice | decimal (18,4) | Linked to regular price |
| SortOrder | Int | linked to WooProducts. menu\_order |
| ManageStock | bool | Do we (or Woo) manage stock |
| QtyInStock | int | Amount in stock, if we manage stock (order must track if order came from woo or manual, if order from woo then we need not send back to woo, else we do |
| ItemType | Enum? | ItemType – Simple, variable, GroupByType, Service, Collection, URL, virtual, other |
| Images | List<Images> | List of URLS in separate table |
| Notes | String | Is a string for extra items |

There may be additional fields for Price support etc. To be done later.

To the class we add:

* a List of Variants using ItemId as a Foreign Key.
* A List of the Categories with ItemId as a Foreign Key to the Item Categories table that links the Item to the Category lookups
* A List of Attributes with ItemId as a Foreign Key to Item Attributes, and there is a list of Item Attribute Terms.

#### Item Types

An item can be of the following types:

* Simple – a simple stock item
* Variable – an item with variations. The variations are linked to at least one type of attribute that is marked as a variable type. This can also link to units of measure.
* GroupByType – used to suppler items that are in a group of similar items to allow a customer to get a different items of the same type each time they order (like for subscriptions.
* Service – like simple but there is no stock support. This allows for labour third party servicing etc.
* Collection – a collection of items. This is used when one Item is used to be a collection of other items
* URL – an item that is a URL or some link to something
* Virtual – any item that is electronically available like a download or something
* Other – any item that does not fit in to any of the other items. Should not really be used

#### Update or Not?

All imported items will have a Boolean associated to their import mapping. If set to true then importing will update the entity and if false will ignore changes (if any)/

### Woo Product Import

Using what has been covered we implement a class WooImportProducts that is an implementation of IWooImportWithParents. Part of the implementation will rely on the implementation if WooImportProductVariant, covered below. Both will rely on a generic mapping class that will implement MapWooDetails

#### Import Logic

There are two options here, one is to retrieve all the products (this may take some time) and one is to get a page at a time and deal with it. To display a percentage (of how far we are) we need to retrieve all products. This may mean we end up throwing an error.

* Get All Woo Products
* Set counters
* For each Product
  + If the Product does not exists add and MapWooProductInfo to new record increase Counter.added
  + If Product does exist and CanUpdate is true then update by copying the WooDetails that are not keys, increase Counter.update

#### Product To Item Mapper

This is used for mapping Woo Products and Variants to our data types. Maps what it can using AutoMapper, and the class setup in the integration Repositories Classes IntegrationMappingProfile. Internal conversion routines are used that map the Product to the Item, as per below

##### Items Mapped

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Maps to | Auto Mapper | Conversion |
| id | WooItemMapping.ItemId | n/a |  |
| name | ItemName | Yes | Truncated |
| type | ItemType | Yes | Converted to Enum |
| status | not imported, we may export | n/a |  |
| featured | not imported, we may export | n/a |  |
| catalog\_visibility | not imported, we may export | n/a |  |
| short\_description | ItemDetail | Yes | Truncated |
| sku | SKU | Yes | Truncated |
| price | BasePrice | Yes | null=> 0.0 |
| virtual | ItemType | Yes | If virtual Item set to virtual |
| manage\_stock | ManageStock | Yes | null=false |
| stock\_quantity | QtyInStock | Yes | null=0 |
| stock\_status | IsEnabled | Yes | stock\_status = "instock" |
| categories | Categories | No | Separate Table |
| images | Images | No | URL saved to ItemImages |
| attributes | Attributes | No | Separate Table |
| default\_attributes | May need | No |  |
| variations | All the item variants | No | Separate Table |
| menu\_order | SortOrder | Yes | 0 = default |

##### Items calculated

|  |  |
| --- | --- |
| Attribute | Calculation |
| ItemAbbreviatedName | First 10 letters of the ItemName that are not vowels or punctuation |
| Notes | Set to update/added date |

MapWooProductInfo:

The maps the info across:

* Set ItemName, SKU, IsEnabled, ItemDetail, SourtOrder as per above Mapping in Table
* For ItemCategotyId – selected the first Category in the array find the CategoryId of it using the WooCategoryMapping and set. We assume the first is primary. Perhaps we should look for one those without parents? Only add if it does not exist.
* If the Item does not exist then also copy the attributes and for each attributes the attribute terms, or varieties.
* For ParentItemId Add to a List ItemParents the ParentId and the ItemId.

Once finished we need to map an Item’s parents. We do this by cycling through the list of items that have parents, then we find the Item and parent’s GUID and take match them.

*Status:*

*Done: Item Categories, Attributes and Attribute Terms are in the Import.*

*To Do: Item variations, Item types, display of the woo mapping tables?*

#### Support Methods

##### Assign Woo Product Category Async

Using the current Item, assign any product categories to it, using our mapped data.

|  |  |
| --- | --- |
| Parameters: | **Item** currItem: Current Item we are working with, or we need to assign the categories too  **Product** currWooProd: Current Woo Product information used to populate. |
| Returns: | The modified Item with the Categories added, if any |

##### Add Or Update Varieties Async

Using the Item we are working Add or update variation of this Item.

|  |  |
| --- | --- |
| Parameters: | **Item** currItem: Current Item we are working with, or we need to assign the Attributes too  **Product** currWooProd: Current Woo Product information used to populate. |
| Returns: | The modified Item with the Attribute Varieties added if any |

##### Add Or Update Item Attributes Async

Using the Item we are working with check if an attribute is assigned, then add if not or update.

|  |  |
| --- | --- |
| Parameters**:** | **Item** currItem: Current Item we are working with, or we need to assign the Attributes too  **Product** currWooProd: Current Woo Product information used to populate.  **bool** HasAttributes: Does the Item have attributes |
| Returns: | The modified Item with the Attribute Varieties added, if any |

##### Assign Woo Product Attributes Async

Using the current Item, assign any product attributes to it, using our mapped data.

|  |  |
| --- | --- |
| Parameters: | **Item** currItem: Current Item we are working with, or we need to assign the Attributes too  **Product** currWooProd: Current Woo Product information used to populate. |
| Returns: | The modified Item with the assigned Attribute Varieties added, if any |

##### Add Product To Items Async

Add Woo Product to Item table as the Id in the Mapping was not found.

|  |  |
| --- | --- |
| Parameters: | **Product** currWooProd: Current Woo Product information used to populate. |
| Returns: | **Guid** of the Item added. |

##### Delete Woo Product Map Async

Delete a Map from the item map table.

|  |  |
| --- | --- |
| Parameters: | **Product** currWooProd: Current Woo Product information used to populate. |
| Returns: | **int** 1 if deleted or Error. |

##### Import Product Variations Async

Import the actual Variations of a variable product into the items table and set currItem.Id as parent. Uses the WoImportProductVariant.ImportProductVariants (see below later) to import the variants.

|  |  |
| --- | --- |
| Parameters: | **Product** newWooEntity: New Woo Product information used to populate.  **Item** sourceEntity: Item that is the parent |
| Returns: | **Item** with children added if added |

#### Interface Methods

##### Get Woo Entity Data Async

Retrieve all products from Woo that are either in stock or not.

|  |  |
| --- | --- |
| Parameters: | **bool** OnlyItemsInStock: Option default = true, should only items in stock be retrieved. |
| Returns: | **bool** Success or failure |

##### Get Woo Mapped Entity Id By Id Async

Retrieve all products from Woo that are either in stock or not.

|  |  |
| --- | --- |
| Parameters: | **uint** sourceWooEntityId: Id of the Woo Product to return. |
| Returns: | **uint** Id of the Woo Product we have in the Mapping table |

##### Add Entity Async

Using the Woo Product as a source add Item to our tables.

|  |  |
| --- | --- |
| Parameters: | **Product** newWooEntity: New Woo Product information to use. |
| Returns: | **Guid** of Item Added. |

##### Add Or Get Entity ID Async

Check if the Item of the same name or SKU exists in the Item database, if so update and return, otherwise add and return it.

|  |  |
| --- | --- |
| Parameters: | **Product** sourceEntity: New Woo Product information to use. |
| Returns: | **Guid** of Item Added. |

##### Add Or Update Entity Async

See if an Item of this Id exists in the item table. If so, update it otherwise added it, returning the Id.

|  |  |
| --- | --- |
| Parameters: | **Product** sourceEntity: The Source Woo Product  **Guid** sourceWooMappedEntityId: The Id we found mapped or added. |
| Returns: | **Guid** Id of the Item Added or Updated |

##### Find And Set Parent Entity Async

For the WooItem passed in get the Item’s Child and Parent and set their Ids.

**Logic**:

* Using the ids passed look for the linked attribute to the Id then look for the ParentId get the GUID’s of each and update the database:
* Get ProdWithAParent.Id Guid from ItemAttributeTable = ParentID
* Get ProdWithAParent.ParentID GUID from ItemAttributeTable = ChildID
* Set the ItemAttribute.ParentId = ParentID for ItemsAttrib.ID = ChildID

|  |  |
| --- | --- |
| Parameters: | **WooItemWithParent** sourceWooEntityWithParent: The Source Woo Product with Parent. |
| Returns: | Id of the Item Added or Updated |

##### Set Woo Entity Parent Async

Using the Id of the Parent for the child item set the parent item.

|  |  |
| --- | --- |
| Parameters: | **Guid** sourceChildWooEntityId: Id of the child Item.  **Guid** sourceParentWooEntityId: Id of the Parent. |
| Returns: | **bool** Success or failure |

##### Update Entity Async

Using the Id of the Parent for the child item set the parent item.

|  |  |
| --- | --- |
| Parameters: | **Product** updatedWooEntity: Id of the child Item.  **Item** updateEntity: Id of the Parent. |
| Returns: | **Guid** Item Id Updated |

##### Update Woo Entity Async

Update the anything in the Mapping we need to update, Item with the data we are importing. If the Mapping is pointing to the wrong place delete it.

|  |  |
| --- | --- |
| Parameters: | **Product** updatedWooEntity: Updated Woo Product.  **WooProductMap** targetWooMap: The Mapping we are targeting. |
| Returns: | **Guid** Item Id Updated |

##### Import And Map Woo Entity Data Async

Import all the Woo Product that is passed.

Logic: Check if the Product does not exists add and copy woo details to new record increase Counter.added. If Product does exist and CanUpdate is true then update by copying the Woo Details that are not keys, increase Counter.update.

|  |  |
| --- | --- |
| Parameters: | **Product** sourceEnity: Updated Woo Product |
| Returns: | **Guid** Item Id Updated |

## Item Variations

An item with the type of variable will have variations. As all we get from Woo is the Import of an item is that it is a variable item, and we then get the variant ids, we will either need to cycle through these, or retrieve all variants to import them. This must also be supported so that the Item Edit can pull and push data to and from the Woo Database.

## Item Variant Table

The Item Variant Table is used for an item that has variants. Each Variant links to the Item Parent via ItemId

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Comments** |
| ItemVariantId | Guid | Unique to IT not imported |
| ItemVariantName | String(100) | Links to WooProductVariants.name |
| SKU | String(50) | Links to WooProductVariant.sku |
| IsEnabled | bool | Links to WooProductVariant.stock\_status = instock |
| ItemId | Guid | Points to the Id of the Item that is the Parent |
| ItemVariantAbbreviation | string(10) | Abbreviated name – not imported |
| BasePrice | decimal(18,4) | Linked to regular price |
| SortOrder | int | linked to WooProductVariants. menu\_order |
| ManageStock | bool | Do we (or Woo) manage stock |
| QtyInStock | int | Amount in stock, if we manage stock (order must track if order came from woo or manual, if order from woo then we need not send back to woo, else we do |
| ImageURL | String(500) | URL to image |
| Notes | String | Is a string for extra Item Variants |

There may be additional fields for Price support, backorder etc. To be done later.

To the class we add a link to the Parent Item via ItemId as a Foreign Key.

### Woo Product Variants Import

Import Product Variant Variants for a Woo Product Variant, adding or updating the variant. It inherits from IWooImportWithAParent, as each Product Variant has only one parent. As all Product Variants are Items, this can be a recursive call. The TEntity = Item, TWooEnty = Variantion (a Woo Rest class) and TWooMapEntity = WooProduct VariantMap. Most of this implementation relies on the WooProduct VariantsImport, as we retrieve a Product Variant then add it. As a Product Variant is a separate type of an item. Originally we were going to have variants as Items but with the fact that Woo Variants are missing a number of factors we now have an ItemVariant table that will store only what is required for a variant.

#### Import Logic

Import is done per Product:

* Get All Woo Products Variants for the Product
* Set counters
* For each product variant
  + If the product variant does not exists add and MapWooVariantInfo to new record increase Counter.added
  + If Product does exist and CanUpdate is true then update by copying the WooVariantDetails that are not keys, increase Counter.update

#### Product Variant To Item Variant Mapper

This is used for mapping Woo Products Variants to our data types. Maps what it can using AutoMapper, and the class setup in the integration Repositories Classes IntegrationMappingProfile. Internal conversion routines are used that map the Product Variant to the Item Variant, as per below:

##### Items Mapped

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Maps to | Auto Mapper | Conversion |
| id | WooMappingTable | Yes |  |
| description | ItemVariantName | Yes | Truncated |
| sku | SKU | Yes | Truncated |
| price | BasePrice | Yes | null=> 0.0 |
| virtual | ItemType | Yes | If virtual Item set to virtual |
| manage\_stock | ManageStock | Yes | null=false |
| stock\_quantity | QtyInStock | Yes | null=0 |
| stock\_status | IsEnabled | Yes | stock\_status = "instock" |
| image | Image | Yes | URL to image |
| menu\_order | SortOrder | Yes | 0 = default |

##### Items calculated

|  |  |
| --- | --- |
| Attribute | Calculation |
| VariantAbbreviatedName | First 10 letters of the ItemName that are not vowels or punctuation |
| Notes | Set to update/added date |

MapWooVariantInfo:

As per the Table above field to field Mapping done in IntegrationMappingProfile, the rest manually in Support Method

Private Variables

|  |  |
| --- | --- |
| WooImportProduct Variant  LocalWooImportProduct Variant | Initialise on class creation then used us the get; set; and a local protected far |
| WooImportProduct Variant  \_WooImportProduct Variant | The local var to store current version of the LocalWooImportProduct Variants |

#### Support Methods

##### Map Woo Product Variant Info

Map the Woo Product Variant to our Item Variant variable

|  |  |
| --- | --- |
| Parameters: | **Variation** sourceWooVariant: Source Product Variant information to use.  **ItemVariant** currItemVariant: Target variant (null if new) |
| Returns: | **ItemVariant** with the information copied across. |

#### Interface Methods

##### Get Woo Entity Data Async

Uses the Woo Parent Id to request all the variants associated to that Product.

Logic: Do the Woo REST call to retrieve the variants for this parent product. nothing returned, return null.

|  |  |
| --- | --- |
| Parameters: | **unit** parentAtteribute: The id of the parent product. |
| Returns: | **List<Variation>** of Product Variants. |

##### Get Woo Mapped Entity Id By Id Async

Looks to see if the current Woo Variant is Mapped using Variant Id.

Logic: Find the first instance of the of the Variant Id in the Mapping, if it exists return it otherwise return Guid.Empty.

|  |  |
| --- | --- |
| Parameters: | **unit** sourceWooEntityId: The id of the product variant. |
| Returns: | **Guid** of Item found in Mapping or Guid.Empty if not. |

##### Add Entity Async

Using the Woo Product Variant as a source add Item to our tables.

Logic: To get here we assume the variant does not exist, so we need to create a new Variant map the data, and save the variant. Return the Id of the variant added, or Guid.Empty if not.

|  |  |
| --- | --- |
| Parameters: | **Variantion** newWooEntity: New Woo Product Variant information to use.  **WooProductVariantMap** sourceWooMap**:** The source Woo Product Variant Map.  **Guid** sourceParentId: The Id of the Parent Item for this Variant. |
| Returns: | **Guid** of Item Added or Guid.Empty if not. |

##### Add Or Get Entity ID Async

Return the Id of found Variant or of a an added with details that we have of this variant.

Logic: Check if the Item variant of the same name or SKU exists in the Item database, if so update check if it has the same parent and return, otherwise add and return it.

|  |  |
| --- | --- |
| Parameters: | **Variation** sourceWooVariant: Source Product Variant information to use.  **Guid** sourceMappedItemVariantId: Id of the Variant we found, is it still there?  **Guid** sourceParentId: The Id of the Parent Item for this Variant. |
| Returns: | **Guid** of Item Added or found or Guid.Empty if not |

##### Add Or Update Entity Async

Add or update (if it exists) an item variant

Logic: See if an Item variant of this Id exists in the item variant table. If so, update it otherwise added it, returning the Id.

|  |  |
| --- | --- |
| Parameters: | **WooItemWithParent** sourceWooEntityWithParent: Parent and Child Id of the Woo Variant. |
| Returns: | **bool** success or failure. |

##### Find And Set Parent Entity Async

Find child and parent id of product variant and set them for item variant.

Logic: With a record of the Child and Parent of a Product variant, find the equivalent in our database and make sure they reference one another.

|  |  |
| --- | --- |
| Parameters: | **WooItemWithParent** sourceWooEntityWithParent: Parent and Child Id of the Woo Variant. |
| Returns: | **bool** success or failure. |

Note: not sure if this is needed at the moment, will throw an exception if executed

##### Update Entity Async

Update the Item Variant we assume is found, using the data we got from Woo on the Product Variant.

Logic: Copy data across and update the Item Variant Table.

|  |  |
| --- | --- |
| Parameters: | **Variation** updatedWooEntity: The Woo Variant that we are working on.  **Guid** sourceParentId: The Id of the Item that this is the variant of.  **ItemVariant** updatedEntity: The Item Variant that we found in the tables. |
| Returns: | **Guid** The id of the item variant updated. |

##### Update Woo Mapping Entity Async

Update the Item Variant mapping we assume is found, using the data we got from Woo and the Product Variant Mapping.

Logic: Update the Mapping Using the current Mapping we have.

|  |  |
| --- | --- |
| Parameters: | **Variation** updatedWooEntity: The Woo Variant that we are working on.  **WooProductMap** targetWooMap: The target product variant to item mapping.  **Guid** sourceParentId: Parent Id of the Item’s Variant. |
| Returns: | **Guid** The id of the item variant updated or updated. |

##### Import And Map Woo Entity Data Async

Import the Woo Product Variant that is passed.

Logic: Check if the product variant does not exists add and copy woo details to new record increase Counter.added. If Product does exist and CanUpdate is true then update by copying the Woo Details that are not keys, increase Counter.update.

|  |  |
| --- | --- |
| Parameters: | **Product** sourceEnity: Source Woo Product Variant.  **Guid** sourceParentId: Parent Id of the Items Variant. |
| Returns: | **Guid** Item Id Updated |

##### Import Product Variants

Actual Name: ImportProductVariants.

Import all the variants for the Item passed in. Can be used on the fly or in a full import.

Logic: Get all the variants for this Product. Loop through the variants and add. If variant exists and CanUpdate is true then update by copying the Woo Details that are not keys, otherwise add.

|  |  |
| --- | --- |
| Parameters: | **Uint** sourceProductId |
| Returns: | **int** number of variants imported or Error |

### Future ideas

* Add an ability to select patterns for abbreviations
* Store changes / updates to each Item, so we can store versions

## Item Variety UoM

This is used for orders and needs to be set per Item Variety. It is imported and has to be set up by editing the Item Varieties. Each Item has attributes, and each attribute can have varieties. Each variety can map to a unit of measure.

This is used to link an Item’s variety to a Unit of Measure. So each, box, bottle, bag, packet, etc.

A box of pills has a UOM of box = 6 of each

A variable of 250g has a UoM of kg and the conversion is .25\*Qty.

So we need to store UoM per variation and link that to a Qty per Base UoM

### Item UoM Table

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Comments** |
| ItemUoMID | Int | Pk |
| UoMName | String(100) | Should be smaller |
| UoMSymbol | String(10) |  |
| BaseUoMID | Int | If>0 points to the BaseUoM |
| BaseConversationFactor | Double |  |
| RoundTo | Int | Number of decimals to round to (default 4) |

These will be linked to the Items through then Items Attribute Varieties, and is optional

## For UOM:

Each Item can have multiple variations (or none). There are two types of variations:

* One variation per SKU per Item. This means that the Item has a one-to-one mapping
* Each variation is a Unit of Measure (UoM) that is linked to a quantity or s description

So for example.

1. Coffee A, can be sold as 250g, 275g, 500g, 750g and 1kg. The variation 250g is a one to one Mapping the others are mapped to quantity of .275, .5,.75, 1 kg of coffee (this uses a parent SKU that. The is determined by the variation. For 1-many Mapping there needs to be a parent id and parent SKU
2. Cleaning Pills come in a box of 6 or bottle of 25. Each variation is a 1-1 mapping to the SKU. This should be the default
3. WaterFilters come in for variables white, blue, smart, smartpro. Each variation is a 1-1 mapping per SKU. Can this be change to typeof filter?

For the user entering in a line item in the order / invoice, just enters in the SKU or can search on Item name (auto complete)

The Table WooVariationMaps stores the category information.

Need to store the parent id, so that you can do reporting. Parentid=0 means no parent (like with woo)

Using -> [https://www.quaffee.com/wp-json/wc/v3/Product Variants?per\_page=5&type=variable&stock\_status=instock](https://www.quaffee.com/wp-json/wc/v3/products?per_page=5&type=variable&stock_status=instock)

Scan through the attributes that have variation set to true.

From those variations look for the categories. Then offer a link between the category and the category options

Display a list of these per category, then allow the offered as options to be set.

Using the variation in each category, link the variation to the option:

* Is a separate SKU -> us parent SKU or not?
* Has UoM -> Qty per UoM

Need to set these using a default offering option

Attribute Options

* Packaging
* Preparation Type
* Size
* Model
* Colour
* options

Packaging

Offered as?

Packaging -> 250/275/500/750/200

Packaging - > Box of / Bottle of / each

Colour - > White / Blue / Red / Grey

Capacity ->

Size -> XS, S, M, L, XL, XXL

Model ->

We also need to add Preparation / Serving

Perhaps should be called Packed

Bottles, Packets,

To be linked to a Product Variant variation, so pull the variations and then link those preparation

## Link of Item to Attributes

Linking of Item to Item to ItemAttributes

ActiveItemAttributes

Is used to store which attributes are active for an item

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeID | Int | Pk |
| ItemID | Int | Link to the Item that has this attribute |
| IsUsedForItemVariety | Bool | Is this used as to create item varieties |
| ItemAttributeVarietiesActiveID | Ink | Fk -> to give a list of Attributes. |

### ActiveItemAttributeVariety

Items have Attributes and Attributes have varieties or in woo language terms. So, an item has a list of attributes that has a list of attribute varieties or terms. Attributes can be used for variation, which links to the packaging / preparation through the attribute variety.

Using that ItemAttributeActive per Item you can see which items are active with for this Item.

|  |  |  |
| --- | --- | --- |
| Desc | Type | Comments |
| ItemAttributeVarietiesActiveID | Int | Pk |
| ~~ItemID~~ | ~~Int~~ | ~~Link to the Item that has this attribute (not needed as no in Attribute Table now)~~ |
| ItemAttributeID | Int | Links to parent attribute (could be excluded) |
| IsDefault | Bool | Is this the default variety (of variety type) |
| UoMID | Int? | ->ItemUoM.ID |
| QtyPerUoM | double(12,6) | The quantity this is for UoM |

This allows us to import:

1. WooAttributes and map them to ItemAttributes
2. WooAttribute->Terms and map them to ItemAttributeVarieties
3. When importing Items (later) add an Attribute per attribute in a WooProduct Variant to Item
4. Also for each WooProduct Variant variation we can add a AttributeVariety and then mark it as use for variable IF it is a variable Product Variant and if this attribute is marked to be used as var

### Mapping of Woo Product Variants to Items

A woo Product Variant has a parent produce and then if there are varieties then child Product Variants. In our Table this is stored using the Parent ID and the Item d. So, an Item with no Parent Id is a Parent otherwise it is a variety of the parent.

So, for each variety we need to lookup that Product Variant from woo and then add that Product Variant to the item table with the correct settings. If a variety is actual a quantity, then we need to determine that from the Unit of Measure, which means we need to link Item to UoM.

So, lets say we have Product Variant X, it is has a UoM of kg. If the varieties are then “g” or “kg” or “kg” we need to link them through UoM. So, Item X with UoM kg can be sold as different types of units but still be linked to the same Item.

Items that are in the same category at any level can be tracked / Predicted. Each Category that is predictable will silo items in that category based on UoM Qty, Date supplied, and actual Item.

## Import of Items with Variants

For each Product Variant only the parent Product Variants are imported, not the children (in the above Import). Now all items that are children need to be imported. Only the items that are not null will be specific to the child item. Otherwise, all items will be inherited.

So, for each parent item who has an attribute that is used for variation (or term) we need to import those children.

### Logic

Using the ItemTbl imported, get all items that have attributes marked as Is Used for Item Variety. Retrieve that Product Variant and then import each variety as a Product Variant.

Note: the predictability has to be set manually for each Category.

### Item To Product Mapper

This is used for Mapping or items to Woo Products Variants. Maps what it can using AutoMapper, and the class setup in the integration Repositories Classes IntegrationMappingProfile. Internal conversion routines are used that map the Product Variant to the Item Variant, as per below:

##### Items Mapped

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Maps to | Auto Mapper | Conversion |
| WooItemMapping.ItemId | Id | n/a |  |
| ItemName | Name | Yes | Truncated |
| SKU | sku | Yes | Truncated |
| ItemDetail | Short\_descritpion | Yes | Copied as is, this may cause an issue |
| IsEnabled | stock\_status | n/a | If is enabled, then set to “instock” otherwise set to “outofstock” |
| SortOrder | menu\_order | Yes | 0 = default |
| BasePrice | price | Yes | Decimal map |
| ManageStock | manage\_stock | Yes | Boolean map |
| ItemType | type | Yes | Converted from Enum to string as per rest api |
| QtyInStock | stock\_quantity | Yes | null=0 |
|  | stock\_status | Yes | stock\_status = "instock" |
| Images | images | No | URL saved to ItemImages |
| Categories | categories | No | Separate Table -> Using the WooMap Table. This needs to look up the id, and if not there added it or updated it. |
| Attributes | attributes | No |
| All the item variants | variations | No |

## Items and Item variations or children

If an item is of the type variable, then it will have child items. The child items can be user-entered or retrieved from woo. A child item with a Unit of Measure will have its usage can be mapped to a parent, based on that Unit of Measure

### Examples:

#### Size:

So a child that is a variation of type size (i.e. XS, S, M, L, XL, XXL) will be the same as having an item; the only thing that changes is that item X has size N.

#### Pills

Pills may be, for example, Vitamin C but may come in a box, small bottle, bottle and large bottle. Box = 10 pills, small bottle=30 pills, bottle = 60 pills, large bottle = 120pills

So essentially, 1x Item of type box is an Item packed in a box of 10 pills.

##### Weight

Take an item of coffee. The parent UoM is kg. Item children are 1kg, 250g box, 275g bag, 500g bag, 750g bag.

On item x .75 = same a one child 750g.

So we need to have a mapping table that maps UoM and variations to the master Product Variant/item. Items need to have a default UoM that then is mapped to the children/variations.

## Summary of Items

Items Categories are used for tracking (if a category is marked as tracked)

Item Attributes are used to determine if an item is an item that has variations.

Variations are mapped to UoM that is mapped to a group of Variations that has a master Variation.

This means we can create groups of variations.

So, do we save everything as 1 of the main variations = N x UoM?

When pulling order data from woo, do we need to link the child Product Variant to the parent Product Variant

E.g. Coffee Blend N –.75 x kg packaging 750g

## Items View and Edit

The items page will display all root items in a DataGrid. By root, it is meant that only the items, not the item variants. When the page is loaded, the app status is checked to see if Woo is in Linked; if so, it enables the Woo linked fields.

### Items Grid - UI

#### Default Sort:

Sorted by Item name

#### Columns:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Displays** | **E** | **F** | **S** | **H** | **Format** | **Validation** |
| Item Name | Item name | Y | Y | Y | N | String | ^\S.{2,100} |
| SKU | Item’s SKU | Y | Y | Y | N | String | ^\S.{2,100} |
| Image | First image in list’s URL is displayed | N | N | N | Y | Image | n/a |
| Enabled | Is the Item enabled | Y | Y | Y | N | Y/N | selection |
| Detail | The detail of the Item | Y | Y | Y | Y | HTML? |  |
| Categories | Categories the Item has – primary is bold, predictive is in italics | Y | Y | ? | Y | List | selection |
| Attributes | Attributes the Item has | Y | Y | ? | Y | List | selection |
| Replace By | If the Item was replaced by another item | Y | Y | Y | Y | String | selection |
| Abrv | Item Abbreviation | Y | Y | Y | N | String | ^\S.{2,10} |
| Base Price | The base price of the item | Y | Y | Y | N | Decimal |  |
| Manage Stock | Is the stock managed by Woo or us – needed for Woo updates | Y | Y | Y | N | Y/N | selection |
| Stock Qty | The qty of this Item in stock 0 means nothing or not tracked | Y | Y | Y | N | Int | \d |
| Woo Linked | Is the Item enabled | Y | Y | Y | N | Y/N | selection |

**Legend***: E= Editable; F= Filterable; S = Sortable, H = hide if small screen*

### Items Grid Code Behind - UI

The Items Grid View and Edit use the IWooLinkedView Interface as a base to provide all CRUD calls. A Repository of the Interface IItemWooLinkedViewRepository is initialised using the following:

TEntity - Item

TEntityView - ItemView

TWooMapEntity – WooProductMap

#### Variables

##### Interface Variables

|  |  |  |
| --- | --- | --- |
| **VariableType VariableName** | **Descriptions** | **Pub/Priv** |
| **GridSettings** \_GridSettings | All the settings required to manipulate the Grid | Pub |
| **ItemView** \_SelectedItemRow | The current selected row in the Grid, used for | Priv |
| **ItemView** seletectedItem | Current selected Item | Pub |
| **BulkAction** SelectedBulkAction | The current selected bulk action | Pub |
| **List**<**ItemView**> SelectedItemRows | The current selected row | Pub |

##### Model Variables

|  |  |  |
| --- | --- | --- |
| **VariableType VariableName** | **Descriptions** | **Pub/Priv** |
| **List<ItemView**> dataModels | The list of all the items expanded to include woo view items | Pub |
| **bool** \_IsLoading | Is the grid loading | Priv |
| **bool** \_ShowItemDetail | Show the item detail grid (user selected) | Priv |
| **bool** \_ShowReplaceItem | Must the replacement item be shown (user selected) | Priv |
| **bool** \_ShowWooLinked | Show if the Item is Woo Lined (user selected) | Priv |
| **string** \_Status | Status string to be displayed | Priv |
| **DataGrid**<**ItemView**> \_DataGrid | The data to be included in the Grid | Priv |
| **IItemWooLinkedView** \_ItemWooLinkedViewRepository | The view repo, used for all the CRUD | Priv |

#### Support Methods

##### On Initialised Async

Initialises \_ItemWooLinkedViewRepository, and updates the page

|  |  |
| --- | --- |
| Parameters: | **Non** |
| Returns: | **Task void** |

##### Set Load Status Async

Sets the Status string and updates the DOM with that string. Logs the change in status

|  |  |
| --- | --- |
| Parameters: | **string** statusString: What to set the string to. |
| Returns: | **Task void** |

##### Load Item List Async

Load the actual Items + View additional values into the data model using the current data grid parameters. Also saves the current selected lists and restores it.

|  |  |
| --- | --- |
| Parameters: | **DataGridParameters** currentDataGridParameters: Grid setting to use to load the items + View additional values |
| Returns: | **Task void** but sets the dataModels variable |

##### Reload Async

Resets the page and reloads the Grid.

|  |  |
| --- | --- |
| Parameters: | **Nonde** |
| Returns: | **Task void** |

#### Interface Methods

##### Handle Read Data Async

Uses the Woo Parent Id to request all the variants associated to that Product. Uses Load Item List.

Logic: Store the current selected items then retrieve details form grid then restore the selected items

|  |  |
| --- | --- |
| Parameters: | **DataGridReadDataEventArgs**<**ItemView**>inputDataGridReadData: Paging, sorting and filtering information. |
| Returns: | **Task void** |

##### Handle Customer Search On Key Up Async

Handle Customer Search On Key UP - essentially adds the key stroke to the filter

|  |  |
| --- | --- |
| Parameters: | **DataGridReadDataEventArgs**<**ItemView**>inputDataGridReadData: Paging, sorting and filtering information. |
| Returns: | **Task void** |

##### On Row Inserting Async

When a row is inserted save that row and reload the Grid.

|  |  |
| --- | --- |
| Parameters: | **SavedRowItem<ItemView, Dictionary<string, object>>** insertedItem: ItemView that was inserted. |
| Returns: | **Task void** |

##### Confirm Add Woo Item Click Async

Handle the user confirmation asking to add the Woo Mapping

|  |  |
| --- | --- |
| Parameters: | **bool** clickConfirmed: Did the user confirm they want to add it. |
| Returns: | **Task void** |

##### On Item New Item Default Setter

When a new item is created initialise the new Item.

|  |  |
| --- | --- |
| Parameters: | **ItemView** newItem: the ItemView to be added. |
| Returns: | **Task void** |

##### Update Item Async

Update the Item that is passed in.

|  |  |
| --- | --- |
| Parameters: | **ItemView** updatedCatItemView: the ItemView to be updated. |
| Returns: | **int** value > 0 for success or -1 for error |

##### On Row Updating Async

On row updating - update the row (using Update Row Async).

|  |  |
| --- | --- |
| Parameters: | **SavedRowItem<ItemView, Dictionary<string, object>>** updatedItem: the ItemView to be updated. |
| Returns: | **Void** |

##### On Row Removing

On row removing async - Launch confirmation dialog, so an item can be deleted if the user wants.

|  |  |
| --- | --- |
| Parameters: | **CancellableRowChange<ItemView>** modelItem: the ItemView to be deleted. |
| Returns: | **Void** |

##### Confirm Delete Woo Item Click Async

On row removing async - Launch confirmation dialog, so an item can be deleted if the user wants.

|  |  |
| --- | --- |
| Parameters: | **bool** clickConfirmed: Did the user confirm they want to add it. |
| Returns: | **Task void** |

##### Confirm Delete Item Click Async

Deletion of the Item for system - and also if confirmed in Woo.

|  |  |
| --- | --- |
| Parameters: | **ConfirmModalWithOption.ConfirmResults** confirmationOption: Confirmation option passed by grid |
| Returns: | **Task void** |

##### On Row Removed

When a row is removed handle this.

|  |  |
| --- | --- |
| Parameters: | **ItemView** modelItem: The ItemView to that was removed. |
| Returns: | **Void** |

##### Do Group Action

Perform the group action as selected.

|  |  |
| --- | --- |
| Parameters: | **none** |
| Returns: | **Void** |

### Items Grid Edit Code Behind - Interface

The implementation of the Interface is handled in ItemWooLinkedViewRepository an implementation of IItemWooLinkedViewRepository. This Interface is separated from the UI in a separate project so that any other project (say Web assembly, Xamarin or OS dependant app).

The Interface uses the definitions laid down in IWooLinkedView. A generic Interface where all specifics are left to the Main Interface to implement. All the external calls are those with the word Row in them; however, the Interface allows external use of the “sub” routines that the Row routines invoke to enable other interfaces to call these.

#### Variables

##### Inherited Variables

| **VariableType VariableName** | **Descriptions** | **Pub/Priv** |
| --- | --- | --- |
| **ILoggerManager** \_Logger | Used for logging to central NLog | Pub |
| **GridSettings** \_GridSettings | All the settings required to manipulate the Grid | Pub |
| **IAppUnitOfWork** \_AppUnitOfWork | How we access all the database stuff | Pub |
| **IMapper \_**Mapper | The Mapper we use for the app | Pub |

#### Support Methods

##### Item Woo Linked View Repository

Initialises \_ItemWooLinkedViewRepository as per parameters passed in.

|  |  |
| --- | --- |
| Parameters: | **All inherited variables as per above Table** |
| Returns: | **Class instance** |

#### Inherited Methods

These routines are inherited and not overridden:

##### Get Woo API Settings Async

Retrieves the API settings

|  |  |
| --- | --- |
| Parameters: | **None** |
| Returns: | **WooAPISettings** |

##### Woo Is Active Async

Is Woo currently active.

|  |  |
| --- | --- |
| Parameters: | **ApplicationState** currentApplicationState: The current application state (an an injected service) |
| Returns: | **bool** true or false depending on whether Woo is currently active. If acting must set the state. So normally the currentApplicationState is updated on return. |

#### Support Methods

##### Add Item Categories To Woo Product Sync

Add categories in the addedEntity to the targetProduct using the Woo Mappings we have. This is only called when a complete product is added.

**Logic**: If not null get the mapping using our current Item Category GUID so we can use the Woo values

|  |  |
| --- | --- |
| Parameters: | **Item** addEntity: Entity / Item to be added.  **Product** targetProduct: Woo Product we are working with. |
| Returns: | **Product** we have added the Categories too. |

##### Add Item Attributes To Woo Product Sync

Add Attributes in the addedEntity to the targetProduct using the Woo Mappings we have. This is only called when a complete product is added.

**Logic**: If not null get the mapping using our current Item Category GUID so we can use the Woo values

|  |  |
| --- | --- |
| Parameters: | **Item** addEntity: Entity / Item to be added.  **Product** targetProduct: Woo Product we are working with. |
| Returns: | **Product** we have added the Attributes too. |

##### Add Item to Woo Only Sync

Add Item to woo, after mapping it. Rev 1.1 – Added support to copy across Attributes and Catgories from item.

**Logic**: using the added Entity map the relevant fields and add it to Woo.

|  |  |
| --- | --- |
| Parameters: | **Item** addEntity: Entity / Item to be added. |
| Returns: | **Product** addedto Woo |

##### Add Item to Woo Item Map Async

Using the new woo Product and new Item add a mapping between the two in the mapping table  
**Logic**: map values across and add to mapping table

|  |  |
| --- | --- |
| Parameters: | **Product** newWooProduct: New Woo Product  **Item** addEntity: Entity / Item to be added. |
| Returns: | **int** result of adding the entity |

##### Delete Woo Product Mapping Async

Delete a product mapping in the Item to product Mapping table

**Logic**: use repo to delete.

|  |  |
| --- | --- |
| Parameters: | **WooProductMap** currentSelectedItems: Current selected items. |
| Returns: | **Task void** |

##### Get IWooProduct Async

Get and interface to the Woo Product API

**Logic**: If we have not already created it create it and return it.

|  |  |
| --- | --- |
| Parameters: | None |
| Returns: | **IWooProduct instance** |

##### Delete Woo Product Async

Delete a woo product.  
**Logic**: use repo to delete.

|  |  |
| --- | --- |
| Parameters: | **WooProductMap** deleteWooProductMap: Current woo product mapping. |
| Returns: | **Task void** |

#### Item Options Support Methods

An item has categories and attributes. These are link to a the systems attribute, attribute terms and categories looup up tables. These lookup tables can be mapped to woo too. Item variants exists in the items has an attribute that is marked as used for variants, and then each item variant is mapped to attribute terms linked to the attribute. While the UI can get, add, update and delete in memory the actuall database CRUD operations for these option are only performed once the User has completed their task. For all CRUD operation the database is update as normal. If there is Woo (or similar) integration then a product must be retrieved and any changes checked for so that the whole updated product is sent to the REST API.

This is important to note when considering the Support Methods below. All “in memory” changes are applied as they happen, all add and update will need to retrieve the product and do a comparison.

So these support methods reflect this.

##### AddItemCategoriesToWooProduct

Here we check if any categories have been added or removed from the WooProduct.Categories. If so we set the HasChanged to true and modify the sourceProduct.  
**Logic**: For each Item Category name check if the name exists as a category. If not add it. Then for each product category check if that exist, if not delete it.

|  |  |
| --- | --- |
| Parameters: | **ItemCategory** sourceItemCategory: Entity / Item to be added.  **Product** originalProduct: The Woo Product we are working with |
| Returns: | **Product** originalProduct modifid. |

##### CheckIfCategoriesHaveChanged

Here we check if any categories have been added or removed from the WooProduct.Categories. If so we set the HasChanged to true and modify the sourceProduct.  
**Logic**: For each Item Category name check if the name exists as a category. If not add it. Then for each product category check if that exist, if not delete it.

|  |  |
| --- | --- |
| Parameters: | **Product** originalProduct: The Woo Product we are working with  **Item** addEntity: Entity / Item to be added. |
| Returns: | **ClassHasChanged<Product>** A class that return true if changes, and includes the changed product. |

##### AddItemAttributesToWooProduct

Here we check if any Attributes have been added or removed from the WooProduct.Attributes. If so we set the HasChanged to true and modify the sourceProduct.  
**Logic**: For each Item Attribute name check if the name exists as a Attribute. If not add it. Then for each product Attribute check if that exist, if not delete it.

|  |  |
| --- | --- |
| Parameters: | **ItemAttribute** sourceItemAttribute: Entity / Item to be added.  **Product** originalProduct: The Woo Product we are working with |
| Returns: | **Product** originalProduct modifid. |

##### CheckIfAttributesHaveChanged

Here we check if any Attributes have been added or removed from the WooProduct.Attributes. If so we set the HasChanged to true and modify the sourceProduct.

Terms?

**Logic**: For each Item Attribute name check if the name exists as a Attribute. If not add it. Then for each product Attribute check if that exist, if not delete it.

|  |  |
| --- | --- |
| Parameters: | **Product** originalProduct: The Woo Product we are working with  **Item** addEntity: Entity / Item to be added. |
| Returns: | **ClassHasChanged<Product>** A class that return true if changes, and includes the changed product. |

Add Woo Product Attribute Async

Add an Attribute to a Woo Product.  **Logic**: Using the ItemAttribute passed in get the product map to this item and then using that Id call the ProductAttribute REST API to add the Product Attribute.

|  |  |
| --- | --- |
| Parameters: | **WooProductMap** deleteWooProductMap: Current woo product mapping. |
| Returns: | **Task** int number of records added, 0 if product does not exist or Error if there was one |

##### Add Woo Product Attribute Term Async

Delete a woo product.  
**Logic**: use repo to delete.

|  |  |
| --- | --- |
| Parameters: | **WooProductMap** deleteWooProductMap: Current woo product mapping. |
| Returns: | **Task void** |

##### Add Item Attribute Async

Delete a woo product.  
**Logic**: use repo to delete.

|  |  |
| --- | --- |
| Parameters: | **WooProductMap** deleteWooProductMap: Current woo product mapping. |
| Returns: | **Task void** |

##### Add Item Attribute Term Async

Delete a woo product.  
**Logic**: use repo to delete.

|  |  |
| --- | --- |
| Parameters: | **WooProductMap** deleteWooProductMap: Current woo product mapping. |
| Returns: | **Task void** |

#### Interface Methods

##### Push Selected Items

Saves the current selected items, so they can be retrieved later with Pop

Logic: Store current selected items for later retrieval

|  |  |
| --- | --- |
| Parameters: | **List<ItemView>** currentSelectedItems: Current selected items. |
| Returns: | **Task void** |

##### Pop Selected Items

Retrieves the last pushed selected items.

Logic: Using the stored list see if those items exists still and add them to a selected list

|  |  |
| --- | --- |
| Parameters: | **List<ItemView>** modelViewItems: Current model view items. |
| Returns: | **List<ItemView>** of items that were popped if they exist |

*Get Woo Mapped Item Async*

Search and return for the woo mapped Item

Logic: Find the first Woo item mapped to Item Guid mapped to a Product

|  |  |
| --- | --- |
| Parameters: | **Guid** mapWooEntityID: Item Id to search for. |
| Returns: | **WooProductMap** if it exists otherwise null |

*Get Woo Mapped Items Async*

Search and return for the woo mapped items using the list sent in.

Logic: Get all mapped entities from the WooProductMapRepo using the Contains condition

|  |  |
| --- | --- |
| Parameters: | **Lisr<Guid>** mapWooEntityIDs: All the Item Ids to search for. |
| Returns: | **List<WooProductMap>** List of Product Mappings found – null if none |

##### Get Paged Items Async

Return all the items as per the specific data a grid settings.

Logic: Mainly relies on the ItemRepo implementation which uses the parameters to apply search, filter and paging info to an eager load.

|  |  |
| --- | --- |
| Parameters: | **DataGridParameters** currentDataGridParameters: Current grid parameters |
| Returns: | **List<Item>** List of Items as per parameters – null if none |

##### Get Data Grid Current

Retrieve the current data grid parameters using the values as per set by user and user interaction

**Logic**: get current DataGridParamters from UI paging info and add any filter requirements

|  |  |
| --- | --- |
| Parameters: | **DataGridReadDataEventArgs<ItemView>** inputDataGridReadData: Current grid parameters specific to the ItemView.  string inputCustomerFilter: the string that the user has entered into the general filter |
| Returns: | **DataGridParameters** |

##### Load View Items Paginated Async

Load all the Item and view item data into a single list to be displayed in the UI

**Logic**: Get all paged items using the user selected options. Add the Woo related data specific to if we are integrated with a REST API

|  |  |
| --- | --- |
| Parameters: | **DataGridParameters** currentDataGridParameters: Current grid parameters |
| Returns: | **List<ItemView>** List of ItemViews as per parameters – null if none |

##### New Item Default Setter

Initialise the new Item with default values. These values are then overwritten by the user.

**Logic**: Set each part of the Item record to a default value.

|  |  |
| --- | --- |
| Parameters: | **ItemView** newViewEntity: blank Item passed in |
| Returns: | **ItemView** that is initialised. |

##### Is Duplicate Async

Is the Item a duplicate of an item that already exists.  
**Logic**: Check in an item with that name or SKU exists, if so return true

|  |  |
| --- | --- |
| Parameters: | **Item** targetEntity: Item to check |
| Returns: | **bool** true if it exists. |

##### Is Valid

Is the Item a valid item.  
**Logic**: Check all required fields are there.

|  |  |
| --- | --- |
| Parameters: | **Item** targetEntity: Item to apply the selected action to |
| Returns: | **bool** true if it is valid. |

##### Do Group Action Async

Perform the selected group action on the selected items.  
**Logic**: Perform action - dependant of the Group Action specified.

|  |  |
| --- | --- |
| Parameters: | **ItemView** targetViewEntity: Target item with view additions to apply action to.  **BulkAction** selectedAction: selected action to perform |
| Returns: | Number of records updated or Error if no updates happened. 0 if no action selected or action does not have logic. |

##### Get Item From View

Takes the Item of Type Item View and returns the mapped Item  
**Logic**: Use Mapper to map across values to a new item

|  |  |
| --- | --- |
| Parameters: | **Item** fromVeiwEntity: the View Item to map from. |
| Returns: | **Item** new Item with the mapped values. |

##### Is Insert Row Async

Insert the new Item that user has added. Displaying relevant messages along the way  
Logic: if the Item does not exist then add it. If the integration (eg. Woo) mapping is enabled then add the Item to the integrated system and mapping table.

|  |  |
| --- | --- |
| Parameters: | **ItemView** newVeiwEntity: The new Item View that has been created and needs to be saved. |
| Returns: | **Void** |

##### Get Woo Product Map From ID Async

Find a Woo Product Mapping that is mapped to the source Item’s Id.  
Logic: Check if a mapping exists related to the Item’s Id passed in.

|  |  |
| --- | --- |
| Parameters: | **Guid** sourceWooEntityId: the View Item to map from. |
| Returns: | **WooProductMap** of Item found or null if not |

##### Add Woo Item And Map Async

Add the Item to woo and map that Item to the Item passed in.  
Logic: Check if a mapping exists related to the Item’s Id passed in.

|  |  |
| --- | --- |
| Parameters: | **Item** addEntity: Item that we have added to the system. |
| Returns: | **int** number of records added, or Error |

##### Add Woo Item And Map Async

Add the Item to woo and map that Item to the Item passed in.  
Logic: Check if a mapping exists related to the Item’s Id passed in.

|  |  |
| --- | --- |
| Parameters: | **Item** addEntity: Item that we have added to the system. |
| Returns: | **int** number of records added, or Error |

##### Delete Row Async

Delete the Item passed in from the Item table.  
Logic: Do the delete using the item id. if success display that otherwise display Error

|  |  |
| --- | --- |
| Parameters: | **ItemView** deleteViewEntity: Item that we have added to the system. |
| Returns: | **void** |

##### Update Woo Mapping Async

Depending on the current setting, update whether the woo Product map for the view item can be updated.  
Logic: Retrieve Item from the database. If it exists and the status has changed (i.e we could update, but we now we cannot and vice versa), then update the can map status. If it did not exists, add the Mapping.

|  |  |
| --- | --- |
| Parameters: | **ItemView** updatedViewEntity: Item view we want to update if required. |
| Returns: | **int** number of records updated, or zero means item no longer there. OR return Error if there was one. |

##### Update Woo Item Async

Update the woo product item using the view item that was edited.  
Logic: If the updated Item has an attribute map and can be updated, then Retrieve the Product from Woo using the item id. If it does not exist, add it. If it does, update it if the name has changed.

|  |  |
| --- | --- |
| Parameters: | **ItemView** updatedViewEntity: Item view we want to update if required. |
| Returns: | **int** id of the Product updated or Error, or 0 if nothing was done. |

##### Update Woo Item And Mapping Async

Checks if any of the Woo link values were changed during an edit; if so, update.  
Logic: Update the view item; if it is updated, then show a success message, otherwise an error. Update the Mapping, return the result of Mapping.

|  |  |
| --- | --- |
| Parameters: | **ItemView** updatedViewEntity: Item view we want to update if required. |
| Returns: | **int** id of the Product updated or Error, or 0 if nothing was done. |

##### Update Item Async

Update the Item using the View Item data we have from the system.  
Logic: Retrieve the item by Id; if it does not exist, show error, otherwise update item and map data across and update in the database using update woo item and mapping sync.

|  |  |
| --- | --- |
| Parameters: | **ItemView** updatedViewEntity: Item view we want to update. |
| Returns: | **int** number of records updated, or zero means item no longer there. OR return Error if there was one. |

##### Update Item Row Async

Master update for the view entity that calls all the others.  
Logic: Check if the Item is valid; if so, update item Async. If successful, then display a message or if there was an error, then an error message.

|  |  |
| --- | --- |
| Parameters: | **ItemView** updatedViewEntity: Item view we want to update. |
| Returns: | **void** |

### Item Detail Editing - UI

Editing will occur via a separate edit page, and editing has more detail than the view. This form is also used for creating new Items.

#### UI Layout

The Editing screen is laid out so that the important and most common fields are in a card on the left. On the right will be extra information, like category detail for that Item. Below that card will the attributes with the and without variation attribute shown all the children items of this Item. If this Product Variant that has variations. They are listed below.

##### Detail portion

The details portion is divided into the most common items on the left and the least common items on the right.

###### Left Hand Details

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Field Name** | **Displays** | **E** | **H** | **Format** | **Validation** |
| Item Name | Item name | Y | N | String | ^\S.{2,100} |
| SKU | Item’s SKU | Y | N | String | ^\S.{2,100} |
| Enabled | Is the Item enabled | Y | N | Y/N | selection |
| Detail | The detail of the Item | Y | Y | HTML |  |
| Abrv | Item Abbreviation | Y | N | String | ^\S.{2,10} |
| Sort Order | Order the item is sorted | Y | N | Int | \[0-9]+ |
| Woo Linked | Is the item enabled | Y | N | Y/N | selection |
| Base Price | The base price of the Item | Y | N | Decimal |  |
| Manage Stock | Is the stock managed by Woo or us – needed for Woo updates | Y | N | Y/N | selection |
| Qty in Stock | The qty of this Item in stock 0 means nothing or not tracked | Y | N | Int | \d |
| Image | First image in list’s url is displayed, can be change | Y | N | Image | n/a |

**Legend***: E= Editable; H = hide if small screen*

###### Right Hand Details

Is broken in to two collapsibles, a general one:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Field Name** | **Displays** | **E** | **F** | **S** | **H** | **Format** | **Validation** |
| Primary Category | Category this Item is most associated to. | Y | Y | Y | N | String/lookup | Selection |
| Replace By | If the Item was replaced by another item | Y | Y | Y | Y | String | selection |
| Notes | Any notes about the Item | Y | Y | Y | N | String |  |

And then the Attributes grid

##### Categories Grid

This will be a child component to list Categories that are allocated to an Item are used for information and also for prediction capabilities. The categories appear in a grid as per below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Displays** | **E** | **F** | **S** | **H** | **Format** | **Validation** |
| Category | ItemCategoryDetail.CategoryName | Y | Y | Y | N | String/lookup | Selection |
| Predictive | ItemCategoryDetail UsedForPrediction | N | N | Y | N | Y/N | Selection |
| UoM Base | ItemUoMBase.UoMName | N | N | Y | N | String/Lookup | Selection |

**Legend***: E= Editable; F= Filterable; S = Sortable, H = hide if small screen*

Categories can be added or deleted.

*Note*: Prediction needs to be at a category level, since multiple items can below to one category. Single item prediction will require its own Category.

##### Attributes

Attributes are associated to an item for information and also to allow for variations. Attributes will appear in a master-detail grid. When some clicks on the attribute then the terms associated to that attribute will appear under that Grid.

###### Parent Attribute Grid

This is what the parent attribute grid will display:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Displays** | **E** | **F** | **S** | **H** | **Format** | **Validation** |
| Attribute | ItemAttributeDetail.AttributeName | Y | Y | Y | N | String/lookup | Selection |
| Used for Variation | IsUsedForItemVariety | Y | Y | Y | N | Y/N | Selection |
| Attributes | Text of attributes in a string collation | N | N | Y | N | String | Selection |

**Legend***: E= Editable; F= Filterable; S = Sortable, H = hide if small screen*

###### Child Attribute Grid

This is what the parent attribute grid will display:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Displays** | **E** | **F** | **S** | **H** | **Format** | **Validation** |
| Attribute Variety | ItemAttributeVarietyDetail.VarietyName | Y | Y | Y | N | String/lookup | Selection |
| Default? | IsDefault | Y | Y | Y | N | Y/N | Selection |
| UoM | UoM.UoMName | Y | Y | Y | N | String/lookup | Selection |
| UoM/Item | UoMQtyPerItem | Y | Y | Y | N | String/lookup | Selection |

**Legend***: E= Editable; F= Filterable; S = Sortable, H = hide if small screen*

##### Item Variants

If the Item has variants this is displayed. Variants are displayed in a grid.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Displays** | **E** | **F** | **S** | **H** | **Format** | **Validation** |
| Name | ItemVariantName | Y | Y | Y | N | String | ^\S.{2,100} |
| SKU | SKU | Y | Y | Y | N | String | ^\S.{2,50} |
| Enabled | IsEnabled | Y | Y | Y | N | Y/N | Selection |
| Abrv | Item Abbreviation | Y | Y | Y | N | String | ^\S.{2,10} |
| Base Price | The base price of the item | Y | Y | Y | N | Decimal |  |
| Sort Order | Order the item is sorted | Y | Y | Y | N | Int | \[0-9]+ |
| Manage Stock | Is the stock managed by Woo or us – needed for Woo updates | Y | Y | Y | N | Y/N | Selection |
| Stock Qty | The qty of this Item in stock 0 means nothing or not tracked | Y | Y | Y | N | Int | \d |
| Image | The first image in the list’s URL is displayed, can be change | Y | N | N | Y | String |  |
| Notes | Any notes about the Item | Y | Y | Y | N | String |  |

**Legend***: E= Editable; F= Filterable; S = Sortable, H = hide if small screen*

#### Buttons and Commands

Besides the standard buttons and commands in the Grid, there are buttons at the bottom.

* Resync button – to pull the latest items details from Woo / Integrations – confirms then resyncs. Disabled if no woo/integration
* Save button – to save changes
* Cancel button – to not save changes – should display a warning

#### Changes and Woo

Attributes, attribute terms can be edited, modified and deleted. The resultant change must be reflected in Woo item is updated only if Item’s Woo Linked is set.

### Item Detail Code Behind - UI

Just like the Grid, the Item Edit uses the IWooLinkedView Interface as a base to provide all CRUD calls. The same Repository that is used in the Grid is used here for the three grids that will be on the page, namely for:

* Item Category
* Item Attributes
* Item Variants

These will be similar CRUD calls. Item Attribute and Item Attributes CRUD should be handled by the same Interface that was used for the Grid view.

An Interface will be used for the Item Variants grid. Here the IWooLinkedView interface is a base to provide all CRUD calls. A Repository of the Interface IItemVariantWooLinkedViewRepository is initialised using the following:

TEntity - ItemVariant

TEntityView - ItemVariantiew

TWooMapEntity – WooProductVariantMap

The attributes and Attributes that are added, edited, or removed will need similar interfaces as already covered in the sections above.

All these need to use the Mapper (like they are used in Import)

#### Product Variant To Item Variant Mapper

Used for mapping Woo Products Variants to our data types. Maps what it can using AutoMapper, and the class set up in the integration Repositories Classes IntegrationMappingProfile. Internal conversion routines have been used to map the Product Variant to the Item Variant, as per below:

##### Items Mapped

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute | Maps to | Auto Mapper | Conversion |
| id | WooMappingTable | Yes |  |
| description | ItemVariantName | Yes | Truncated |
| sku | SKU | Yes | Truncated |
| price | BasePrice | Yes | null=> 0.0 |
| virtual | ItemType | Yes | If virtual Item set to virtual |
| manage\_stock | ManageStock | Yes | null=false |
| stock\_quantity | QtyInStock | Yes | null=0 |
| stock\_status | IsEnabled | Yes | stock\_status = "instock" |
| image | Image | Yes | URL to image |
| menu\_order | SortOrder | Yes | 0 = default |

These mappings are used when you add and update the items.

#### Item Edit code behind

The code behing needs to allow CRUD for the item and the grids. As already mentioned the grids will implement their own interface we use for all grids, that do the CRUD wor, broken down into Entiry, ViewEntity and WooMapEntity.

The code behind the Item Edit razor form needs to handle CRUD for the complete item, including

* All the Item details
* The item Attributes assocaited to the item, and the parent UOM
* All Item attributes, including variations
* Item variants if the item is of type variable.

Detail for what is editable is covered under the UI section above. The view extension of the item will deal with Attribute and Attributes as they are part of the item. However, the ItemVariant is a separate view item. This essentially means, as already mentioned there are two Interfaces. The one will expand on the interface used in the grid view (to add Category and Attribute support), the other is a separate interface that will deal only with Item Variants. The Item interface above could be left as is or inherited and expanded. If we examine the Woo Product definition (via the WooCommerce.net) we se that this definition includes the attributes and the Attributes as part of the definition, so it makes sense to include it in the interface, and hence expand the interface. So next to each interface description you will see a Rev 1.1 for each routine that had support added to include attributes and Attributes. As these are also not mapped directly, they are included in the existing map. So upates to the Item that are Attribute and Category specific will use these additions.

### Item Category Component – UI

This component takes the ItemsCategories and displays them in a grid.

##### Categories Grid

This will be a child component to list Categories that are allocated to an Item are used for information and also for prediction capabilities. The categories appear in a grid as per below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Column Name** | **Displays** | **E** | **F** | **S** | **H** | **Format** | **Validation** |
| Category | ItemCategoryDetail.CategoryName | Y | Y | Y | N | String/lookup | Selection |
| Predictive | ItemCategoryDetail UsedForPrediction | N | N | Y | N | Y/N | Selection |
| UoM Base | ItemUoMBase.UoMName | N | N | Y | N | String/Lookup | Selection |

**Legend***: E= Editable; F= Filterable; S = Sortable, H = hide if small screen*

Categories can be added or deleted.

*Note*: Prediction needs to be at a category level, since multiple items can below to one category. Single item prediction will require its own Category.

This is a in memory grid so most of the items are standard. The code behind handles the CRUD which implete an In memory implementation of I.

Each line of the grid can be ediatable, unlike Woo where the categories are tickled here they are added or deleted into the grid. Editing changes the category using a select list.

### Item Detail Code Behind – Interfaces

#### IItemVariantWooLinkedViewRepository

The is the CRUD Repo that is used to Implement the UI an

#### Interface Variables

|  |  |  |
| --- | --- | --- |
| **VariableType VariableName** | **Descriptions** | **Pub/Priv** |
| GridSettings \_GridSettings | All the settings required to manipulate the Grid | Pub |
| ItemView \_SelectedItemRow | The currently selected row in the Grid, used for | Priv |
| ItemView seletectedItem | Current selected Item | Pub |
| BulkAction SelectedBulkAction | The currently selected bulk action | Pub |
| List<ItemView> SelectedItemRows | The current selected row | Pub |

#### Model Variables

|  |  |  |
| --- | --- | --- |
| **VariableType VariableName** | **Descriptions** | **Pub/Priv** |
| List<ItemView> dataModels | The list of all the items expanded to include woo view items | Pub |
| bool \_IsLoading | Is the grid loading | Priv |
| bool \_ShowItemDetail | Show the item detail grid (user selected) | Priv |
| bool \_ShowReplaceItem | Must the replacement item be shown (user selected) | Priv |
| bool \_ShowWooLinked | Show if the Item is Woo Lined (user selected) | Priv |
| string \_Status | Status string to be displayed | Priv |
| DataGrid<ItemView> \_DataGrid | The data to be included in the Grid | Priv |
| IItemWooLinkedView \_ItemWooLinkedViewRepository | The view repo, used for all the CRUD | Priv |

#### Support Methods

#### Interface Methods

#### IItemCategoryGridViewRepository

Inherits from [IGridViewRepository](#_IGridViewRepository), which implements the following a generic version to support the GridView CRUD. There are three routines that need to be overwritten to suit the category Grid view:

* NewViewEnityDefaultSetter
* IsValid
* IsDuplicate

##### Overwritable Grid Classes

###### New View Entity Default Setter

Used to create a new entity when the add item button is pressed, this needs to be overwritten at a class specific level.  
Logic: return null, as we have no idea what will make the generic class valid.

|  |  |
| --- | --- |
| Parameters: | **TEntity** newEntity: blank entity to be initialised |
| Returns: | **null**. |

###### Is Duplicate Async

See if entity a duplicate, this needs to be overwritten at a class specific level.  
**Logic**: return false, as we have no idea what will make the generic class a duplicate.

|  |  |
| --- | --- |
| Parameters: | **TEntity** sourceEntity: Entity to be checked |
| Returns: | **Bool** true if a duplicate false if not. |

###### Is Valid

See if entity valid, this needs to be overwritten at a class specific level.  
**Logic**: return true, as we have no idea what will make the generic class valid.

|  |  |
| --- | --- |
| Parameters: | **TEntity** sourceEntity: Entity to be checked |
| Returns: | **Bool** true if a valid false if not. |

##### Additional Grid Classes

###### Get Item Category By Id Async

## Get the Item Category Lookup Referenced by its Id. **Logic**: Use generic GetById to return the ItemCategoryLookup by Id.

|  |  |
| --- | --- |
| Parameters: | **Guid** sourceItemCategoryLookupId: Id to search for |
| Returns: | **ItemCategoryLookup** ItemCategoryLookup item, if found or null |

###### Get Item UoM By Id Async

## Get the Item UoM Lookup Referenced by its Id. **Logic**: Use generic GetById to return the ItemUoMLookup by Id.

|  |  |
| --- | --- |
| Parameters: | **Guid** sourceItemUoMLookupId: Id to search for |
| Returns: | **ItemUoMLookup** ItemUoMLookup item, if found or null |

#### IItemAttributeGridViewRepository

Inherits from [IGridViewRepository](#_IGridViewRepository), which implements the following a generic version to support the GridView CRUD. There are three routines that need to be overwritten to suit the Attribute Grid view:

* NewViewEnityDefaultSetter
* IsValid
* IsDuplicate

##### Overwritable Grid Classes

###### New View Entity Default Setter

Used to create a new entity when the add item button is pressed, this needs to be overwritten at a class specific level.  
Logic: return null, as we have no idea what will make the generic class valid.

|  |  |
| --- | --- |
| Parameters: | **TEntity** newEntity: blank entity to be initialised |
| Returns: | **null**. |

###### Is Duplicate Async

See if entity a duplicate, this needs to be overwritten at a class specific level.  
**Logic**: return false, as we have no idea what will make the generic class a duplicate.

|  |  |
| --- | --- |
| Parameters: | **TEntity** sourceEntity: Entity to be checked |
| Returns: | **Bool** true if a duplicate false if not. |

###### Is Valid

See if entity valid, this needs to be overwritten at a class specific level.  
**Logic**: return true, as we have no idea what will make the generic class valid.

|  |  |
| --- | --- |
| Parameters: | **TEntity** sourceEntity: Entity to be checked |
| Returns: | **Bool** true if a valid false if not. |

##### Additional Grid Classes

###### Get Item Attribute By Id Async

## Get the Item Attribute Lookup Referenced by its Id. **Logic**: Use generic GetById to return the ItemAttributeLookup by Id.

|  |  |
| --- | --- |
| Parameters: | **Guid** sourceItemAttributeLookupId: Id to search for |
| Returns: | **ItemAttributeLookup** ItemAttributeLookup item, if found or null |

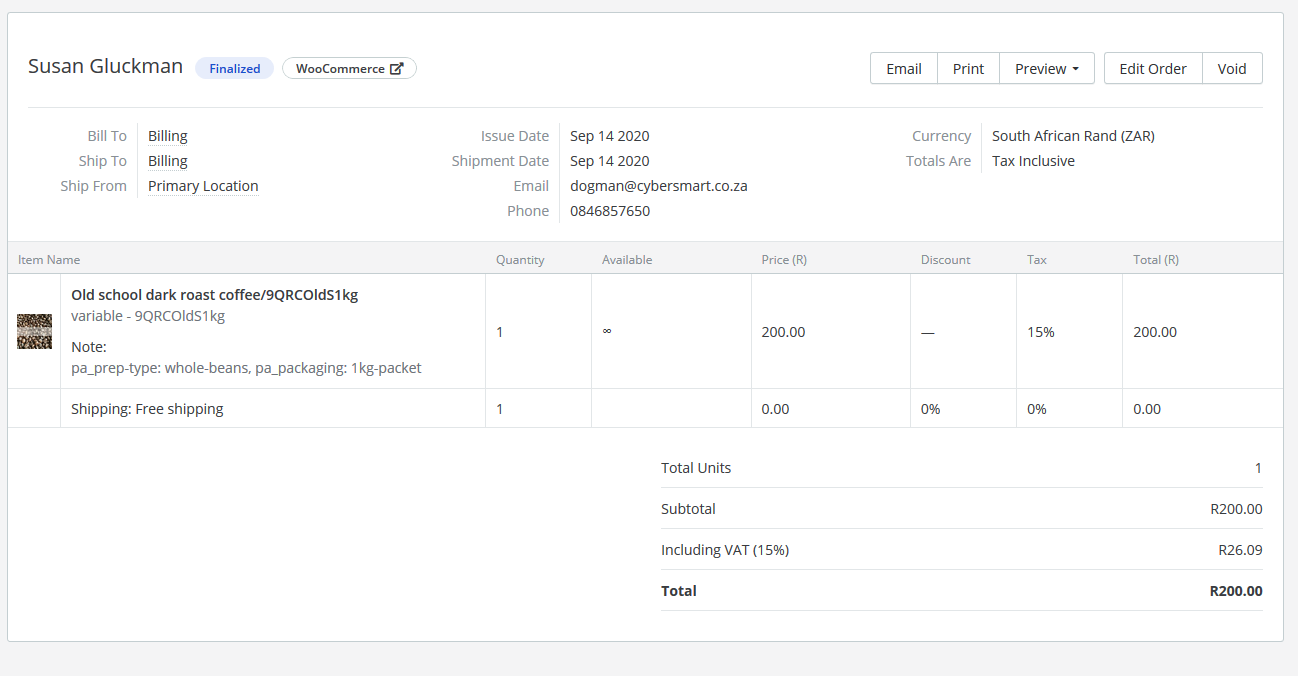
###### Get Item UoM By Id Async

## Get the Item UoM Lookup Referenced by its Id. **Logic**: Use generic GetById to return the ItemUoMLookup by Id.

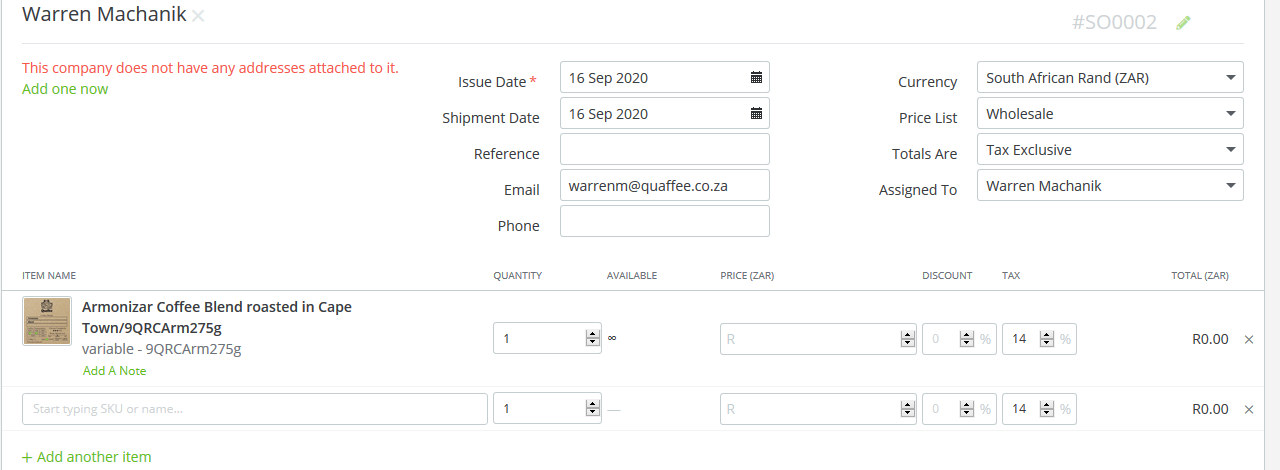
|  |  |
| --- | --- |
| Parameters: | **Guid** sourceItemUoMLookupId: Id to search for |
| Returns: | **ItemUoMLookup** ItemUoMLookup item, if found or null |

# Orders

Orders have the notes in them that include the packaging and prep type variables:



You can add notes



Orders Status:

When order is closed then Rest call:

PUT - https://www.quaffee.com/wp-json/wc/v3/orders/21174?status=completed&consumer\_key=ck\_3b3fffb1d97832e4af0041104b1b839fde66dc02&consumer\_secret=cs\_06da46d3e7b3f0d62a501368afdae8f138cbb0a3

# UI Messages

There are three type of message the system sends to the user:

1. Messages that require no user response
2. Messages that require use acknowledgment and
3. Messages that require a response

## No Response required

This type of message appears when something is done, or if we want the user to note something. For this type of message we will uses toasts. As implemented here: <https://github.com/Blazored/Toast>.

This will be the most common way of sending a message. Each time the message is sent it will also be logged. Using the ILogger service. So, this means that a Component will need to be written that combines the notifications with the logging.

The component will have a ShowMessage routine that will use an enum with the value (Info, Success, Warning, Error). Each of those enums will call the relevant log and toast message.

### Component:

Code Guide insert the component with a reference to the class, so it can be manipulated.

<PopUpAndLogNotification NotificationMessage="Message" NotificationTitle="" @ref="@PopUpRef" />

The message is what is displayed, and can be overwritten. If the title is left then it uses the message type. In the code you display the message using the ref:

PopUpRef.ShowNotification(NotificationType.*type,”message”,”title”*)

The type is as above message is what is to be displayed, title is optional.

## Messages that require acknowledgment

These have fallen outr of favour but could be used for errors. The component is added to the page and can be called with a ref and then the ref controls the call like above. It is essentially like a toast message but with an ok/cancel

## Messages that require a response

These are used when we need the user to confirm. For example a delete. The component is added to the razor file with a ref. A call via the ref can overwrite the question and acknowledgement buttons. The component merely sorts out the defaults.

### Component:

Code example:

<ConfirmModal @ref="DeleteConfirmation" ConfirmationChanged="ConfirmDelete\_Click" ConfirmationMessage="Are you sure you want to delete?") ConfirmButtonText="Delete" CancelButtonText="Cancel" />

Here the call sets the call back routine that accepts a true/false which is confirmed or not.

To call the modal you use the ref set and any variables you want to change (like the item name you want to delete):

DeleteConfirmation.ShowModal("Delete confirmation", $"Are you sure you want to delete: {*Item*}?");

If the user selects the confirmation button the ConfirmedClick routine is called with true, otherwise false.

# Attribute Lookup Page

The Attributes are displayed in a grid. If the attribute is selected the variations of the attribute are then displayed as a grid.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Attribute Name | Varieties | S/Order | WooSync | Notes | [New] |
| Attrib1 | Var1,var2,var3 | 0 | Yes | Some notes | [Edit][Delete] |
| Attrib2 | Var4,var5 | 1 | No | Some other notes | [Edit][Delete] |

Varieties

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Variety Type | UoM | Symbol | FGColour | BGColour | S/Order | WooSync | Notes | [New] |
| Var1 | kg |  |  |  | 0 | Yes |  | [Edit][Delete] |
| Var2 | kg |  |  |  | 1 | No |  | [Edit][Delete] |
| Var3 | g |  |  |  | 2 | Yes |  | [Edit][Delete] |

As an attribute is select the varieties are displayed and can be edited? The attributes are edited in a modal since varieties can be added to them. The varieties table can only be edited in line. A new variety can be added via the modal Edit of the attribute name.

# Common Classes

### Class GridSettings

Used so all grid settings are in a single class.

public int PageSize = 15;

public string customFilterValue { get; set; } = string.Empty;

public bool IsNarrow { get; set; }= true;

public bool IsFilterable { get; set; }= false;

protected ConfirmModal DeleteConfirmation { get; set; }

public PopUpAndLogNotification PopUpRef { get; set; }

### Class WooLinkedGridSettings

Inherits from GridSettings and adds:

public bool WooIsActive { get; set; } = false;  
public ConfirmModal AddToWooConfirmation { get; set; }  
public ConfirmModal DeleteWooItemConfirmation { get; set; }

# Generic UI Linked Repositories

## IGridViewRepository

### Grid variables

public GridSettings \_GridSettings {get; set;}

### Generic Grid Classes

##### Get Entity By Id Async

Does a generic GetByIdAsync and returns the result  
Logic: Using the AppUnitOfWork.repostitory<TEntity> GetByIdAsync the object passed in?

|  |  |
| --- | --- |
| Parameters: | **Object** Id: Id to search for |
| Returns: | **TEntity** item found or null if not |

##### Find First By Async

Search for the Entity using a predicate.  
Logic: Using the AppUnitOfWork.repostitory<TEntity> FindFirstByAsync using the predicate object passed in?

|  |  |
| --- | --- |
| Parameters: | **Expression<Func<TEntity, bool>>** predicate: LINQ predicate to search for |
| Returns: | **TEntity** entity found or zero |

##### Insert View Row Async

Insert the new Entity to the database  
Logic: Assumes that the entity does not exists and adds it, should be preceded by FindFirstByAsync?

|  |  |
| --- | --- |
| Parameters: | **TEntity** newEntity: new entity to be saved  **String** newEntityDescription: Description of the entity to be added. Used for notifications. |
| Returns: | **TEntity** saved entity. |

##### Delete View Row By Id Async

Delete the source Entity from the database  
Logic: Uses Delete By Id to delete

|  |  |
| --- | --- |
| Parameters: | **object** sourceEntityId: Id to be deleted (uses GetNyId to find the entity)  **String** deletedEntityDescription: Description of the entity to be deleted. Used for notifications. |
| Returns: | **int** >0 for deleted, Error if error. |

##### Update View Row Async

Update the source Entity in the database  
Logic: Assumes the entity was originally got from the database and is current

|  |  |
| --- | --- |
| Parameters: | **object** sourceEntity: Entity to be updated  **String** updatedEntityDescription: Description of the entity to be updated. Used for notifications. |
| Returns: | **Guid** of item updated or Guid.Empty in not. |

### Recommended Overwritable Grid Classes

##### New View Entity Default Setter

Used to create a new entity when the add item button is pressed, this needs to be overwritten at a class specific level.  
Logic: return null, as we have no idea what will make the generic class valid.

|  |  |
| --- | --- |
| Parameters: | **TEntity** newEntity: blank entity to be initialised |
| Returns: | **null**. |

##### Is Duplicate Async

See if entity a duplicate, this needs to be overwritten at a class specific level.  
**Logic**: return false, as we have no idea what will make the generic class a duplicate.

|  |  |
| --- | --- |
| Parameters: | **TEntity** sourceEntity: Entity to be checked |
| Returns: | **Bool** true if a duplicate false if not. |

##### Is Valid

See if entity valid, this needs to be overwritten at a class specific level.  
**Logic**: return true, as we have no idea what will make the generic class valid.

|  |  |
| --- | --- |
| Parameters: | **TEntity** sourceEntity: Entity to be checked |
| Returns: | **Bool** true if a valid false if not. |

## IWooLinkedView

There are a number of grids/views that will display woo linked items. This lends it self to creating classes and interfaces that do that are used to display (view) the data and perform the CRUD tasks.

### Display Classes

Included in the display call are the common elements that are use in each grid:

// Interface Stuff used for the grid -that is used by the interface

These values are essentially a generic class used in setting up the Grid and support it. For any page that uses a grid this class can be used.

### Woo Linked View

Each Grid should have a view model, that inherits the main data from the data.models (TEntity) but adds items that are needed to either include as part of the woo support or to display on the gird, like Foreground and background colours in a single column.

### CRUD

Using the Woo Linked view (TViewEntity). The crud operations for the grid that need to be

async Task<List<TEntity>> GetAllAsync(TEntity)

async Task<List<TEntityWooMap> GetWooMappedItemAsync(Guid wooTEntityID)

TViewEntity MapItemToWooItemAsync(TEntity)

async Task OnRowInsertedAsync(TVeiwEntity)

OnNewItemDefaultSetterAsync(TViewEntity)

// used for the grid new item

async Task<int> UpdateItemAsync (TViewEntity) => private?

async Task<int> UpdateWooMappingAsync(TViewEntity)

IsDuplicate(TEntity)

IsValid(TEntity)

async Task OnRowUpdatedAsync(TViewEntity)

OnRowRemovingAsync - > not sure about this

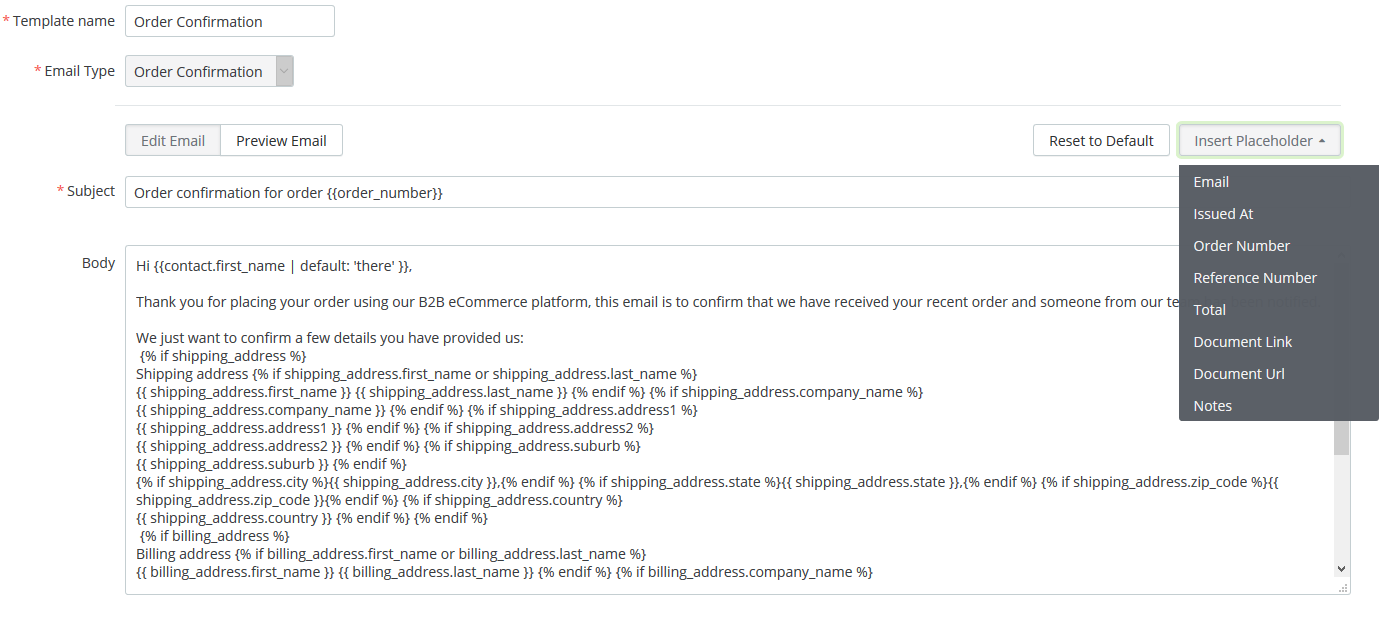
async Task ConfirmDeleteAsync(bool deleteConfirmed)

async Task<int> DoGroupActionAsync(ItemView)

Each of these will need to use the PopUpRef to log and the delete ref, so that will need to be passed in.

Woo Link?

Notifications:



Settings:

