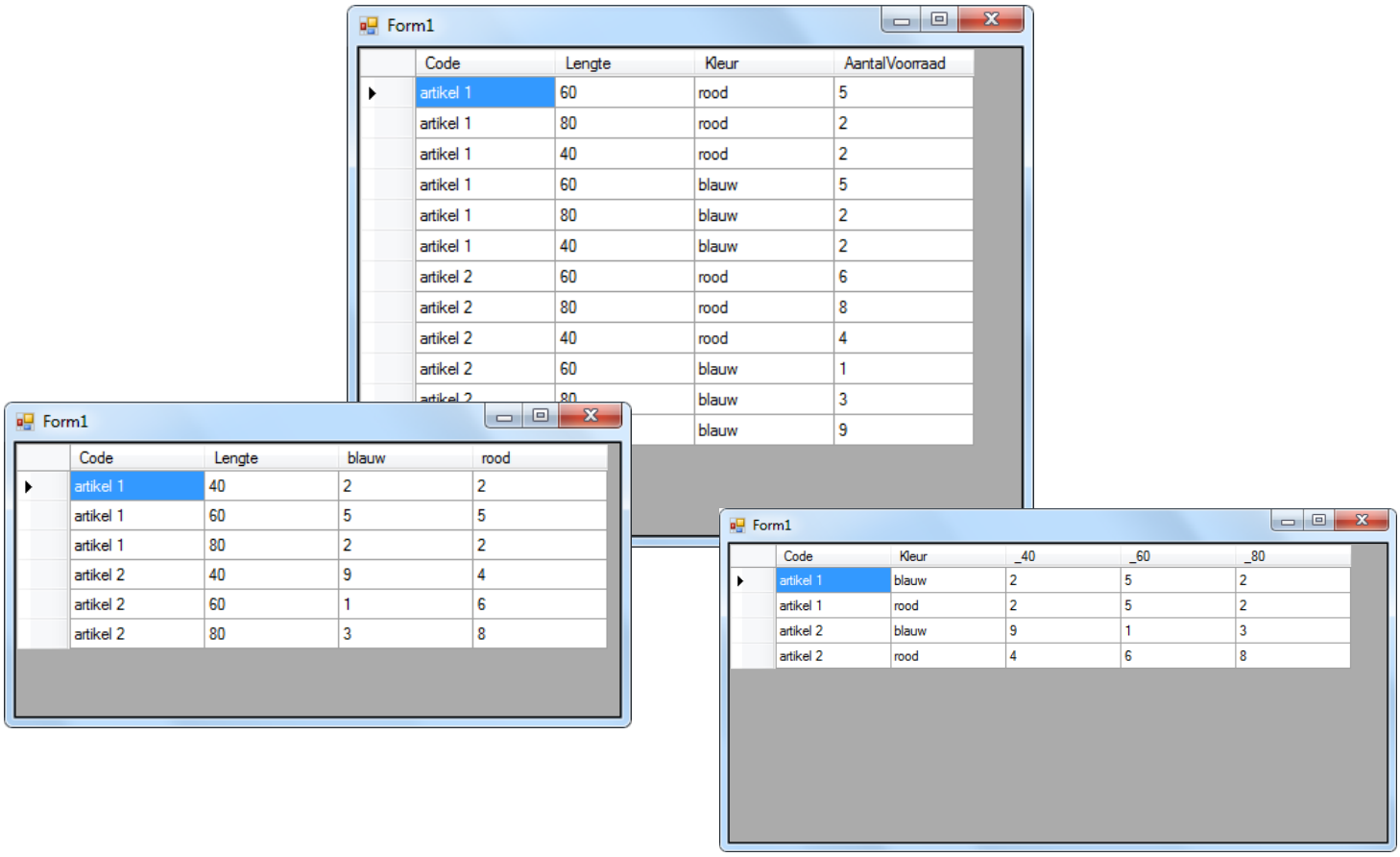
Maak een pivot uit een Generic.List

Introductie in extensions, reflection en code generation

Nivo: 400



Introductie

In bepaalde gevallen komt het voor dat je een Generic.List van een specifieke class hebt, die je als een Pivot wilt tonen (Pivot wordt ook wel genoemd crosstab of draaitabel).

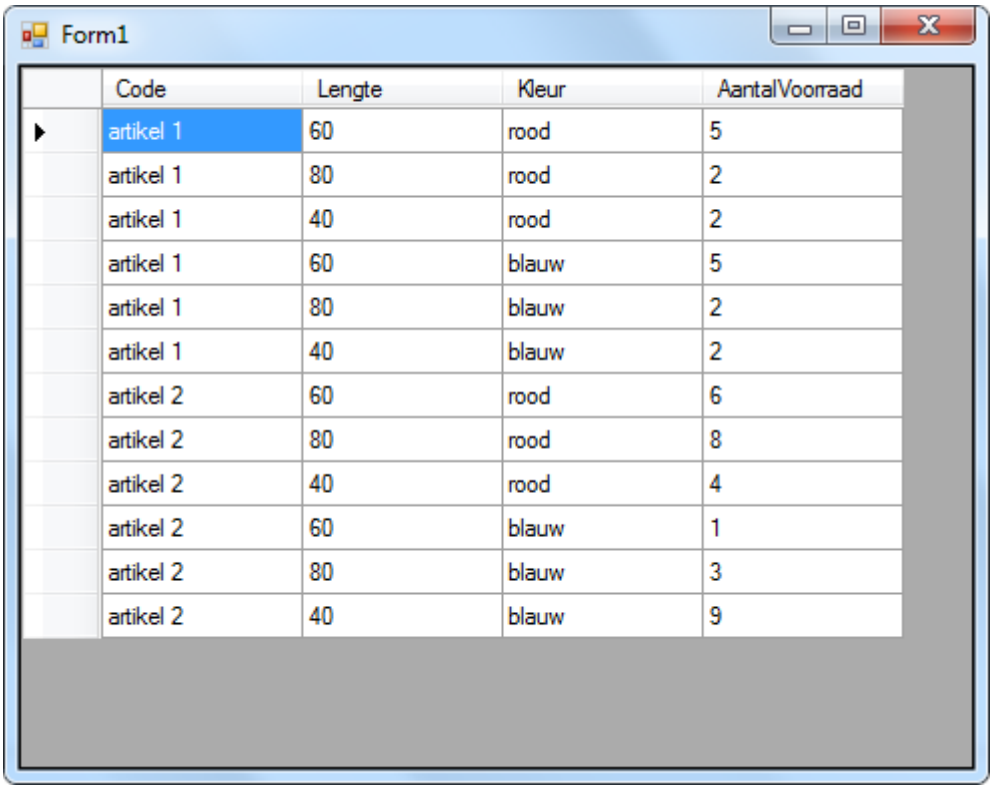
Hiervoor is een standaard oplossing mogelijk, die via een extension method, reflection en code injection elke Generic.List(of T) kan omtoveren tot een pivot.

De case

We beginnen met een simpele class:

```
Public Class Artikel
    Private pCode As String
    Private pLengte As Double
    Private pKleur As String
    Private pAantalVoorraad As Integer

    Public Sub New(ByVal code As String, ByVal lengte As Double, ByVal kleur As String, _
                   ByVal aantalVoorraad As Integer)
        pCode = code
        pLengte = lengte
        pKleur = kleur
        pAantalVoorraad = aantalVoorraad
    End Sub

    Public ReadOnly Property Code() As String
        Get
            Return pCode
        End Get
    End Property

    Public ReadOnly Property Lengte() As Double
        Get
            Return pLengte
        End Get
    End Property

    Public ReadOnly Property Kleur() As String
        Get
            Return pKleur
        End Get
    End Property

    Public ReadOnly Property AantalVoorraad() As Integer
        Get
            Return pAantalVoorraad
        End Get
    End Property
End Class
```

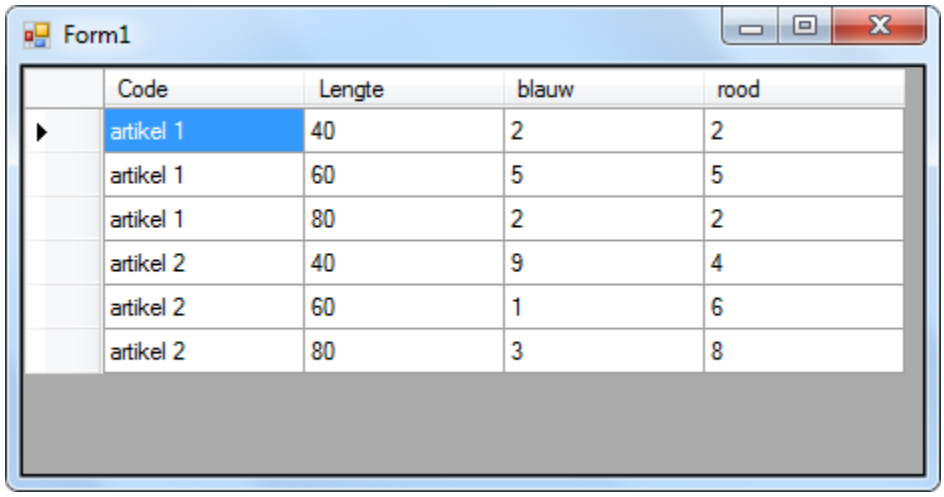
Wanneer we nu een form maken, met hierop een DataGridView dgArtikel, en deze vullen met een Generic.List(Of Artikel), krijgen we het volgende te zien:



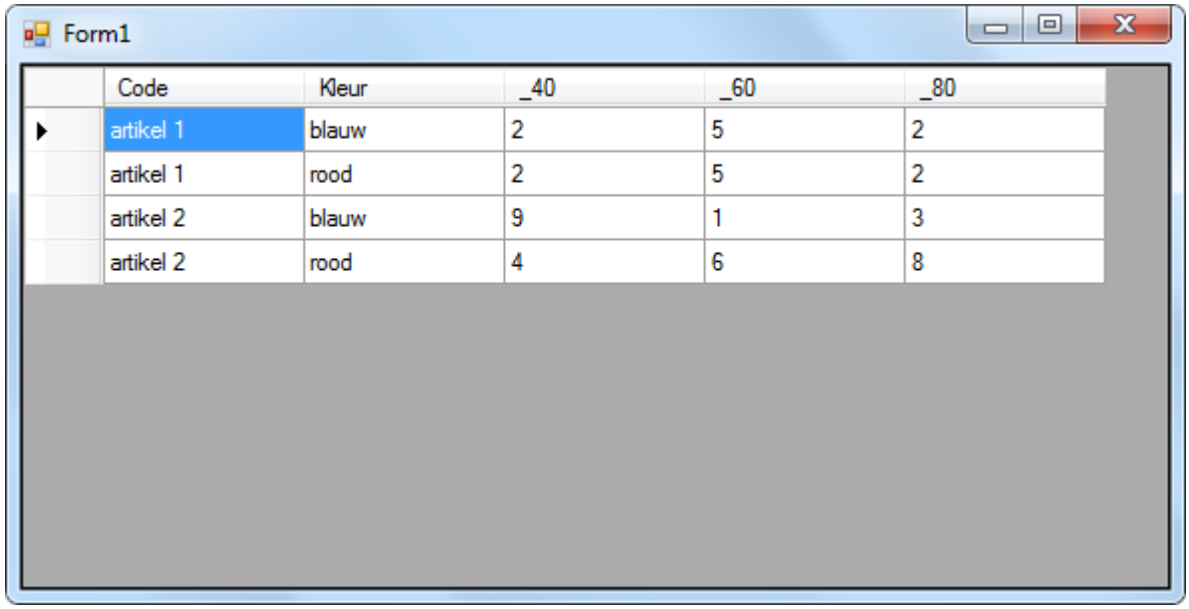
Form Code:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        Dim Artikelen As New Generic.List(Of Artikel)

        Artikelen.Add(New Artikel("artikel 1", 60, "rood", 5))
        Artikelen.Add(New Artikel("artikel 1", 80, "rood", 2))
        Artikelen.Add(New Artikel("artikel 1", 40, "rood", 2))
        Artikelen.Add(New Artikel("artikel 1", 60, "blauw", 5))
        Artikelen.Add(New Artikel("artikel 1", 80, "blauw", 2))
        Artikelen.Add(New Artikel("artikel 1", 40, "blauw", 2))
        Artikelen.Add(New Artikel("artikel 2", 60, "rood", 6))
        Artikelen.Add(New Artikel("artikel 2", 80, "rood", 8))
        Artikelen.Add(New Artikel("artikel 2", 40, "rood", 4))
        Artikelen.Add(New Artikel("artikel 2", 60, "blauw", 1))
        Artikelen.Add(New Artikel("artikel 2", 80, "blauw", 3))
        Artikelen.Add(New Artikel("artikel 2", 40, "blauw", 9))

        dgArtikelen.DataSource = Artikelen
    End Sub
End Class
```

Maar wat we willen zien is (gegroepeerd op kleur):



Of (gegroepeerd op lengte):



De Extension Method

We beginnen met het extenden van de class Generic.List(Of T), waarbij we een pivot functie toevoegen aan de class.

Om de Pivot te maken, hebben we de volgende gegevens nodig:
- De kolom(men) die getoond moeten worden (en waarop dus gesorteerd moet worden)
- De kolom(men) waarop de pivot waardes gesorteerd moeten worden
- De kolom waarop de pivot gemaakt wordt

De pivot functie zelf gaat wederom een Generic.List teruggeven, maar met een nog onbekende class type. We definieren de returnwaarde daarom als Generic.List(Of Object)

Hiervoor voegen we een Module toe aan het project, waarin de extensie gedefinieerd wordt:

```
Option Strict Off
Public Module PivotExtension

    <System.Runtime.CompilerServices.Extension()> _
    Public Function Pivot(Of T)(ByVal list As Generic.List(Of T), _
                                ByVal ShowFields As Generic.List(Of String), _
                                ByVal PivotOrderbyFields As Generic.List(Of String), _
                                ByVal PivotField As String) As Generic.List(Of Object)
        Return New Generic.List(Of Object)(list.ToArray)
    End Function

End Module
```

Het attribuut Extension geeft aan dat dit een extension Method is – dus een method die een bestaande class uitbreid zonder dat de class hiervoor inherited wordt. De eerste parameter van de functie is de class die uitgebreid wordt, de rest van de parameters zijn de parameters van de functie.
Voorlopig retourneren we het object waar de functie op uitgevoerd word als een Object list.

Om straks de data gesorteerd op kleur en samengevoegd op lengte te kunnen bekijken, veranderen we alvast de code in het form; de Show field zijn artikel en kleur, de PivotOrderBy field is Lengte, en de Pivot field is AantalVoorraad.

De code in het form wordt dan:

```
Public Class Form1

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
        Dim Artikelen As New Generic.List(Of Artikel)

        Artikelen.Add(New Artikel("artikel 1", 60, "rood", 5))
        Artikelen.Add(New Artikel("artikel 1", 80, "rood", 2))
        Artikelen.Add(New Artikel("artikel 1", 100, "rood", 2))
        Artikelen.Add(New Artikel("artikel 1", 60, "blauw", 5))
        Artikelen.Add(New Artikel("artikel 1", 80, "blauw", 2))
        Artikelen.Add(New Artikel("artikel 1", 100, "blauw", 2))
        Artikelen.Add(New Artikel("artikel 2", 60, "rood", 6))
        Artikelen.Add(New Artikel("artikel 2", 80, "rood", 8))
        Artikelen.Add(New Artikel("artikel 2", 100, "rood", 4))
        Artikelen.Add(New Artikel("artikel 2", 60, "blauw", 1))
        Artikelen.Add(New Artikel("artikel 2", 80, "blauw", 3))
        Artikelen.Add(New Artikel("artikel 2", 100, "blauw", 9))
```

```
        Dim ShowFields As New Generic.List(Of String)
        Dim PivotOrderByFields As New Generic.List(Of String)

        ShowFields.Add("Code")
        ShowFields.Add("Kleur")
        PivotOrderByFields.Add("Lengte")

        dgArtikelen.DataSource = Artikelen.Pivot(ShowColumnsFields, PivotOrderByFields, _
                                                 "AantalVoorraad")
    End Sub
End Class
```

De implementatie van een runtime gegenereerde class

De class die straks door de pivot functie teruggegeven wordt is nergens in het programma gedefinieerd. Deze class wordt dynamisch gegenereerd in een gegenereerde memory only assembly – deze gegenereerde assembly wordt dus nergens op schijf opgeslagen.

De tweede voorbereiding wordt dus het bouwen van een class generator, waarbij via reflection een complete assembly wordt opgebouwd.

Stap 1: Maak de module

```
    Private Function CreateAssembly(ByRef className As String) As ModuleBuilder
        className &= Guid.NewGuid.ToString.Replace("-", "_") ' create a unique name

        Dim an As New AssemblyName(className)
        Dim ad As AppDomain = AppDomain.CurrentDomain
        Dim ab As AssemblyBuilder = ad.DefineDynamicAssembly(an, AssemblyBuilderAccess.Run)

        Return ab.DefineDynamicModule(an.Name)
    End Function
```

Eerst maken we een unieke naam voor de module door de gewenste classname uit te breiden met een Guid.
(Deze uitbreiding wordt ook teruggegeven; className is ByRef, zodat de class straks hetzelfde heet als de module waarin deze gegenereerd is)
Vervolgens maken we een nieuwe assemblyname aan, vragen het huidige appdomain op, en maken daarin een nieuwe dynamische assembly aan, die alleen uitgevoerd kan worden (en dus niet opgeslagen)
In deze assembly maken we een module aan, die we vervolgens teruggeven als functie resultaat.
Note: de assembly, de module en de class hebben straks dus allemaal dezelfde naam. De enige reden hiervoor is dat dit het geheel wat simpeler maakt, en dat deze namen nooit gerefereerd zullen worden. Het maakt dus niet uit hoe ze heten, zolang de namen maar uniek zijnj voor hun soort.

De module kan nu simpel gecreerd worden door de volgende aanroep:

```
Dim ModuleBuilder As ModuleBuilder = CreateAssembly(ClassName)
```

Stap 2: Maak de class

```
    Private Function CreateClass(ByVal builder As ModuleBuilder, _
                                 ByVal className As String) As TypeBuilder
        Return builder.DefineType(className, TypeAttributes.Public Or TypeAttributes.Class)
    End Function
```

Hier wordt een public class geinitieerd in de zojuist gecreerde module.

De class kan nu duidelijk worden aangemaakt met de volgende aanroep:

```
Dim TypeBuilder As TypeBuilder = CreateClass(ModuleBuilder, ClassName)
```

Stap 3: Voeg properties toe aan de class

Om de gegevens aan de class toe te kunnen voegen hebben we 1 of meerdere properties nodig, welke we maken me (je raad het al) een Property Builder.
Maar, een propery is wat ingewikkelder; een property bestaat uit:
- Een storage field
- Een property definitie
- Een Get method
- Een Set method

Om een property te maken moeten dus alle 4 de elementen aangemaakt worden, en moeten de getter en de setter methods ook geimplementeerd worden:

```
    Private Sub CreateProperty(ByVal builder As TypeBuilder, ByVal propertyName As String, _
                               ByVal propertyType As Type)
        Dim FieldBuilder As FieldBuilder = _
                         builder.DefineField("p" & propertyName, propertyType, FieldAttributes.Private)
        Dim PropBuilder As PropertyBuilder = _
                  builder.DefineProperty(propertyName, PropertyAttributes.None, propertyType, Nothing)
        Dim getSetAttr As MethodAttributes = _
                 MethodAttributes.Public Or MethodAttributes.SpecialName Or MethodAttributes.HideBySig
        Dim PropGetAccessBuilder As MethodBuilder = _
                 builder.DefineMethod("Get" & propertyName, getSetAttr, propertyType, Type.EmptyTypes)
        Dim PropSetAccessBuilder As MethodBuilder = _
               builder.DefineMethod("Set" & propertyName, getSetAttr, Nothing, New Type() {propertyType})
        Dim propGetIL As ILGenerator = PropGetAccessBuilder.GetILGenerator()
        Dim propSetIL As ILGenerator = PropSetAccessBuilder.GetILGenerator()

        propGetIL.Emit(OpCodes.Ldarg_0)
        propGetIL.Emit(OpCodes.Ldfld, FieldBuilder)
        propGetIL.Emit(OpCodes.Ret)
        PropBuilder.SetGetMethod(PropGetAccessBuilder)

        propSetIL.Emit(OpCodes.Ldarg_0)
        propSetIL.Emit(OpCodes.Ldarg_1)
        propSetIL.Emit(OpCodes.Stfld, FieldBuilder)
        propSetIL.Emit(OpCodes.Ret)
        PropBuilder.SetSetMethod(PropSetAccessBuilder)
    End Sub
```

De aanroep bevat de type builder omdat de properties hieraan toegevoegd moeten worden, gevolgd door de naam en het type van de te genereren property.
Als eerste wordt er een FieldBuilder ingezet om het storage field te genereren. Ik maak hier het storage field onder dezelfde naam aan als de property, met een p als naam Prefix. Het storage field krijgt als type het meegegeven type bij de functie aanroep, en wordt als een private field gegenereerd

Vervolgens wordt de propery definitie gegenereerd met de juiste naam en type, zonder specifieke eigenschappen (daarmee dus public), en zonder property parameters (aangegeven door Nothing).

Hierna worden de specifieke attributen gemaakt die aan de getter en setter methods toegevoegd moeten worden. Zonder deze specifieke attributen werken de getter en setter niet op een property.

Hierna worden 2 method builders aangemaakt, een voor de getter en 1 voor de setter. De getter heeft een returnvalue die qua type hetzelfde is als de property, en heeft geen parameters, de setter heeft een parameter die qua type hetzelfde is als de property, en heeft geen return value.

Hierna worden de methods met een ILGenerator geimplementeerd.

Bij de Getter is de eerste IL call is `Emit(OpCodes.Ldarg_0)`.Deze call maakt een plaats op de stack waarin de returnvalue van de routine geplaatst kan worden. (Deze is ook nodig bij een Sub, omdat dit eigenlijk een functie is met als returnvalue <void>). Hierna komt de call `Emit(OpCodes.Ldfld, FieldBuilder)` die de waarde van de private variabele die met behulp van de fieldbuilder is gemaakt leest, waarna deze met de call `Emit(OpCodes.Ret)` wordt teruggegeven.

De setter maakt ook eerst plaats voor de returnvalue, waarna de parameter wordt geladen. Deze parameter wordt vervolgens in de private variabele die met behulp van de fieldbuilder is gemaakt weggeschreven, waarna de functie beeindigd wordt.

Een property kan vanaf nu dus simpel worden toegevoegd mey de call:
`CreateProperty(TypeBuilder, propName, propType)`

Stap 4: De complete class generator

Door nog een paar kleine wijzigingen toe te voegen, zoals een structure voor het netjes opslaan van de property naam en type, een wrapper over de al gemaakte functies heen te leggen waarin de gemaakte class geinstantieerd wordt via de Activator class, is de code van de classgenerator klaar:

```
Imports System.Reflection
Imports System.Reflection.Emit

Public Module ClassBuilder

    Public Structure PropertySet
        Dim Name As String
        Dim [Type] As Type
    End Structure

    Public Function CreateClass(ByVal className As String, _
                                ByVal properties As Generic.List(Of PropertySet)) As Type
        Return CreateNewClass(className, properties)
    End Function

    Public Function CreateInstance(ByVal classType As Type) As Object
        Return Activator.CreateInstance(classType)
    End Function

    Private Function CreateNewClass(ByVal ClassName As String, _
                                    ByVal properties As Generic.List(Of PropertySet)) As Type
        Dim ModuleBuilder As ModuleBuilder = CreateAssembly(ClassName)
        Dim TypeBuilder As TypeBuilder = CreateClass(ModuleBuilder, ClassName)

        For Each prop As PropertySet In properties
            CreateProperty(TypeBuilder, prop.Name, prop.Type)
        Next
        Return TypeBuilder.CreateType
    End Function
```

```
    Private Function CreateAssembly(ByRef className As String) As ModuleBuilder
        className &= Guid.NewGuid.ToString.Replace("-", "_") ' create a unique name

        Dim an As New AssemblyName(className)
        Dim ad As AppDomain = AppDomain.CurrentDomain
        Dim ab As AssemblyBuilder = ad.DefineDynamicAssembly(an, AssemblyBuilderAccess.Run)

        Return ab.DefineDynamicModule(an.Name)
    End Function

    Private Function CreateClass(ByVal builder As ModuleBuilder, _
                                 ByVal className As String) As TypeBuilder
        Return builder.DefineType(className, TypeAttributes.Public Or TypeAttributes.Class)
    End Function

    Private Sub CreateProperty(ByVal builder As TypeBuilder, ByVal propertyName As String, _
                               ByVal propertyType As Type)
        Dim FieldBuilder As FieldBuilder = _
                          builder.DefineField("f" & propertyName, propertyType, FieldAttributes.Private)
        Dim PropBuilder As PropertyBuilder = _
                     builder.DefineProperty(propertyName, PropertyAttributes.None, propertyType, Nothing)
        Dim getSetAttr As MethodAttributes = _
                    MethodAttributes.Public Or MethodAttributes.SpecialName Or MethodAttributes.HideBySig
        Dim PropGetAccessBuilder As MethodBuilder = _
                    builder.DefineMethod("Get" & propertyName, getSetAttr, propertyType, Type.EmptyTypes)
        Dim PropSetAccessBuilder As MethodBuilder = _
              builder.DefineMethod("Set" & propertyName, getSetAttr, Nothing, New Type() {propertyType})
        Dim propGetIL As ILGenerator = PropGetAccessBuilder.GetILGenerator()
        Dim propSetIL As ILGenerator = PropSetAccessBuilder.GetILGenerator()

        propGetIL.Emit(OpCodes.Ldarg_0)
        propGetIL.Emit(OpCodes.Ldfld, FieldBuilder)
        propGetIL.Emit(OpCodes.Ret)
        PropBuilder.SetGetMethod(PropGetAccessBuilder)

        propSetIL.Emit(OpCodes.Ldarg_0)
        propSetIL.Emit(OpCodes.Ldarg_1)
        propSetIL.Emit(OpCodes.Stfld, FieldBuilder)
        propSetIL.Emit(OpCodes.Ret)
        PropBuilder.SetSetMethod(PropSetAccessBuilder)
    End Sub

End Module
```

Het dynamisch sorteren van de data met een comparer method

We hebben nu de extension method en de class generator klaar, zodat we nu kunnen beginnen aan het implementeren van de pivottering van de aangeboden data.

We beginnen nu met het ordenen van de data, zodat deze alvast in de juiste volgorde staat voor het tonen in de Pivot.

De velden waarop gesorteerd moet worden, zijn als een Generic.List(Of string) meegegeven. De waarden voor deze properties worden met behulp van reflection opgehaald uit de aangeboden class:

```
Dim PropOrderInfo As Reflection.PropertyInfo = x.GetType.GetProperty(field)
OrderByValue = PropOrderInfo(1).GetValue(x, Nothing)
```

De property info van de public property wordt opgehaald, waarna de waarde van deze property wordt gelezen uit de instantie x (nothing staat voor het aantal parameters van deze property – geen parameters dus). Indien de property niet gevonden is, is PropOrderInfo Nothing. Hierop moet dus nog wel getest worden.

Voor sortering hebben we te maken met 2 verschillende instanties die met elkaar vergeleken worden. In het volgende stuk haal ik de waarden op voor instantie x en y:

```
        Dim PropOrderInfo As Reflection.PropertyInfo
        Dim OrderByValue(1) As Object

        PropOrderInfo = x.GetType.GetProperty(field)
        If PropOrderInfo IsNot Nothing Then
            OrderByValue(0) = PropOrderInfo.GetValue(x, Nothing)
            OrderByValue(1) = PropOrderInfo.GetValue(y, Nothing)
        Else
            OrderByValue(0) = Nothing
            OrderByValue(1) = Nothing
        End If
```

Omdat x en y van hetzelfde type zijn, hoeft de property info maar 1x ophaald te worden, waarna deze gebruikt kan worden om de warden uit beide instanties te lezen. Indien de property niet bestaat (of parameters heeft, of niet public is), worden de waardes op Nothing gezet.

Vervolgens kunnen de object waardes (van een onbekend type) met elkaar vergeleken worden. Omdat de types niet bekend zijn, weten we ook niet of de operators > en < geimplementeerd zijn. Dit proberen we dus gewoon uit via een Try-Catch constructie; als de operators niet geimplementeerd zijn vallen we terug op de ToString methode, waarna we een string vergelijking kunnen uitvoeren:

```
    Dim ReturnValue As Integer = 0
    Try
        If OrderByValue(0) Is Nothing AndAlso OrderByValue(1) Is Nothing Then
            ReturnValue = 0
        ElseIf OrderByValue(0) Is Nothing AndAlso OrderByValue(1) IsNot Nothing Then
            ReturnValue = -1
        ElseIf OrderByValue(0) IsNot Nothing AndAlso OrderByValue(1) Is Nothing Then
            ReturnValue = 1
        ElseIf OrderByValue(0) < OrderByValue(1) Then 'option strict off needed
            ReturnValue = -1
        ElseIf OrderByValue(0) > OrderByValue(1) Then 'option strict off needed
            ReturnValue = 1
        End If
    Catch ex As Exception
        'operators > and < are not defined, so do a string comparisation
        If OrderByValue(0).ToString < OrderByValue(1).ToString Then
            ReturnValue = -1
        ElseIf OrderByValue(0).ToString > OrderByValue(1).ToString Then
            ReturnValue = 1
        End If
    End Try
```

Nu kunnen we hier een loop omheen zetten om te vergelijken op alle opgegeven velden. Op het moment dat x groter of kleiner is dan y op een bepaald property, kunnen we de loop verlaten omdat het resultaat dan bekend is.

En voila; we hebben een dynamische comparer gemaakt die we toevoegen aan dezelfde module als de Extension method:

```
    Private fRowOrderByFields As Generic.List(Of String)

    <Global.System.Diagnostics.DebuggerNonUserCodeAttribute()> _
```

```
Private Function Comparison(Of T)(ByVal x As T, ByVal y As T) As Integer
    Dim ReturnValue As Integer = 0
    Dim PropOrderInfo As Reflection.PropertyInfo
    Dim OrderByValue(1) As Object

    For Each field As String In fRowOrderByFields
        PropOrderInfo = x.GetType.GetProperty(field)
        If PropOrderInfo IsNot Nothing Then
            OrderByValue(0) = PropOrderInfo.GetValue(x, Nothing)
            OrderByValue(1) = PropOrderInfo.GetValue(y, Nothing)
            Try
                If OrderByValue(0) Is Nothing AndAlso OrderByValue(1) Is Nothing Then
                    ReturnValue = 0
                ElseIf OrderByValue(0) Is Nothing AndAlso OrderByValue(1) IsNot Nothing Then
                    ReturnValue = -1
                    Exit For
                ElseIf OrderByValue(0) IsNot Nothing AndAlso OrderByValue(1) Is Nothing Then
                    ReturnValue = 1
                    Exit For
                ElseIf OrderByValue(0) < OrderByValue(1) Then 'option strict off needed
                    ReturnValue = -1
                    Exit For
                ElseIf OrderByValue(0) > OrderByValue(1) Then 'option strict off needed
                    ReturnValue = 1
                    Exit For
                End If
            Catch ex As Exception
                'operators > and < are not defined, so do a string comparisation
                If OrderByValue(0).ToString < OrderByValue(1).ToString Then
                    ReturnValue = -1
                    Exit For
                ElseIf OrderByValue(0).ToString > OrderByValue(1).ToString Then
                    ReturnValue = 1
                    Exit For
                End If
            End Try
        End If
    Next
    Return ReturnValue
End Function
```

De comparer kunnen we nu toevoegen aan de extension Method, waarna het resultaat alvast juist gesorteerd is:

```
<System.Runtime.CompilerServices.Extension()> _
Public Function Pivot(Of T)(ByVal list As Generic.List(Of T), _
                            ByVal ShowField As Generic.List(Of String), _
                            ByVal PivotOrderbyFields As Generic.List(Of String), _
                            ByVal PivotField As String) As Generic.List(Of Object)
    fRowOrderByFields = ShowFields
    List.Sort(new Comparison(Of T)(AddressOf Comparison))
    Return New Generic.List(Of Object)(list.ToArray)
End Function
```

Het maken van de Pivot

Stap 1: Het kopieren van de property gegevens van niet-pivot columns

De volgende stap is het kopieren van de property gegevens (zonder de pivot waardes). Deze waardes worden in de volgende stap uitgebreid met properties voor de pivot columns.
Hiervoor gaan we de meegegeven fields doorlopen en opslaan in een Generic.List(Of PropertySet) die gebruikt kan worden om de class te genereren:

```
    Private Function AddFieldProperties(Of T)(ByVal list As Generic.List(Of T), _
                                              ByVal showFields As Generic.List(Of String)) _
                                                    As Generic.List(Of ClassBuilder.PropertySet)
        Dim Properties As New Generic.List(Of ClassBuilder.PropertySet)
        Dim prop As ClassBuilder.PropertySet
        Dim PropInfo As Reflection.PropertyInfo

        For Each visibleField As String In showFields
            PropInfo = list.Item(0).GetType.GetProperty(visibleField)
            If PropInfo IsNot Nothing Then
                prop = New ClassBuilder.PropertySet
                prop.Name = visibleField
                prop.Type = PropInfo.GetType
                Properties.Add(prop)
            End If
        Next
        Return Properties
    End Function
```

Stap 2: Het genereren van de property gegevens voor de pivot columns

Nu gaan we de property gegevens voor de pivot columns genereren; in ons geval dus de unieke waardes in de column “Lengte”:

```
    Private Function AddPivotProperties(Of T)(ByVal list As Generic.List(Of T), _
                                        ByVal properties As Generic.List(Of ClassBuilder.PropertySet), _
                                        ByVal PivotOrderbyFields As Generic.List(Of String), _
                                        ByVal PivotField As String) _
                                                    As Generic.List(Of ClassBuilder.PropertySet)
        Dim PivotProps As New Generic.SortedDictionary(Of String, ClassBuilder.PropertySet)
        Dim prop As ClassBuilder.PropertySet
        Dim PropValueType As System.Type
        Dim PivotColumnName As String

        'The pivot value Type
        PropValueType = list.Item(0).GetType.GetProperty(PivotField).PropertyType

        For Each line As T In list
            'Create the pivot column name from the values of the orderBy columns
            PivotColumnName = MakePivotColumnNameForLine(list, line, PivotOrderbyFields)
            'If we do not have this value yet, we are going to add the property
            If Not PivotProps.ContainsKey(PivotColumnName) Then
                prop.Name = PivotColumnName
                'The value type of the column is the value type of the pivot column
                prop.Type = PropValueType
                'And add the new property to the list
                PivotProps.Add(PivotColumnName, prop)
            End If
```

```
        Next
        'And we return the generic list of pivot properties
        Return New Generic.List(Of ClassBuilder.PropertySet)(PivotProps.Values.ToArray)
    End Function

    Private Function MakePivotColumnNameForLine(Of T)(ByVal list As Generic.List(Of T), _
                                        ByVal line As T, _
                                        ByVal PivotOrderbyFields As Generic.List(Of String)) As String
        Dim PropOrderInfo(PivotOrderbyFields.Count - 1) As Reflection.PropertyInfo
        Dim OrderBy As String = ""

        'the pivot orderby fields property info
        For i As Integer = 0 To PivotOrderbyFields.Count - 1
            PropOrderInfo(i) = list.Item(0).GetType.GetProperty(PivotOrderbyFields(i))
        Next

        For i As Integer = 0 To PivotOrderbyFields.Count - 1
            If PropOrderInfo(i) IsNot Nothing Then
                If PropOrderInfo(i).GetValue(line, Nothing) IsNot Nothing Then
                    OrderBy &= PropOrderInfo(i).GetValue(line, Nothing).ToString
                End If
            End If
        Next
        'We cannot have numeric properties, so add an underscore to nummeric values
        If IsNumeric(OrderBy) Then
            OrderBy = "_" & OrderBy
        End If
        Return OrderBy
    End Function
```

We halen eerst de property info op voor de pivot orderby columns, en de type van de pivot column. Vervolgens lopen we de bron-generic list door om de unieke orderby values uit te lezen, en deze unieke waarden worden in de juiste volgorde teruggeven (de sortering wordt automatisch uitgevoerd door de sorted dictionary).

Stap 3: Het genereren van het resultaat

Nu hebben we alle informatie om een nieuwe class te genereren en deze te vullen met de juiste waardes:

```
    Private Function FillPivot(Of T)(ByVal list As Generic.List(Of T), _
                                ByVal rowProperties As Generic.List(Of ClassBuilder.PropertySet), _
                                ByVal pivotProperties As Generic.List(Of ClassBuilder.PropertySet), _
                                ByVal PivotOrderbyFields As Generic.List(Of String), _
                                ByVal pivotField As String) As Generic.List(Of Object)
        Dim Pivot As New Generic.List(Of Object)
        Dim AllProperties As New Generic.List(Of ClassBuilder.PropertySet)
        Dim NewClassType As Type
        Dim NewClass As Object = Nothing
        Dim MyPropInfo As Reflection.PropertyInfo
        Dim NewPropInfo As Reflection.PropertyInfo
        Dim PivotPropInfo As Reflection.PropertyInfo
        Dim AllFields As String
        Dim LastAllFields As String = ""
        Dim PivotColumnName As String
        Dim PivotValue As Object

        'One list with all properties
        AllProperties.AddRange(rowProperties)
```

```
        AllProperties.AddRange(pivotProperties)

        'Create the new type
        NewClassType = CreateClass("pivot", AllProperties)

        'The property info of the pivot value column
        PivotPropInfo = list.Item(0).GetType.GetProperty(pivotField)

        'Loop all lines to fill the new pivot list
        For Each line As T In list
            'Get the unique values from the non-pivot field, so we can suppress the duplicate rows
            AllFields = ""
            For Each prop As PropertySet In rowProperties
                MyPropInfo = line.GetType.GetProperty(prop.Name)
                AllFields &= MyPropInfo.GetValue(line, Nothing).ToString
            Next
            If AllFields <> LastAllFields Then
                'New unique row, so add a new generated class
                LastAllFields = AllFields
                NewClass = CreateInstance(NewClassType)
                'Copy the values of the all the row column properties
                For Each prop As PropertySet In rowProperties
                    MyPropInfo = line.GetType.GetProperty(prop.Name)
                    NewPropInfo = NewClass.GetType.GetProperty(prop.Name)
                    NewPropInfo.SetValue(NewClass, MyPropInfo.GetValue(line, Nothing), Nothing)
                Next
                'Add the class to the pivot list
                Pivot.Add(NewClass)
            End If
            'Create the pivot column name from the values of the orderBy columns,
            'and get the info on this column in the pivot class
            PivotColumnName = MakePivotColumnNameForLine(list, line, PivotOrderbyFields)
            NewPropInfo = NewClass.GetType.GetProperty(PivotColumnName)

            'get the curreny value of the pivot column
            PivotValue = NewPropInfo.GetValue(NewClass, Nothing)

            'Get the value of the pivot column and add this to the existing pivotvalue
            MyPropInfo = line.GetType.GetProperty(pivotField)
            PivotValue += MyPropInfo.GetValue(line, Nothing)

            'And write the new value to the pivot
            NewPropInfo.SetValue(NewClass, PivotValue, Nothing)

        Next
        Return Pivot
    End Function
```

Eerst voegen we de row properties en pivot properties bij elkaar zodat we de juiste property array hebben om de class type te genereren, wat we dan vervolgens direct doen. Hierna halen we de property info van de value column op, en gaan alle lines in de bron list doorloopen om de nieuwe class te vullen.

Het resultaat is een gepivotteerde Generic.List.