Ucommerce Master Class

Deep Dive Handouts

Table of Contents

Expertise 1010: Querying

Intro

LINQ to Ucommerce provides rich capabilities to query the data stores of Ucommerce, but with great power comes great responsibility. This exercise introduces the three levels of data APIs.

Relevant APIs

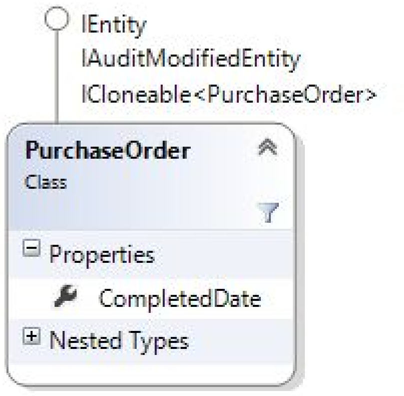
```
UCommerce.Infrastructure
      ObjectFactory.Resolve<T>

UCommerce.EntitiesV2
      IRepository<T>
            .Select()
      ISessionProvider
            .GetSession()

UCommerce.EntitiesV2
      Product
      PurchaseOrder
            .CreatedDate

NHibernate.Linq
      EagerFetchingExtensionMethods.Fetch()
      EagerFetchingExtensionMethods.FetchMany()
```

Quyering Excercises

Query for orders created after a certain date.

Intro

Once a customer has placed an order, the checkout pipeline will be executed. One of the tasks executed in that pipeline will set the CompletedDate property on the order so CompletedDate can be assumed the exact time the customer has checked out.

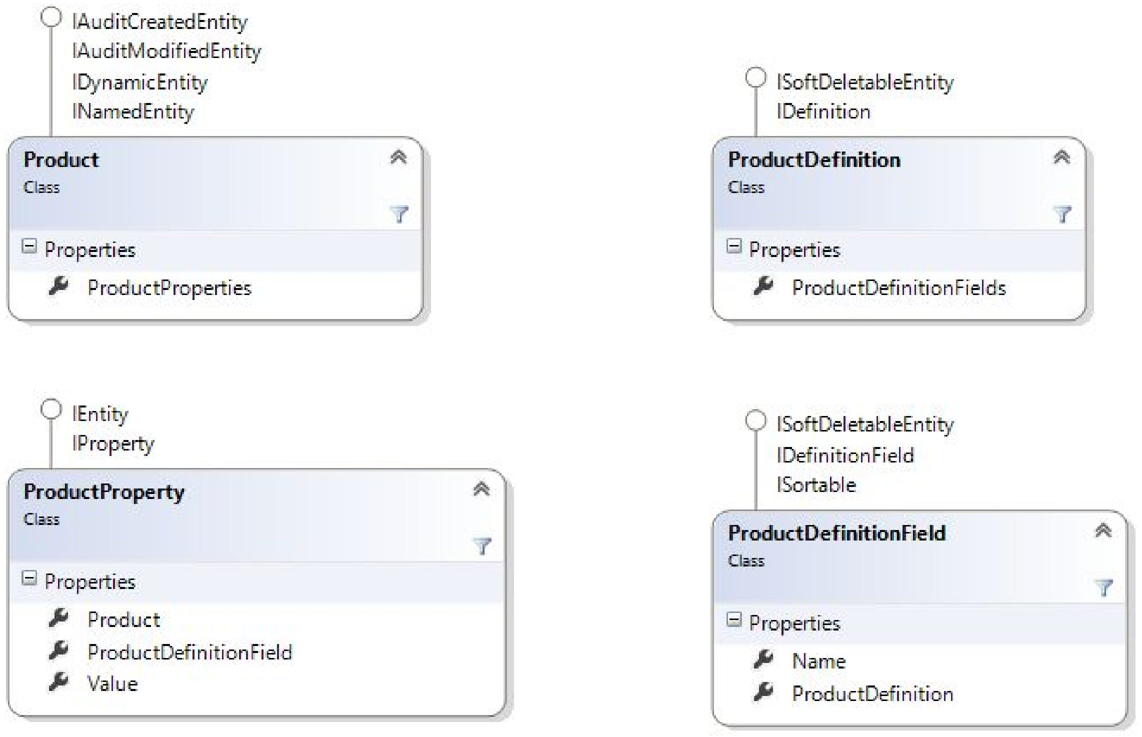


Query for products promoted to the homepage

- ProductProperties
- ProductDefinitionField
- “ShowOnHomepage”

Intro

Each product is based on a definition that once saved through the backend will have a list of properties persisted. Properties will be created based on the DefinitionFields for the Definition configured on the Product. Properties will usually reflect attributes for a product like the size or color of a shirt. Properties can also be used to create more flexible websites, like promoting a product to be shown on the homepage.

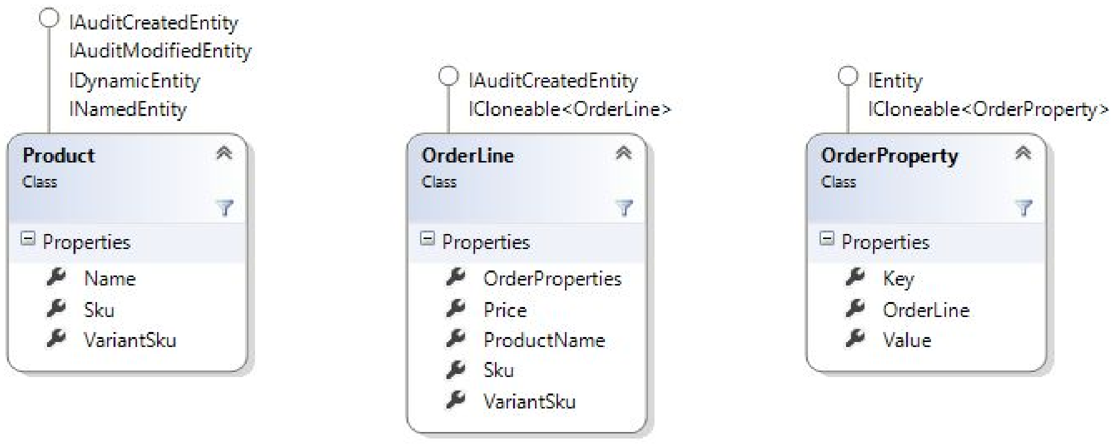


Join product to order line on Sku and VariantSku

- Anonymous type for join { orderline.Sku, orderline.VariantSku }

Intro

The catalog foundation and transaction foundation of Ucommerce is disconnected from each other. Once you add a product to the basket, you do not store the actual product but information that identifies what the product you are purchasing is. That is why an Orderline has Sku and VariantSku along with Product name and pricing information. You also have a collection of properties available you can use to store additional information regarding the product.
As an effect of the disconnected systems, you cannot navigate to the product from the orderline. If you wish to access it anyways, you can use Join with LINQ to query the product associated with the orderline.



Expertise 1015: Tuning your Queries

Intro

With LINQ to UCommerce comes great capabilities to query the database for data. The downside to this is that a lot of magic is happening behind the scenes. If you're querying products, why are the variants available? How much data is loaded?

N+1

- Query products
- Foreach over all products
- Access ParentProduct property in foreach
- Observe behavior in SQL Profiler
- How many queries do you see?

Eager Loading to Avoid N+1

- Use Fetch on ParentProduct property on product to tell NHibernate to initialize
- How many queries do you see?

Cartesian products

- Use Fetch on Product to grab Variants and ProductRelations
- How many queries do you see?
- How does the result set look like?

Large Cartesian Products Avoided

- Set up three queries to load product and variants
- Use ToFuture on LINQ queries to defer execution
- Execute one query
- How many batches are executed in SQL Server (use the tuning template in profiler)
- How does the result set look like?

Total Control of SQL with HQL

- Use ISessionProvider to create a query
- Use keyword join, outer join, fetch to control SQL
- Watch the resulting queries in SQL Profiler (use the tuning template in profiler)

Expertise 1020: Override CatalogContext

Intro

CatalogContext determines which store, catalog, and price group the customer is using at any given time. In this exercise, we will override the active catalog based on whether the customer is logged in or not.

Relevant APIs

```
UCommerce.Runtime
      ICatalogContext
      CatalogContext
            .CurrentCatalogName

      EagerFetchingExtensionMethods.Fetch()
      EagerFetchingExtensionMethods.FetchMany()
```

Steps

New catalog "Private Catalog"

New category "My Private Category"

Add a few products to "My Private Category"

Add new class in business logic, "MyCatalogContext"

Override CurrentCatalogName property

Register MyCatalogContext in an App

Create a new folder in Apps

Remove existing components

Find existing catalog context in Configuration/Core.config

Copy in that component

Change type to use MyCatalogContext

Expertise 1030: Extending Pipelines

Intro

A typical part of ecommerce is integration. Whether it is products being synced in to the website or orders flowing back to your favorite 3rd party system, proper integration needs to be done. In this case we want to export the order. In this exercise we'll peek into one of the most core concepts of Ucommerce – Pipelines.

Relevant APIs

```
UCommerce.Pipelines
      IPipelineTask<T>

UCommerce.EntitiesV2
      PurchaseOrder
            OrderNumber
```

Steps

Create a new class for the pipeline task called "ExportOrderToErpSystem"

Register in new app called " ExportOrderToErpSystem " using a component

Hook into ToCompleted pipeline using partial-component

Use File I/O to write the OrderNumber into a text file.

```
<configuration>
    <components>
        <partial-component id="ToCompletedOrder">
            <parameters>
                <tasks>
                    <array>
    <value insert="last">${ToCompletedOrder.ExportOrderToErpSystem}</value>
                    </array>
                </tasks>
            </parameters>
        </partial-component>
    </components>
</configuration>
```

Expertise 1040: Custom Data Type

Intro

When you need to be able to control how fields on products and categories are edited, a custom data type is just the thing. In this exercise you will create a new editor to set up tax groups on a per product level to support differentiated tax per product.

Relevant APIs

```
UCommerce.Presentation.Web.Controls
      IControlFactory
      IControlAdapter

UCommerce.EntitiesV2
      DataType
      PriceGroup
      IRepository<T>
```

Steps

Create a new class called "PriceGroupFactory"

Implement IControlFactory

Register component in a new app called "Differential Tax"

Set up a new data type in Ucommerce back-end in settings

Pick the Price Group component

Create a new field on a product definition using the new data type

Edit a product with the

Expertise 1050: Override Tax Calculation

Intro

To support products with individual tax set we need to override ITaxService to make it take the product setting into account rather than the default tax info from the price group.

Relevant APIs

```
UCommerce.Catalog
      ITaxService
      TaxService

UCommerce.EntitiesV2
      Product
      PriceGroup

UCommerce.Extensions
      DynamicProperty<T>()
```

Steps

Set up a field with the PriceGroupControlFactory

Set a custom price group on a product

Inherit TaxService and override the CalculateTax method

Check the product for the field (take into account variants)

Load the price group based on the id stored in the field

Calculate tax based on the loaded price group