Remi Caron is CTO bij Wantit B.V.

Visual Basic .NET versus C#

EEN KWESTIE VAN SMAAK

Met de komst van .NET en de daarbij behorende belofte van cross-language code-reuse, zou je een staakt-het-vuren verwachten tussen de strijdende partijen. In de praktijk blijkt dit echter niet het geval te zijn. De C#-ontwikkelaar en de Visual Basic .NET-ontwikkelaar hebben over en weer behoorlijk wat op elkaar aan te merken; beter gezegd op elkaars voorliefde voor C# of Visual Basic .NET. Heeft één van beiden gelijk? Wat zijn eigenlijk de verschillen tussen beide kampen? Zijn de talen wel zo weinig verschillend als men ons wil doen geloven? Zijn de verschillen in Visual Studio 2005 toegenomen of afgenomen? En als de verschillen klein zijn, blijft dat dan ook zo?

Het mag duidelijk zijn dat de syntax van beide talen afwijkt. C# is van de grond af opgebouwd en heeft geleerd van al het goede van reeds bestaande talen. Visual Basic .NET aan de andere kant vindt duidelijk zijn oorsprong in Visual Basic 6.0 en zijn voorgangers. Als je voor een van beide talen kiest om daar al jouw projecten mee uit te voeren, zul je van deze verschillen geen last hebben. Maar wat als je je niet in deze luxe positie bevindt en je als consultant op een project komt waar ze beide door elkaar worden gebruikt? Of je koopt een third party library vanwege de geboden functionaliteit en deze is geschreven in een andere taal dan de taal van jouw voorkeur? Op dat moment gaan de verschillen in de talen wel degelijk een rol van betekenis spelen.

Zo zijn identifiers in C# case-sensitive daar waar dit in Visual Basic .NET niet het geval is. Dit is vaak een onderdeel van discussie tussen de Visual Basic-ontwikkelaars en de C#-developer. De Visual Basic-ontwikkelaars vinden dit veelal vervelend, terwijl de C#-ontwikkelaars dit de normaalste zaak van de wereld vinden. De verschillende kijk hierop wordt vooral ingegeven door de achtergrond van de ontwikkelaar in kwestie. Terwijl de Visual Basic .NET-ontwikkelaar zijn roots heeft liggen in een Visual Basic-versie x - ook zonder case-sensitivity, daar komt de C#-ontwikkelaar vaak uit de C++ of Java-hoek waar de case-sensitivity juist weer heel normaal is. Ik heb zelf een achtergrond in een niet-case-sensitive en niet-strong-typed ontwikkeltool, maar door de vele webapplicaties in combinatie met JScript heb ik er aan kunnen wennen. Hetzelfde geldt voor het gebruik van de *{}* als statementblock-indicators. Ik moet zeggen dat bij het nesten van conditional statements de leesbaarheid van de code er niet op vooruit gaat. De duidelijkheid van bijvoorbeeld een ***End if*** biedt in dat opzicht meer overzicht. Ik heb vanuit mijn Javascript-ervaring de gewoonte van het inline-comment op de ***closing }*** meegenomen. Hiermee creëer je dan zelf weer overzicht in combinatie met de juiste indentation.

Het declareren en toekennen van waarden aan variabelen is ook afwijkend. Dit wordt aangetoond met een tweetal eenvoudige voorbeelden ter illustratie van de verschillen in syntax.

Purpose	Visual Basic .NET	C#
Declare an integer	DIM i,j As Integer	int I,j;
Declare an array	DIM i(6) As Integer	int[] = new int[6];

Tabel 1. Verschillen in syntax: array-declaraties

Uit bovenstaande tabel licht ik de array er even specifiek uit, want daar zit een addertje onder het gras. Als je in C# een array declareert met vijf elementen ***int[]a = new int[5];*** , dan bestaat de array uit de elementen a[0] t/m a[4]. Als je een dergelijke array in Visual Basic .NET declareert ***Dim a() As Integer = new Integer(5)***, dan krijg je zes elementen a(0) t/m a(5). Visual Basic voegt automatisch een extra element toe aan de array om het porten of het hergebruik van 'oude' Visual Basic-libraries iets te vereenvoudigen. Dit kan leiden tot problemen als je array's gebruikt als parameter tussen routines die zijn geschreven in de verschillende talen. Voor de Visual Basic-ontwikkelaar geldt dat hij ook moet opletten als er array's als parameter worden gestuurd naar Framework-routines.

In de voorgaande paragraaf zijn de verschillen tussen de array's belicht. Zoals je wellicht is opgevallen gebruikt C# *[]* en Visual Basic .NET *()* om een array te declareren. Dit verschil beperkt zich echter niet tot array-declaraties alleen, zoals de tabel 2 weergeeft.

Statement termination

In C# wordt ieder statement beëindigd met een ; , terwijl in Visual Basic het einde van een regel een statement beëindigt. In Visual Basic en in C# kun je meer statements op een regel kwijt en een statement over meer regels verdelen zoals uit het voorbeeld in tabel 3 blijkt.

In Visual Basic .NET gebruik je de : om meer statements op één regel te plaatsen en de _ om een statement over meer regels te verdelen. In C# is het plaatsen van de ; op de juiste plek voldoende voor hetzelfde resultaat. In de versies voorafgaand aan de

```
if (myVar == 1)
{
   If (yourVar == 2)
   {
      ...Some code;
   } // if yourVar
else
{
   ... Some Code;
} //else myVar
} // if myVar
```

Codevoorbeeld 1.

Overzicht door juiste indentation

Purpose	Visual Basic .NET	C#
Declare an array	Dim i(6) As Integer	int[] = new int[6];
Initialize an array	Dim i() As Integer = {1,2,3,4,5}	int[] = new int[5] {1,2,3,4,5};
Reallocate an array	Redim i(10)	Niet beschikbaar
Property indexers	myDataSet.Tables("Authors")_ .Rows(5)._ Columns("AuthorId")	myDataSet.Tables["Authors"] .Rows[5]. Columns["AuthorId"]

Tabel 2. Nog meer verschillen

komende release van Visual Studio 2005 heeft Visual Basic .NET geen ondersteuning geboden aan de volgende datatypes: sByte, uShort, uLong, en uInt. In de volgende versie van Visual Studio is deze 'achterstand' op C# ingelopen. Tot en met versie 2003 is dit een verschil geweest tussen Visual Basic .NET en C#; en bij interop tussen beide talen eentje om rekening mee te houden.

Operator overloading

Operator overloading was een andere feature die ontbrak in de Visual Basic .NET-versies tot aan de Visual Studio 2005 release waarin deze functionaliteit beschikbaar is voor de Visual Basic-ontwikkelaar. Je maakt gebruik van operator overloading als je twee strings samenvoegt in een nieuwe string met behulp van het **+-teken**.

De string-class heeft de + operator overloaded om in plaats van een daadwerkelijke optelling de strings samen te voegen in de variabele s3. Vanaf Visual Studio 2005 ben je in staat om ook in jouw eigen classes-operator overloading toe te passen. Naast de eerder gememoreerde verschillen zijn er tal van kleine verschillen op taalniveau, zoals alle conditional structures, operators, error handling, unmanaged code enzovoort. Het zou te ver voeren deze allemaal hier te behandelen. De verschillen tussen de talen komen wellicht beter tot uiting door naar de voordelen van beide talen te kijken. We zetten de voordelen op een rij.

Visual Basic .NET

- Ondersteuning voor optionele parameters. Dit is handig wanneer gebruik wordt gemaakt van COM+-componenten (OLE automation). In C# moet er voor alle parameters System.Missing.Value worden ingevuld op optionele parameters.
- Ondersteuning voor late binding met de Option strict off. De type-safety wordt geofferd, maar het hergebruik van legacy-libraries wordt eenvoudiger.
- Named indexers - wordt ook wel naar gerefereerd als: properties met parameters.
- Veel legacy Visual Basic-functies worden geleverd in de Microsoft. Visual Basic namespace. Deze functies zijn niet altijd efficiënt in gebruik, maar het vereenvoudigt wel de overstap van Visual Basic versie x naar Visual Basic .NET. Deze assembly kan ook worden gebruikt in andere talen dan Visual Basic .NET, bijvoorbeeld in C#.

```
• With construct:
With MyButton
   .Height = 2000
   .Width = 2000
   .Text = "MyButton"
End With
```

Dit is een voordeel voor de Visual Basic-ontwikkelaar. Het scheelt in ieder geval een hoop typewerk. In C# zouden alle regels voluit moeten worden geschreven.

Visual Basic .NET	C#

a=5
b=6 : c=7
MyFunc(a,_
b,_
c)
	a=5;
	b=6; c=7;
	MyFunc(a,
	b,
	c);

Tabel 3. Meer statements op een regel

- Eenvoudiger in zijn syntax voor event-handling. Een methode kan aangeven dat het een event afhandelt in plaats van dat de handler moet zijn gedefinieerd in code.
- De mogelijkheid om interfaces te implementeren met afwijkende method-namen. Dit maakt het wel moeilijker om een implementatie van een interface terug te vinden.
- Catch when clause. Biedt de mogelijkheid exceptions te filteren gebaseerd op runtime-expressies in plaats van alleen maar op type.
- Background-compiling is een voordeel voor kleinere projecten. In geval van grotere projecten vertraagt het de IDE.

C#

- *XML-documentatie gegenereerd vanuit de source-code.*
- *Operator overloading.*
- *Taalondersteuning voor unsigned types.*
- *Using-statement*
- Expliciete interface-implementatie. Een interface die al is geïmplementeerd in een base class kan opnieuw worden geïmplementeerd in een derived class. Dit maakt de code wel moeilijker te begrijpen.
- Unsafe code die nog steeds managed code is. Biedt de mogelijkheid om direct met pointer te werken zoals in C++. Dit is handig in het geval van device-drivers.
- Visual Basic is een meer 'verbose' taal wat in de praktijk meer toetsaanslagen voor hetzelfde resultaat betekent.

Opmerking: De cursief gedrukte teksten in de opsomming zijn in Visual Studio 2005 ook beschikbaar in Visual Basic .NET

De voorgaande lijsten zouden de indruk kunnen wekken dat Visual Basic de voorkeur verdient boven C#. Daarbij dient te worden opgemerkt dat enkele zaken pas in Visual Studio 2005 beschikbaar zijn. Voor de puristen onder ons heeft Visual Basic .NET redelijk wat legacy in zich waar je rekening mee moet houden en waar je bij voorkeur ook direct afstand van moet nemen in deze nieuwe omgeving. Het is vooral bedoeld om de overstap te vereenvoudigen.Tot nu toe heb ik me vooral gericht op de huidig beschikbare versies van Visual Basic .NET en C#. De voordelen van beide talen tot en met Visual Studio 2003 zijn belicht, en de belangrijkste verschillen zijn aangehaald. Het wordt tijd om te bezien hoe beide zich tot elkaar verhouden in de nieuwe release. In de volgende release van Visual Studio zijn er verschillen weggenomen en nieuwe verschillen geïntroduceerd. De grootste verschillen in de Whidbey-release zitten in de IDE en de productivity-tools. We zullen kijken naar de productiviteitverbeteringen, taalinnovaties en de diverse overige aanpassingen/uitbreidingen.

Productiviteitverbeteringen Visual Basic .NET

My Namespace

Visual Basic .NET introduceert in Visual Studio 2005 de My Namespace. In het kort is My Namespace een shortcut naar een groot aantal classes in het .NET Framework. Het biedt veelal een single line reference naar zaken die anders meer regels code

```
Dim s1 As String = "Hello"
Dim s2 As String = " World!"
Dim s3 = s1 + s2
```

Codevoorbeeld 2.
Twee strings samenvoegen met +

```
Using myClass As New myClass(arg1, arg2)
   Somecode
End using
```
Codevoorbeeld 4.
Het using-statement

```
public sealed class ClassX
{
   Private ClassX() {} // Prevent instantiating
}
```
Codevoorbeeld 3.
Het vermijden van instance methods in een static class in het verled

kosten voor hetzelfde resultaat. De My Namespace biedt de volgende classes: Application, Computer, User, Forms, Resources, en Settings. Met behulp van My Namespace kun je bijvoorbeeld My.Computer.Audio.Play ("C:\Beep.wav") aanroepen om een audiofile te starten.

Autocorrect-options
Visual Basic .NET implementeert de autocorrect-mogelijkheden in Visual Studio 2005. Er verschijnt een smart-tag naast bijvoorbeeld een foutief gespeld keyword. Als je een imports-statement hebt vergeten, zal autocorrect je de fully qualified name van de class aanreiken die je probeert te gebruiken.

Using keyword
Visual Basic .NET introduceert in Visual Studio 2005 het using-statement. Dit is niet de using van C# - imports van Visual Basic - maar een statement-block dat aan het einde van het block de dispose van een gebruikt object binnen het block aanroept. De wijze waarop het block is verlaten is hierbij niet relevant. Dit is in C# reeds aanwezig in de huidige versies.

IsNot keyword
Nog een aardige toevoeging aan Visual Basic .NET is het IsNot keyword. Dit vereenvoudigt syntax als deze: ***If Not (obj Is Nothing) Then*** naar ***If obj IsNot Nothing Then***.

Background worker object
Visual Basic .NET verlost je van de pijn om de callback van een asynchroon proces te laten draaien op de juiste thread. Dit is in de huidige releases behoorlijk wat werk. Het background worker-object is een component die dit automatisch voor je regelt. Hierdoor kun je trage processen eenvoudig op de achtergrond laten draaien. Je sleept simpelweg de component op een form, 'invoked' de benodigde methods en klaar ben je. Thread-safety en andere hoofdbrekers worden voor je afgehandeld door de component.

Productiviteitverbeteringen C#
Refactoring
De C# IDE biedt de mogelijkheid van code-refactoring (informatie over refactoring: http://www.refactoring.com) De refactoring-tool biedt de volgende refactor-mogelijkheden: method-extraction, rename, interface-extraction, field-encapsulation, method signature change, en arraylist-replacement.

Static classes
In C# is het mogelijk een static class te definiëren die alleen static methods kan bevatten; bijvoorbeeld: **Public sealed static class ClassX {}**. De compiler waarschuwt als je toch instance methods mocht hebben gedeclareerd. Dit kon in het verleden alleen worden bereikt door als volgt te werk te gaan:

Anonymous methods
C# biedt in Visual Studio 2005 de mogelijkheid om anonymous methods te gebruiken op plaatsen waar een delegate verwacht

```
btnMessage.Click += delegate(object sender, EventArgs e) {
MessageBox.Show(sender.ToString()); };
```
Codevoorbeeld 5.
Anonymous method

wordt. Bijvoorbeeld: ***btnMessage.Click += delegate { MessageBox.Show("Hello world!"); };*** De standaard parameters voor de eventhandler zijn niet verplicht zoals blijkt uit het voorbeeld. Ze zijn bewust niet meegegeven, omdat ze in het codeblock niet worden gebruikt. Als je ze wel wilt meegeven, omdat je ze nodig hebt, ziet het voorbeeld er als volgt uit.

Framework aanpassingen
Debugger datatips
Deze extensie op de debugger werkt voor zowel Visual Basic .NET als C#. Het biedt je de mogelijkheid complexe types zoals array's en datasets op een eenvoudige wijze te inspecteren.

Het Immediate-window
Dit venster kan in Visual Studio 2005 worden gebruikt voor het aanroepen van methoden zonder het project te compileren en te starten; dus rechtstreeks vanuit de designtime-omgeving. Deze feature is beschikbaar in zowel C# als Visual Basic .NET.

Windows Forms-aanpassingen
Er zijn te veel aanpassingen om alles hier te noemen, maar deze twee productiviteitaanpassingen wil ik even aanhalen. Ten eerste: alignment-tools waarmee je eenvoudig controls kunt plaatsen en uitlijnen op een form. Ten tweede: inline property-editing, waarmee je in één keer een property van al je controls op een form kunt wijzigen. Deze mogelijkheden zijn beschikbaar in zowel Visual Basic .NET als C#.

Intellisense / Code-snippets
Je kunt in de Visual Basic- en de C#-editor een code-snippet kiezen uit een hiërarchische lijst. De keuze resulteert in een insert van een codeblock in je programma met daarin placeholders op plekken waar je zelf nog wijzigingen moet aanbrengen om het ook in je eigen project te laten werken. Dit scheelt niet alleen typen, het maakt het overstappen van Visual Basic versie x naar Visual Basic .NET eenvoudiger. Je bent niet alleen in staat code-snippets / expansions te gebruiken, maar je kunt ook je eigen snippets / expansions toevoegen aan de lijst van snippets.

Edit and Continue
De diehard Visual Basic-programmeur zal blij zijn met de terugkeer van deze feature. Let wel, er zullen wijzigingen zijn waardoor je gedwongen zult worden terug te keren naar de designer en je project opnieuw te builden. Maar voor alle andere wijzigingen kun je vanuit de breakmode je code aanpassen en daarna de applicatie-execution verder laten gaan waar hij is onderbroken. Dit is echter geen reden om te kiezen voor Visual Basic .NET in Visual Studio 2005, omdat deze feature ook in C# wordt geleverd in de aankomende release.

Click once installation
De ingebruikname van .NET-applicaties is vanaf dag één eenvoudiger geweest dan het uitrollen van een COM+-based applicatie. Dit is nu verder vereenvoudigd door click once. Dit biedt de mogelijkheid om het installeren en updaten van applicaties te laten plaatsvinden vanaf een centrale plek op het netwerk of via internet. Deze feature is beschikbaar voor zowel de C#-ontwikkelaar als de Visual Basic-ontwikkelaar in Visual Studio 2005.

Making controls databound
In Visual Studio 2005 is het nog eenvoudiger geworden om data aan je controls op een form te binden. Je kunt simpelweg de velden slepen uit het datasource-window. Dit window 'managed'

Visual Basic .NET	C#
Dim InvoiceList As New List(of Invoices) InvoiceList.Add(New Invoice("2004001")) InvoiceList.Add(New Invoice("2004002")) InvoiceList.Add(New Invoice("2004003"))	List<Invoices> InvoiceList = new List<Invoices>(); InvoiceList.Add(new Invoice("2004001")); InvoiceList.Add(new Invoice("2004002")); InvoiceList.Add(new Invoice("2004003"))

Tabel 4. Generics: Strong Typed Collections

al de datasources van je project. Je hoeft dit dus niet meer zelf in code te doen. Dragging en dropping field op controls is voldoende. Visual Studio 2005 zorgt voor het zetten van de juiste properties van de control.

Partial type / partial classes
Visual Basic .NET en C# stellen je in staat partial classes te maken. De class-opbouw wordt verdeeld over meer files. Het grootste voordeel hiervan is dat codegenerators zoals Visual Studio de generated code-sections in aparte files kan plaatsen. Hierdoor verdwijnt de generated code-region uit je sourcecode-file. Partial classes worden gedeclareerd door het keyword partial toe te voegen aan de class-declaratie, bijvoorbeeld: ***partial class ClassX(){…};***

Generics
In zowel Visual Basic .NET als C# zijn generics toegevoegd. De Namespace System.Collections.Generics biedt je de mogelijkheid om strong-typed collections te definiëren. Deze namespace biedt de mogelijkheid om strong-typed collections te maken van de List, Dictionary, SortedDictionary, Stack en Queue. En als je er behoefte aan hebt kun je ook een eigen generic class definiëren.

Ondanks dat je waarschijnlijk niet voor iedere applicatie strong-typed collections nodig hebt, is het goed te weten dat ze er zijn. Als je een type-safe collection nodig hebt, maak dan gebruik van de mogelijkheden die worden geboden door de generic classes.

C# versus Visual Basic .NET

Uit het artikel kan worden geconcludeerd dat de verschillen klein zijn. De talen en de IDE's van beide zijn in Visual Studio 2005 nog verder naar elkaar toegegroeid. Het maken van een keuze is dus geen echte noodzaak. Een ontwikkelaar is in staat zichzelf redelijk snel een andere syntax eigen te maken. Zeker in een .NET-omgeving waarbij de talen gebruik maken van een eenduidig onderliggend Framework dat voor elke taal gelijk is. Mocht je toch een keus kunnen en willen maken, dan kun je jouw keus baseren op de volgende criteria:

- Bestaande kennis en ervaring in het bedrijf.
- Visual Studio .NET ondersteunt zowel Visual Basic als C# goed. Dit is echter niet het geval met third party tool-leveranciers. Deze leveranciers ondersteunen op het moment eerder C# dan Visual Basic (C# Builder van Borland bijvoorbeeld). Het wordt gezegd dat de overgang van C# naar Visual Basic .NET eenvoudiger is dan de andere kant op; dit geeft ook aan dat de C# leercurve iets steiler is dan die van Visual Basic.
- Voor C# zijn er meer resources zoals: communities, code samples, websites, publicaties, enzovoort.
- Verwachting van beschikbaar personeel nu en in de toekomst. Op dit moment zijn er meer Visual Basic-ontwikkelaars, maar de Gartner Group verwacht op termijn meer C#- dan Visual Basic-ontwikkelaars.
- Microsoft ontwikkelt veel nieuwe producten of nieuwe versies van reeds bestaande producten in C#; bijvoorbeeld BizTalk 2004.
- Voorkeur van opdrachtgevers.
- Noodzaak voor unmanaged code.
- Persoonlijke voorkeur.

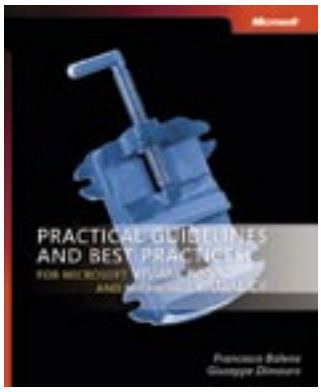
Zal C# sneller of juist langzamer vernieuwingen kunnen implementeren dan Visual Basic doordat de taal is gedeponeerd bij de standaardisatiecommisie ECMA? In theorie zou het antwoord kunnen zijn: langzamer. In de praktijk zal Microsoft haar paradepaard natuurlijk niet laten achterlopen bij de overige .NET-talen. Zaken als generics en partial types zijn reeds geruime tijd geleden bij de ECMA aangeboden. Daarnaast zal er voor elke feature ondersteuning moeten zijn in het onderliggende Framework, dus mocht er al sprake zijn van 'ontbrekende' functionaliteit in een van de talen dan zal het niet lang duren voordat het wordt toegevoegd.

Remi Caron is CTO bij Wantit B.V. in Haarlem. Hij is sinds 1989 actief in de automatisering. Wantit vervaardigt standaard producten, geeft trainingen en is een partner voor coaching-trajecten waarbij de 'oude programmeeromgeving' wordt vervangen door .NET. Sinds 2003 is hij MVP for .NET.

Nuttige internetadressen

http://msdn.microsoft.com/vbasic/
http://msdn.microsoft.com/vcsharp/
http://msdn.microsoft.com/library/en-us/dnvs05/html/languageenhancements.asp
http://msdn.microsoft.com/vcsharp/team/faq/vb/default.aspx

(advertentie Microsoft Press)



Practical Guidelines and Best Practices for Microsoft Visual Basic and Microsoft Visual C# Developers
ISBN: 0-7356-2172-1
Auteur: Francesco Balena and Giuseppe Dimauro
Pagina's: 560

MCAD/MCSD Self-Paced Training Kit: Implementing Security for Applications with Microsoft Visual Basic.NET and Microsoft Visual C# .NET
ISBN:0-7356-2121-7
Auteur: Tony Northrup
Pagina's: 640