Blueprint41

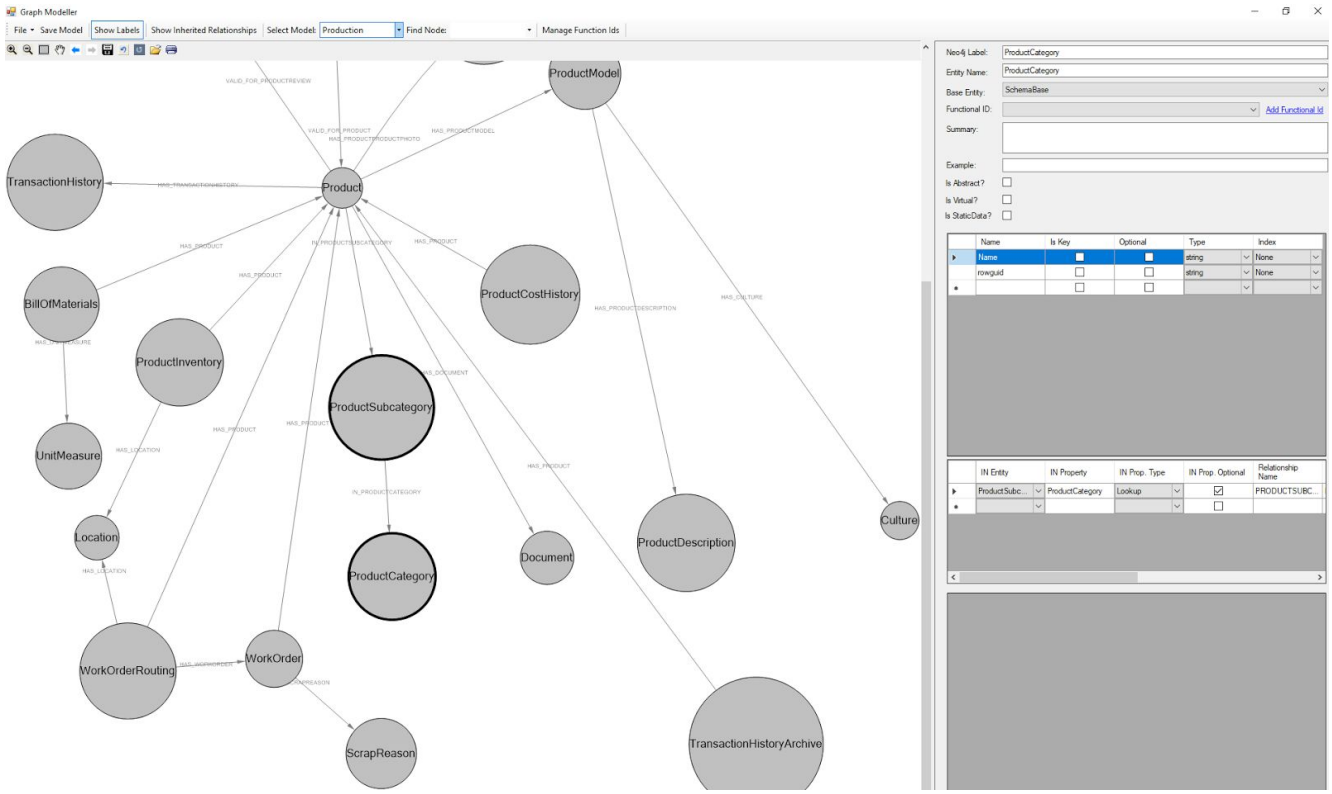
1. Overview

Blueprint41 is an Object Graph Mapper for CSharp and Neo4j. It has support for defining the object model as a schema. It support refactor scripts written in CSharp, which you can add to your project. When you run your program and the graph is of an older version, the upgrade script will automatically be executed against the graph. It also support generation of type-safe data objects, so you know at compile time if your code is compatible with the latest upgrades.

2. Project Structure

A. Blueprint41 [Solution]

a. Blueprint41
b. Blueprint41.Modeller
   - We can use the Graph Modeller to define the Model. See image below:



We have created sample AdventureWorks model located in this project under Xml folder (modeller.xml)

c. Blueprint41.Modeller.Compare [unavailable]
    - This is not part of the open source project. Please contact **xirqlz** for more information.

d. Blueprint41.Modeller.Laboratory [unavailable]
    - This is not part of the open source project. Please contact **xirqlz** for more information.

e. Blueprint41.Modeller.Schemas
f. Blueprint41Test

B. AdventureWorks [Solution]

- This is a sample project on how to generate entities, nodes and relationships based on the model that we define.

1) Datastore
2) Datastore.Generated

3. Getting Started

The following will guide you on how to generate your Neo4j entities, nodes and relationships. Here we use Adventureworks data model as an example.

Building the AdventureWorks Datastore

In AdventureWorks [Solution], we have already defined the model that will be used to generate AdventureWorks Entities, Nodes and Relationships.

Datastore [Project]

> AdventureWorks.cs

The Datastore project contains the Model where entities and relationships are initialized.

See #region Entities to see examples on how to initialize Adventureworks entities.

See #region Relationships to see examples on how to initialize Adventureworks relationships.

See #region Testing to see an example on how to initialize **AdventureWorks - ProductSubcategory.**

Here we used blueprint41 functionalIds (see https://github.com/xirqlz/functionalid).

Functional Id's can be generated via a procedure on Neo4j (version 3+). The database is used to persist the Functional Id state.

```
FunctionalIds.Default = FunctionalIds.New("Shared", "0", IdFormat.Numeric, 0);
```

The ff shows as an example for initializing **AdventureWorks - ProductSubcategory** entities:

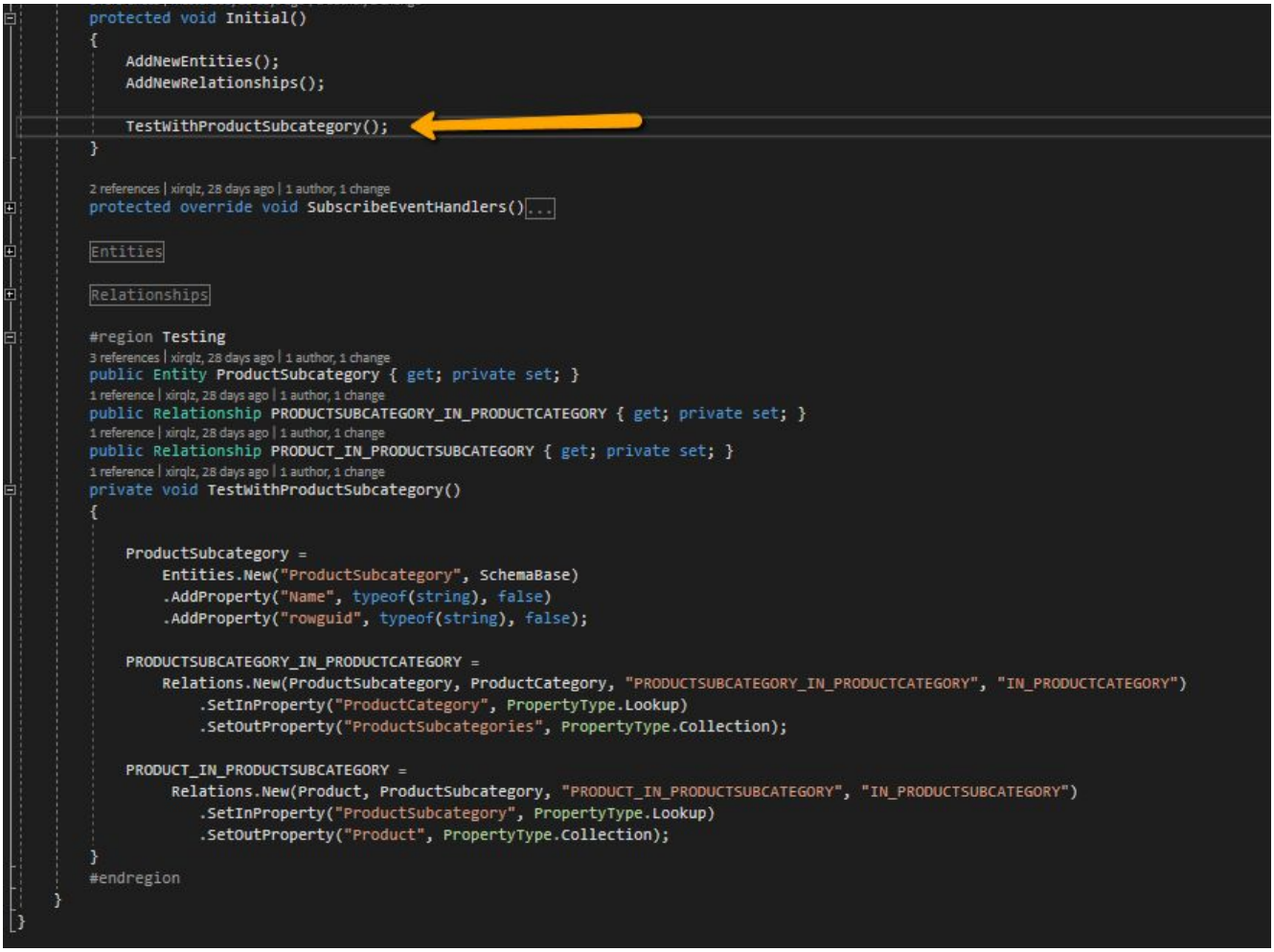
```
        public Entity Neo4jBase { get; private set; }
        public Entity SchemaBase { get; private set; }
        public Entity ProductCategory { get; private set; }
        public Entity ProductSubcategory { get; private set; }
        ...
```

```
Neo4jBase =
        Entities.New("Neo4jBase")
        .Abstract(true)
        .Virtual(true)
        .AddProperty("Uid", typeof(string), false, IndexType.Unique)
        .SetKey("Uid", true);

    SchemaBase =
        Entities.New("SchemaBase", Neo4jBase)
        .Abstract(true)
        .Virtual(true)
        .AddProperty("ModifiedDate", typeof(DateTime), false);

    ProductCategory =
        Entities.New("ProductCategory", SchemaBase)
        .AddProperty("Name", typeof(string), false)
        .AddProperty("rowguid", typeof(string), false);

    ProductSubcategory =
        Entities.New("ProductSubcategory", SchemaBase)
        .AddProperty("Name", typeof(string), false)
        .AddProperty("rowguid", typeof(string), false);
```

The ff shows as an example for **AdventureWorks - ProductSubcategory** relationships:

```
public Relationship PRODUCTSUBCATEGORY_IN_PRODUCTCATEGORY { get; private set; }

public Relationship PRODUCT_IN_PRODUCTSUBCATEGORY { get; private set; }

...


PRODUCTSUBCATEGORY_IN_PRODUCTCATEGORY =

        Relations.New(ProductSubcategory, ProductCategory,
                        "PRODUCTSUBCATEGORY_IN_PRODUCTCATEGORY",
                        "IN_PRODUCTCATEGORY")

        .SetInProperty("ProductCategory", PropertyType.Lookup)

        .SetOutProperty("ProductSubcategories", PropertyType.Collection);


PRODUCT_IN_PRODUCTSUBCATEGORY =

            Relations.New(Product, ProductSubcategory, "PRODUCT_IN_PRODUCTSUBCATEGORY",
        "IN_PRODUCTSUBCATEGORY")

        .SetInProperty("ProductSubcategory", PropertyType.Lookup)

        .SetOutProperty("Product", PropertyType.Collection);
```

To test ProductSubcategory, simply call TestWithProductSubcategory() in Initial() method and rebuild the project.



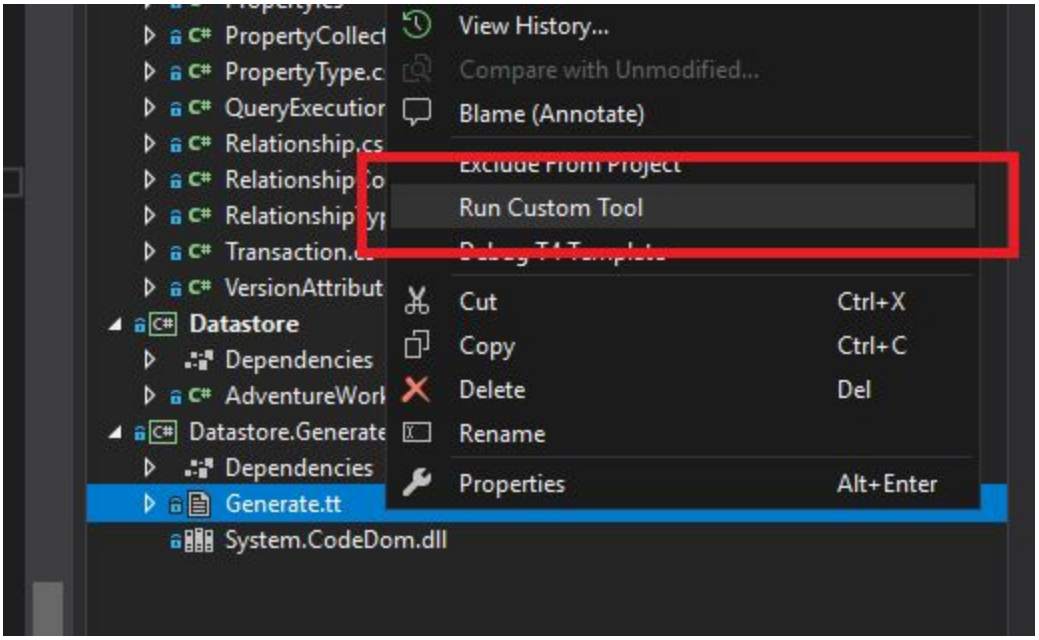
```
        protected void Initial()
        {
            AddNewEntities();
            AddNewRelationships();

            TestWithProductSubcategory();
        }

        2 references | xirqlz, 28 days ago | 1 author, 1 change
        protected override void SubscribeEventHandlers()...

        Entities

        Relationships

        #region Testing
        3 references | xirqlz, 28 days ago | 1 author, 1 change
        public Entity ProductSubcategory { get; private set; }
        1 reference | xirqlz, 28 days ago | 1 author, 1 change
        public Relationship PRODUCTSUBCATEGORY_IN_PRODUCTCATEGORY { get; private set; }
        1 reference | xirqlz, 28 days ago | 1 author, 1 change
        public Relationship PRODUCT_IN_PRODUCTSUBCATEGORY { get; private set; }
        1 reference | xirqlz, 28 days ago | 1 author, 1 change
        private void TestWithProductSubcategory()
        {

            ProductSubcategory =
                Entities.New("ProductSubcategory", SchemaBase)
                .AddProperty("Name", typeof(string), false)
                .AddProperty("rowguid", typeof(string), false);

            PRODUCTSUBCATEGORY_IN_PRODUCTCATEGORY =
                Relations.New(ProductSubcategory, ProductCategory, "PRODUCTSUBCATEGORY_IN_PRODUCTCATEGORY", "IN_PRODUCTCATEGORY")
                    .SetInProperty("ProductCategory", PropertyType.Lookup)
                    .SetOutProperty("ProductSubcategories", PropertyType.Collection);

            PRODUCT_IN_PRODUCTSUBCATEGORY =
                 Relations.New(Product, ProductSubcategory, "PRODUCT_IN_PRODUCTSUBCATEGORY", "IN_PRODUCTSUBCATEGORY")
                    .SetInProperty("ProductSubcategory", PropertyType.Lookup)
                    .SetOutProperty("Product", PropertyType.Collection);
        }
        #endregion
    }
}
```

Generating AdventureWorks Entities, Nodes and Relationships

Datastore.Generated [Project]

> Generate.tt

The Datastore.Generated project contains all generated files when Generator.tt is ran. Simply right click on Generate.tt and click “Run Custom Tool”.



Generate.txt should show “Success!”

The following files will then be generated based from the Datastore Model that we updated.

Datastore.Generated [Project]

> Entites [Folder]

> ProductSubcategory.cs

> Nodes [Folder]

> ProductSubcategoryNode.cs

> Relationships [Folder]

> PRODUCT_IN_PRODUCTSUBCATEGORY.cs

> PRODUCTSUBCATEGORY_IN_PRODUCTCATEGORY.cs