DataGrid
Database Insert

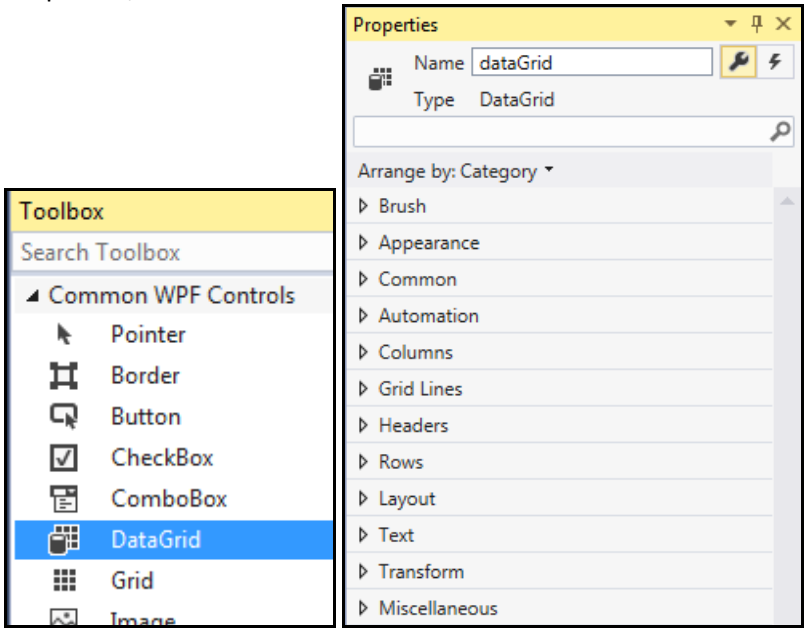
Inhalt

1 DataGrid

DataGrids werden hauptsächlich im Zusammenhang mit Datenbanktabellen verwendet. Sie zeigen einen Datensatz in einer Zeile an, die Spalten ergeben sich aus den Properties des Datensatzes.

1.1 DataGrid-Control

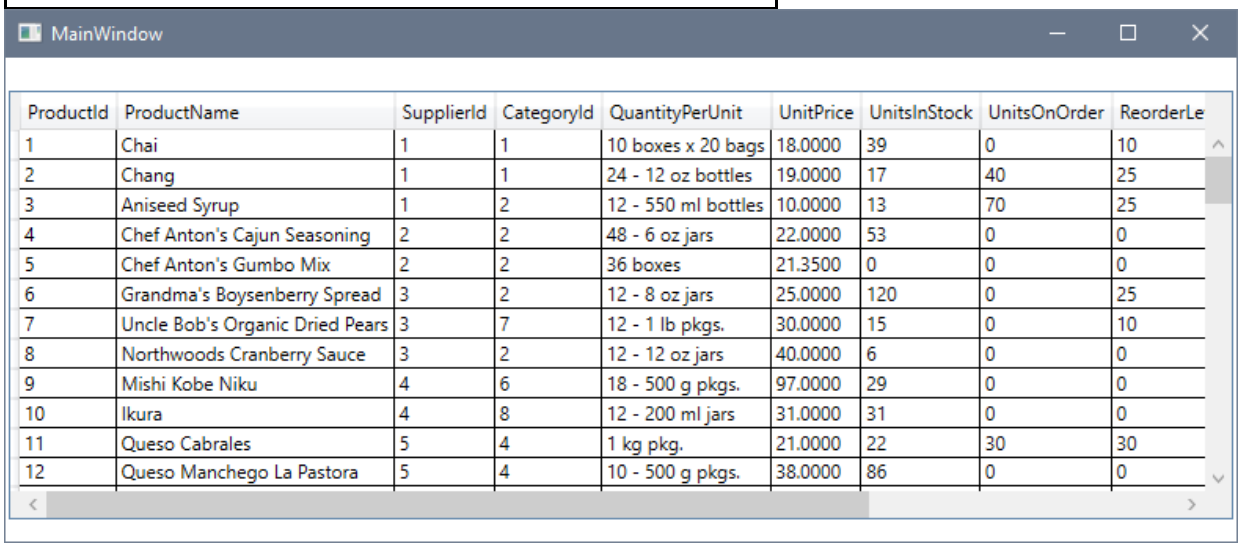
Das DataGrid findet man in der Toolbox und verwendet es wie jedes andere Control auch, d.h. es gibt jede Menge Properties, die man einstellen kann.



1.2 ItemsSource

Die einfachste Variante der Verwendung ist, dass man einfach eine Liste von Objekten auf die Property ItemsSource zuweist.
Für jede Property der Objekte wird dann eine Spalte im Grid erzeugt.

```
var db = new NorthwindContext();
grdProducts.ItemsSource = db.Products.ToList();
```



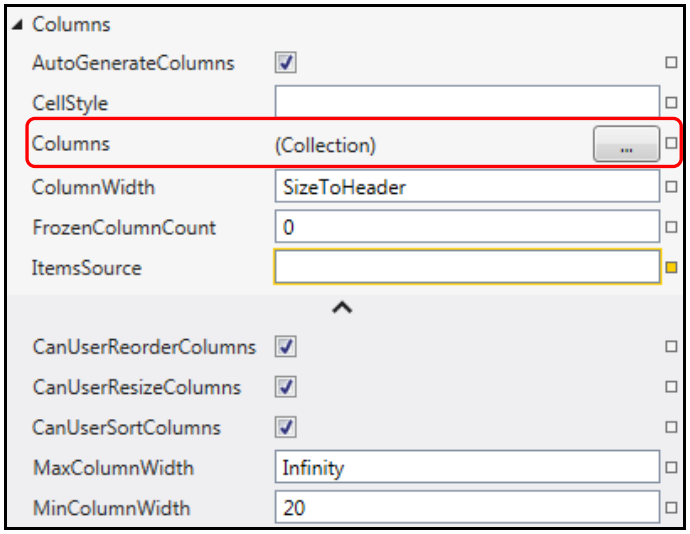
ProductId	ProductName	SupplierId	CategoryId	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLe
1	Chai	1	1	10 boxes x 20 bags	18.0000	39	0	10
2	Chang	1	1	24 - 12 oz bottles	19.0000	17	40	25
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.0000	13	70	25
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.0000	53	0	0
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.3500	0	0	0
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.0000	120	0	25
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30.0000	15	0	10
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40.0000	6	0	0
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97.0000	29	0	0
10	Ikura	4	8	12 - 200 ml jars	31.0000	31	0	0
11	Queso Cabrales	5	4	1 kg pkg.	21.0000	22	30	30
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	38.0000	86	0	0

Das liegt an der Property **`AutoGenerateColumns`** von DataGrid, die per Default **`true`** ist.

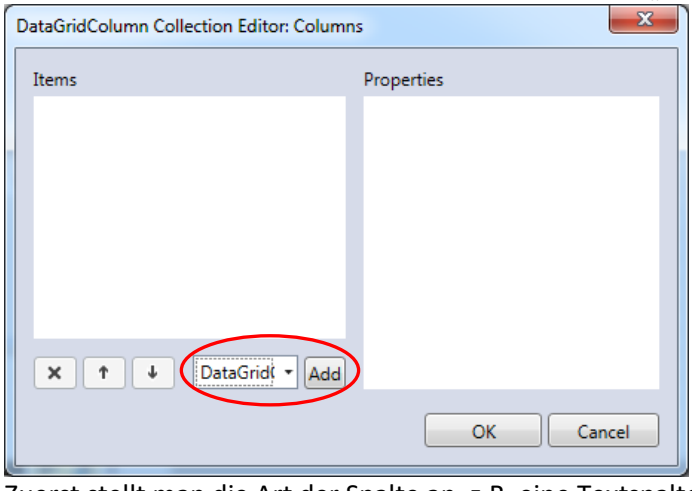
1.3 Konfiguration

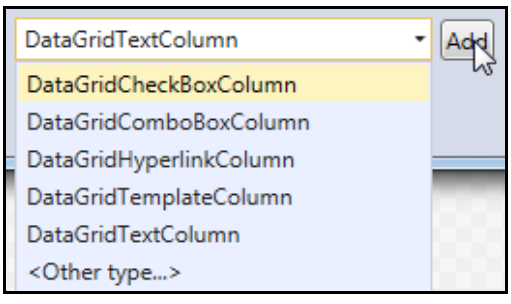
Man kann das aber auch selber konfigurieren.

1.3.1 Properties-Fenster



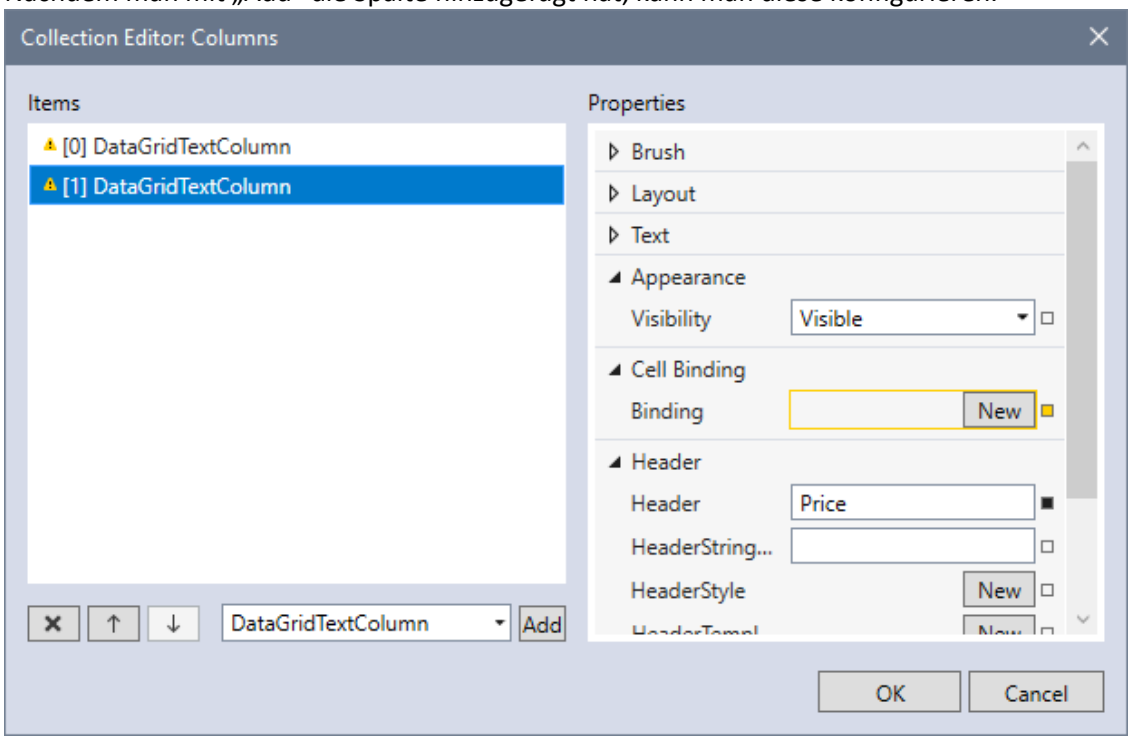
Die Columns kann man durch Klick auf den Button bei „Columns“ einstellen. Diese ist zunächst noch leer:



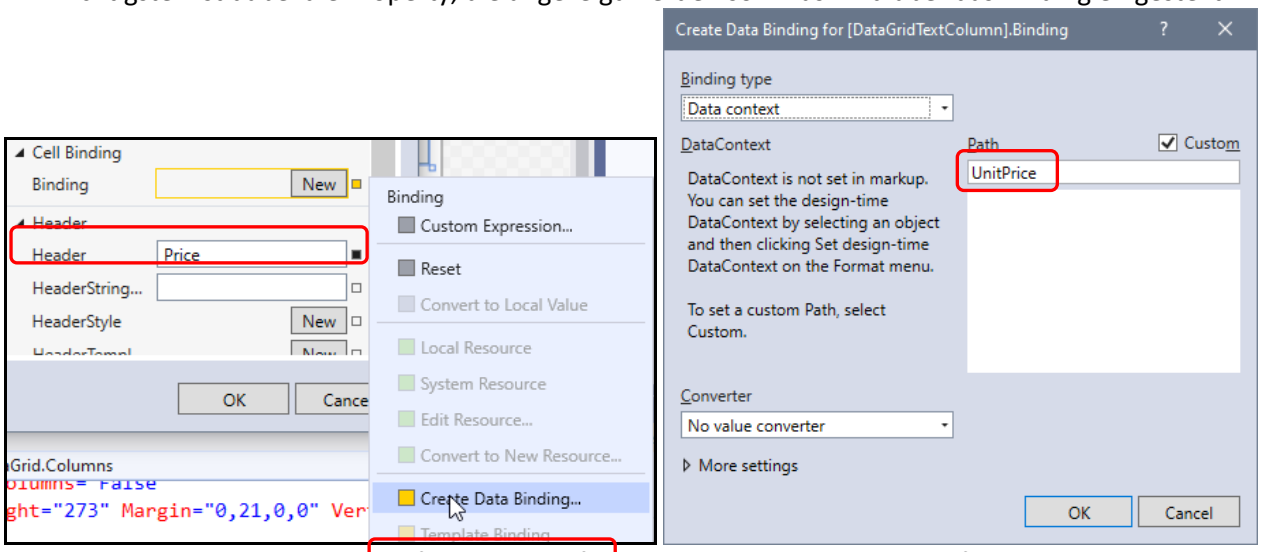


Zuerst stellt man die Art der Spalte an, z.B. eine Textspalte.

Nachdem man mit „Add“ die Spalte hinzugefügt hat, kann man diese konfigurieren.



Am wichtigsten ist dabei die Property, die angezeigt werden soll. Das wird über das Binding eingestellt:



Die Spalten kann man ziemlich umfangreich konfigurieren, z.B. Spaltenüberschrift, Spaltenbreite….. Das ist im Dialog praktisch selbsterkärend.

1.3.2 XAML

Die Einstellungen werden natürlich wieder als XAML gespeichert. Die weiteren Spalten kann man dann vermutlich schneller direkt im XAML konfigurieren. Wichtig ist, dass man für das DataGrid das automatische Erzeugen der Spalten durch **`AutoGenerateColumns="False"`** verhindert:

```
<DataGrid Name="grdProducts" AutoGenerateColumns="False"
          HorizontalAlignment="Center" Height="273" Margin="0,21,0,0" VerticalAlignm
  <DataGrid.Columns>
    <DataGridTextColumn Binding="{Binding ProductName}" Header="Name" Width="150"/>
    <DataGridTextColumn Binding="{Binding UnitPrice}" Header="Price"/>
    <DataGridTextColumn Binding="{Binding UnitsInStock}" Header="Stock"/>
  </DataGrid.Columns>
</DataGrid>
```

Damit sieht es so aus:

Name	Price	Stock
Chai	18.0000	39
Chang	19.0000	17
Aniseed Syrup	10.0000	13
Chef Anton's Cajun Season	22.0000	53
Chef Anton's Gumbo Mix	21.3500	0
Grandma's Boysenberry Sp	25.0000	120
Uncle Bob's Organic Dried	30.0000	15
Northwoods Cranberry Sau	40.0000	6
Mishi Kobe Niku	97.0000	29
Ikura	31.0000	31
Queso Cabrales	21.0000	22
Queso Manchego La Pastoı	38.0000	86
Konbu	6.0000	24

Man kann auch Daten aus referenzierten Tabellen anzeigen. Dann darf man aber auf Include beim Select nicht vergessen!

```
<DataGridTextColumn Binding="{Binding Category.CategoryName}" Header="Category"/>
```

```
grdProducts.ItemsSource = db.Products
  .Include(x => x.Category)
  .ToList();
```

1.3.3 Custom Klasse

Eine andere und vermutlich einfachere Möglichkeit der Konfiguration ist, eine eigene Klasse zu erzeugen, die genau die Properties enthält, die angzeigt werden sollen.
Das ist vor allem dann sinnvoll, wenn man Daten aus mehreren Tabellen anzeigen will.

```
public class ProductRow
{
  public string Name { get; set; }
  public double Price { get; set; }
  public string Category { get; set; }
}
```

```
grdProductsOther.ItemsSource = db.Products
  .Select(x => new ProductRow
  {
    Name = x.ProductName,
    Price = (double)x.UnitPrice,
    Category = x.Category.CategoryName
  })
  .ToList();
```

In diesem Fall kann man dann wieder AutoGenerateColumns auf true belassen.

Name	Price	Category
Chai	18	Beverages
Chang	19	Beverages
Aniseed Syrup	10	Condiments
Chef Anton's Cajun Seasoning	22	Condiments
Chef Anton's Gumbo Mix	21.35	Condiments
Grandma's Boysenberry Spread	25	Condiments
Uncle Bob's Organic Dried Pears	30	Produce
Northwoods Cranberry Sauce	40	Condiments

1.4 Events

Das DataGrid löst viele Events aus. Stellvertretend sollen SelectionChanged besprochen werden.

1.4.1 SelectionChanged

Damit hat man Zugriff auf den gerade ausgewählten Datensatz.

```
<DataGrid Name="grdProducts" AutoGenerateColumns="False"
          HorizontalAlignment="Center" Height="273" Margin="0
          SelectionChanged="GrdProducts_SelectionChanged">
```

Über die Property **`SelectedItem`** hat man eine Referenz auf das Objekt:

```
private void GrdProducts_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
  var product = grdProducts.SelectedItem as Product;
  System.Console.WriteLine($"SelectionChanged --> {product.ProductName}");
}
```

2 DB Insert

Mit Entity Framework kann man nicht nur aus der Datenbank lesen, sondern auch schreiben, aktualisieren und löschen. Vorerst soll nur Insert besprochen werden.

2.1 SaveChanges

Änderungen in der Datenbank führt man durch, indem man die C#-Objekte (die sogenannten Entities) verändert bzw. erzeugt, und diese dann mit **`SaveChanges()`** in die Datenbank speichert.
Wichtig: Es gibt keine separaten Methoden wie InsertRecords() o.ä., egal welche Änderung man durchführt heißt die Methode immer SaveChanges()!

2.2 Insert

Ein Insert sieht so aus:

```
var category = new Category { CategoryName = "abc", Description = "aa bb cc" };
db.Categories.Add(category);
db.SaveChanges();
```

Die Vorgangsweise dabei ist also folgende:

1. **neues C#-Objekt** erzeugen
2. dem zugehörigen **DbSet hinzufügen**
3. **`db.SaveChanges()`** aufrufen

Vor allem den zweiten Schritt (hinzufügen zum DbSet) darf man nicht vergessen. Ein Datensatz-Objekt ist ja ein ganz normales C#-Objekt (POCO), das nichts von einer Datenbank weiß. Erst das DbSet stellt das Bindeglied zw. Objekt und Datenbank her.