Mark Blomsma is softwarearchitect bij Omnext.NET

C# 2.0

DE NIEUWE FEATURES VAN C# 2.0 ONDER DE LOEP

In de loop van 2005 komt het langverwachte Whidbey uit. Whidbey is de codenaam van Visual Studio 2005. Inmiddels is bèta 1 van dit product uit en kun je deze met een MSDN-abonnement downloaden van de Microsoft-website. Visual Studio 2005 is niet slechts een upgrade van de huidige editie. Naast een enorme hoeveelheid nieuwe features wordt samen met Visual Studio 2005 ook het .NET Framework 2.0 uitgebracht. Dit betekent een nieuwe versie van de .NET Runtime. De runtime is verbeterd, gefixt en natuurlijk uitgebreid. Met C# 2.0 wordt het mogelijk de nieuwe features en uitbreidingen van de Common Language Runtime - zoals Generics - te benutten. In C# 2.0 zul je onder andere de volgende vernieuwingen aantreffen: Generics, Nullable types, Anonymous methods, Iterators en Partial types. De auteur neemt de vernieuwingen een voor een onder de loep.

Generics maken het mogelijk om classes, structs, interfaces, delegates en methods te parametriseren en te manipuleren, met het type data die zij bevatten. Dit is handig omdat Generics het mogelijk maken dit te doen vanuit de compiler en zo compile-time type-checking uit te voeren, minder expliciete conversies uit te voeren tussen data-types en de noodzaak voor 'boxing' en run-time checks terug te brengen. Generics vormen een extensie van de Common Language Runtime en maken het de ontwikkelaar mogelijk types te definiëren waarbij specifieke details nog niet zijn gespecificeerd. Deze details worden ingevuld op het moment dat de Generic Type in de code wordt gebruikt om een specifieke implementatie van de Generic Type te creëren. Hiermee wordt al duidelijk wat voor soort code we kunnen schrijven met generics: hele generieke code. Utility-classes lenen zich bijzonder goed om generiek op te zetten. Ook zaken als lijsten, collections, stacks en queue's zijn in het bijzonder geschikt, maar ook helper-classes kun je prima generiek maken.
De Common Language Runtime ondersteunt een veelvoud aan programmeertalen. Er zullen dus verscheidene syntaxen ontstaan rondom Generics. In Visual Studio 2005 zullen Visual Basic .NET en C++ ook ondersteuning voor Generics bieden. Voor J# is geen ondersteuning van Generics gepland. Tijd om te kijken hoe Generics er in code uitzien. Een specifieke implementatie van een Generic Type leidt uiteindelijk tot een Constructed Type. In codevoorbeeld 1 geeft <T> aan dat GenericMath een Generic Type-declaratie is. Pas op het moment dat we in de code GenericMath gaan gebruiken zullen we het type van <T> bepalen.
In C# wordt tussen de haken '<' en '>' een variabele aanduiding voor het, later te bepalen, System.Type aangegeven. Zonder dit type is er onvoldoende informatie om het Generic Type te instantiëren. De compiler zal hierop ook een foutmelding geven. In codevoorbeeld 2 zien we hoe we GenericMath kunnen gebruiken. In dit voorbeeld gebruiken we het type 'double' om aan te geven dat we een constructed type van het type 'GenericMath<double>' willen declareren.
Het op deze manier instantiëren heeft natuurlijk voornamelijk nut als dit type ook binnen de class gebruikt kan worden. We kunnen binnen onze generic class <T> op meerdere plekken gebruiken. Codevoorbeeld 3 laat zien dat we het type T op diverse plekken in de class gebruiken. Het mooie van Generics is dat de compiler automatisch overal controleert of we wel het juiste type hanteren op het moment dat we een constructed type maken; zie codevoorbeeld 4. Codevoorbeeld 3 laat ook zien dat het mogelijk is een constraint of beperking aan te brengen op welke classes gebruikt kunnen worden bij het declareren van de constructed type. In ons voorbeeld kunnen we alleen types gebruiken die IComparable implementeren; zie codevoorbeeld 5.

```
public class GenericMath<T>
{
    ...
}
```

Codevoorbeeld 1.
Een Generic Type declaratie

```
GenericMath<double> gmd = new GenericMath<double>();
```

Codevoorbeeld 2.
Een Constructed Type 'GenericMath<double>'

```
public class GenericMath<T> where T : IComparable
{
   private T localT;

   public T Min( T item1, T item2 )
   {
      if ( item1.CompareTo( item2 ) < 0 )
      {
         return item1;
      }
      localT = item2;
      return item2;
   }
}
```

Codevoorbeeld 3.
Gebruik <T> binnen de generic class

```
GenericMath<int> gmd = new GenericMath<int>();
gmd.Min( 1, 2 );      // gaat goed, resultaat is 1
gmd.Min( 1.1, 2 );    // compiler error!
```

Codevoorbeeld 4.
De compiler controleert de types

Er kan nog meer met Generics. Zo is het bijvoorbeeld ook mogelijk een Generic methode te definiëren. Belangrijk is om te weten dat een aantal zaken ook niet kan met Generics:

- Het is niet mogelijk webservices te definiëren met Generic Type-parameters. Dit komt doordat de W3C-standaards voor webservices het concept van Generics niet onderkennen.
- Het is niet mogelijk om Generic Types te definiëren die erven van Serviced Components. De reden hiervoor is dat Generics niet voldoen aan de COM visibility-eisen die gelden voor Serviced Components.

Ook het .NET Framework 2.0 maakt gebruik van Generics en biedt diverse baseclasses aan die nu met behulp van een generic zijn geïmplementeerd. Het meest in het oog springend zijn de classes die beschikbaar zijn in de namespace 'System.Collections.Generic'. Hier wordt nu bijvoorbeeld een generic List<T> aangeboden. Deze list biedt de mogelijkheid aan om een constructed type te maken waarbij bijvoorbeeld een lijst van integers gemaakt kan worden.

Nullable Types

In de system namespace is de nieuwe class 'System.Nullable<T>' opgenomen; een Generic-class die aan een willekeurige class de eigenschap 'Nullable' toevoegt. Binnen C# 2.0 is er vervolgens een nieuwe syntax geïntroduceerd om handig gebruik te maken van deze Generic. Hiervoor wordt het '?' gebruikt. Codevoorbeeld 6 laat zien hoe dit werkt.
Zoals codevoorbeeld 6 laat zien, is het mogelijk om 'i' impliciet te laten casten naar 'j'. Dit gebeurt op het moment dat we zeggen 'j = i'. Indien je dit ondoorzichtig vindt, kun je ook gebruik maken van de extra properties zoals 'HasValue', een Boolean property die aangeeft of de variabele ongelijk aan null is, en 'Value', de waarde van de variabele. Beide zijn beschikbaar op alle Nullable types. Dit wordt gedemonstreerd door if (j.HasValue) i = j.Value;. Het nullable kunnen maken van type is met name interessant als je in C# toch een soort optionele parameter wilt meegeven, of juist expliciet leeg wilt laten.
De code in codevoorbeeld 7 kan worden aangeroepen door UpdateKlant("Mark", null, null). Zeker op het moment dat er gegevens van en naar de database moeten worden geschreven, kan het nog wel eens voorkomen dat een waarde onbekend is. Met nullable types is het nu ook mogelijk deze ongeïnitialiseerde waarden als null binnen .NET te gebruiken.

Anonymous methods

C# 2.0 biedt de mogelijkheid anonieme methodes te gebruiken. Een anonieme methode is een stuk code die op een plek ingetikt kan worden waar vroeger alleen een benoemde methode gehanteerd kon worden. Dit onderwerp speelt bij delegates.
Codevoorbeeld 8 toont dat het in C# 1.1 noodzakelijk is altijd een gedefinieerde methode mee te geven aan de new EventHandler(AddClick). Met C# 2.0 hoeft dit niet meer en is het mogelijk om op de plek van de eventhandler direct de code te plaatsen. Dit heeft ook als voordeel dat de code die in-line wordt opgevoerd de beschikking heeft over de 'shared local state' van de overkoepelende methode. Hoe code uit codevoorbeeld 8 na herschrijving met C# 2.0 eruit ziet, toont codevoorbeeld 9.
We zien in codevoorbeeld 9 dat in de syntax van een stuk anonymous code gebruik wordt gemaakt van het keyword delegate, gevolgd door een optionele lijst met parameters en vervolgens binnen een accoladeblok de code die bij de anonymous methode hoort. Codevoorbeeld 10 laat zien hoe je parameters mee kunt geven.

Iterators

In C# is het mogelijk om met het foreach-statement een verzameling van elementen te itereren, mits die collection 'enumerable' is. In C# 2.0 zijn iterators methodes die herhaald kunnen worden aangeroepen om iedere keer een volgende waarde in een sequence

```
GenericMath<TextBox> x = new GenericMath<TextBox>(); // compiler error
```

Codevoorbeeld 5.
De compiler meldt een error omdat IComparable niet wordt geïmplementeerd door TextBox

```
int i = 0;
i = null;   // compiler error

int? j = 0;
j = null;
j = i;
if ( j.HasValue ) i = j.Value;

System.Nullable<int> k = 0;  // is gelijk aan: int? k = 0
k = null;
```

Codevoorbeeld 6.
Nullable types

```
public void UpdateKlant( string naam, DateTime? geboorteDatum,
int? aantalKinderen )
{
    // TODO
}
```

Codevoorbeeld 7.
Nullable parameters

```
class InputForm: Form
{
   ListBox listBox;
   TextBox textBox;
   Button addButton;
   public MyForm()
   {
      ListBox = new ListBox(...);
      textBox = new TextBox(...);
      addButton = new Button(...);
      addButton.Click += new EventHandler(AddClick);
   }
   void AddClick(object sender, EventArgs e)
   {
      listBox.Items.Add(textBox.Text);
   }
}
```

Codevoorbeeld 8.
In C# 1.1 moest een delegate benoemd zijn

```
class InputForm: Form
{
   ListBox listBox;
   TextBox textBox;
   Button addButton;
   public MyForm() {
      listBox = new ListBox(...);
      textBox = new TextBox(...);
      addButton = new Button(...);
      addButton.Click += delegate {
         listBox.Items.Add(textBox.Text);
      };
   }
}
```

Codevoorbeeld 9.
Hetzelfde resu ltaat met inline-code

```
addButton.Click += delegate(object sender, EventArgs e) {
   MessageBox.Show(((Button)sender).Text);
};
```

Codevoorbeeld 10.
Een anonymous methode met parameters

```
public class CarCollection : IEnumerable
{
   string[] _cars = { "Firebird", "T-Ford", "Chevy"};

   public IEnumerator GetEnumerator()
   {
      for ( int i = 0; i< _cars.Length; i++ )
         yield return _cars [i];
   }

   public IEnumerator Alphabetical()
   {
      string[] _sortedCars = SortCars( _cars );
      for ( int i = 0; i< _sortedCars.Length; i++ )
         yield return _sortedCars [i];
   }

}

   // nu een methode om de iterator te gebruiken
public void Test()
{
   CarCollection cars = new CarCollection();
   foreach ( string car in cars )
   {
      Console.WriteLine( car );
   }
   foreach ( string car in cars.Alphabetical )
   {
   Console.WriteLine( car );
   }
}
```

Codevoorbeeld 11.

Implem enteer meer iterators op een class

```
// dit deel van class zit in file Test99a.cs
public partial class Test99
{
   public Test99()
   {
      // ondanks dat Age in een andere file zit is
      // deze toch via this te benaderen
      System.Diagnostics.Debug.WriteLine( this.Age );
   }
}

// dit deel van class zit in file Test99b.cs
public partial class Test99
{
   public int Age()
   {
      return 55;
   }
}
```

Codevoorbeeld 12.

Een partial class

terug te geven. In C# 1.1 is het noodzakelijk om IEnumerable te implementeren om een class itereerbaar te maken. Met C #2.0 wordt dit door middel van een iterator een stuk eenvoudiger. Een iterator is een methode die een waarde 'yield' haalt uit een reeks van waardes. Een iterator is een statement-block - code tussen accolades - dat zich onderscheidt van reguliere code door de aanwezigheid van een of meer yield-statements:

- het yield return-statement geeft de volgende waarde binnen de reeks
- het yield break-statement geeft aan dat de laatste waarde binnen de reeks is geweest.

In codevoorbeeld 11 wordt weergegeven dat het met het 'yield statement' betrekkelijk eenvoudig is een enumerator te implementeren. Ook laat dit voorbeeld zien dat het mogelijk is om meer iterators aan te maken. In dit voorbeeld is de default iterator gebruikt om door een array van auto's te lopen, maar is een aparte iterator geïmplementeerd die zorgt dat we met ons foreach-statement een alfabetische lijst van auto's kunnen inzien.

Partial types

Partial types is een eenvoudige, maar voor de ontwikkelaar zeer ingrijpende verandering in de C#-taal. Met de komst van partial types is het mogelijk de definitie van een class te splitsen over verscheidene files. Dit is bijzonder handig als je met meer ontwikkelaars aan een zelfde class wilt ontwikkelen, of als een deel van de class door een wizard of designer wordt gegenereerd. In ASP.NET 2.0 zijn alle pagina's nu partial classes en wordt de code-behind dus niet meer via inheritance benaderd, maar kan de page, als zijnde een partial class, gewoon de methodes op zichzelf benaderen. Voor het toepassen van de partial class is het keyword 'partial' beschikbaar. Zo staat in codevoorbeeld 12 dat bij het splitsen van een class over twee files de properties en methodes van de gehele class toch in beide files beschikbaar zijn. Het is overigens niet mogelijk een partial class-file toe te voegen aan een reeds gecompileerde assembly. Met andere woorden, alle delen van een partial class moeten in hetzelfde project en dus in dezelfde assembly geplaatst worden.

Tot slot

We hebben gekeken naar de nieuwe features van C#2.0. Het doel van de nieuwe features is om het leven van ons ontwikkelaars gemakkelijker te maken en om ons van extra gereedschap te voorzien om het ontwikkelwerk beter te kunnen doen. De code in dit artikel is tot stand gekomen op basis van de Mei Alpha-versie en Juni Bèta 1-versie van Visual Studio 2005 Enterprise Architect-editie. Deze laatste is inmiddels te downloaden met een MSDN-abonnement. C# 2.0 is ook beschikbaar als C# 2.0 Express bèta 1. Deze versie biedt qua taal dezelfde mogelijkheden en is voor iedereen vrij te downloaden.

Nuttige internetadressen

Downloads from Microsoft, http://msdn.microsoft.com/downloads

C# 2.0 Language specifications, http://download.microsoft.com/download/8/1/6/81682478-4018-48fe-9e5e-f87a44af3db9/SpecificationVer2.doc

C# Express, http://msdn.microsoft.com/express

Mark Blomsma is softwarearchitect bij Omnext.NET (www.omnext.net), en daarnaast voorzitter van het Nederlandse C# Developer Network (www.developernetwork.nl) en een Most Valuable Professional voor .NET. E-mail: mark.blomsma@omnext.net