



Ein Blick auf .NET

2001 Revolution

NET - ein **neues Schlagwort**, das vor allem Programmierer heiß **diskutieren**, die sich in der Windows-**Welt bewegen**. Doch was bringt die Zukunft, wann und **vor allem wie?** Fragen, die dieser Überblick klären will.

THOMAS WÖLFER

Wenn es nach Microsoft geht, soll .NET (DOT NET) eine historische Wende für die Windows-Plattform einläuten. Bei .NET handelt es sich um eine Sammlung aus Tools und Libraries, die man am besten als Laufzeitumgebung für die Entwicklung von Web-Anwendungen beschreibt. Den Schwerpunkt setzte Microsoft dabei auf die Entwicklung für firmenweite Anwendungen. Allerdings verändert .NET die Welt der Windows-Programmierung derart, dass praktisch kein Windows-Entwickler daran vorbeikommen wird. Auf die eine oder andere Art wird .NET, sofern es sich den Wünschen Microsofts entsprechend entwickelt, die Entwicklung von Windows-Anwendungen grundlegend verändern.

QUICK INDEX

- ▶ **Die Laufzeitbibliothek**
Den Kern von .NET bildet fraglos die CLR (Common Language Runtime).
- ▶ **Microsoft verwendet P-Code**
NET Binaries werden in dieser Form, also als MSIL, auf der Platte gespeichert.
- ▶ **C++ mit Verwaltung**
Die Speicherverwaltung wird von Laufzeitsystem erledigt.
- ▶ **Einführung in C#**
Werfen Sie einen Blick auf das schärfste aller Zeiten.

Das .NET SDK ist seit der Developer-Konferenz im Juli 2000 in Florida verfügbar. Zwar nur in Beta- und anderen Vorabversionen, doch auch ganz "normale" MSDN-Abonnenten haben mittlerweile Zugriff auf die ersten Komponenten. Man muss kein Microsoft-Intimus mehr sein, wenn man sich bereits heute ernsthaft mit der neuen Plattform beschäftigen möchte.

■ Die Laufzeitbibliothek

Den Kern von .NET bildet fraglos die CLR (Common Language Runtime). Dahinter verbirgt sich eine Laufzeitbibliothek, die öffentliche Interfaces exportiert, die im großen und ganzen allen .NET-Komponenten zur Verfügung stehen. Zu dieser Laufzeitbibliothek gehören unter anderem folgende Komponenten und Eigenschaften:

- Ein objektorientiertes Programmiermodell mit Vererbung, Polymorphismus und Ausnahmebehandlung.
- Ein gemeinsames Typensystem für alle beteiligten Programmiersprachen.
- Eine Klassenbibliothek, die nicht nur den Großteil der Win32 API abbildet, sondern auch weitere Technologien wie XML-Parser zur Verfügung stellt.
- Eine dramatisch verbesserte Debugging-Unterstützung über mehrere Sprachen, Rechner und Plattformen hinweg. (So kann eine HTML-Seite mit JavaScript debugged werden, die ein in C++ geschriebenes Objekt verwendet, das auf einem anderen Rechner läuft. Wird von diesem Objekt aus eine Datenbank-

abfrage an einen weiteren entfernten Rechner gesendet, so kann die Ausführung der SQL-Statements auf der dritten Maschine gebugged werden: Alles innerhalb einer DIE und eines Debuggers.)

- Ein systemweiter Garbage Collector. Abgesehen von der Java VM im Internet Explorer ist der Garbage Collector eine Neuheit für Microsoft-Programmiersprachen. Bisher stellten diese entweder keinen dynamischen Speicher zur Verfügung, oder der Programmierer war selbst für das Memory-Management verantwortlich. Bevor die C- und C++-Programmierer nun die Hände über dem Kopf zusammenschlagen: Der Garbage Collector ist keineswegs Zwang. Wer mehr Effizienz wünscht und sich dafür lieber selbst um die Speicherverwaltung kümmert, kann dies auch innerhalb von .NET tun.

■ Microsoft verwendet P-Code

Ein weiteres Novum bei .NET, zumindest in diesem Ausmaß, ist die Verwendung von P-Code, oder, wie die neue Bezeichnung lautet "Microsoft Intermediate Language" (MSIL). .NET Binaries werden in dieser Form, also als MSIL, auf der Platte gespeichert. Erst wenn das MSIL-Modul geladen wird, wird es von einem Just-In-Time-Compiler in nativen Code umgewandelt. Auch hier gilt aber: kein Zwang. Wer lieber selbst auf seinem Entwicklungssrechner nativen Code erzeugen lassen möchte, kann dies weiterhin tun.

Praktisch ist IL-Code hingegen sehr: Jede Sprache, die in der Lage ist, IL Code zu generieren, kann mit dem Typensystem jedes anderen .NET-Moduls im System interagieren. Dadurch wird es möglich, von Klassen zu erben, die in anderen Programmiersprachen geschrieben wurden. Dies ist im Wesentlichen der Trick, der dazu führt, dass die



gemeinsame Laufzeitumgebung im Prinzip von allen Sprachen parallel benutzt werden kann. Die benötigten Spezifikationen werden von Microsoft weitergegeben, so dass jeder Hersteller für sich entscheiden kann, ob er .NET unterstützt oder nicht. Im SDK selbst und vermutlich auch im kommenden Visual Studio.NET – wird zumindest eine Variante von Visual Basic, eine von Visual C++ und eine von C# (nicht "Cash" ausgesprochen, sondern laut offiziellem Wortlaut "C sharp") enthalten sein, die diese Möglichkeit bieten.

■ C++ mit Verwaltung

Die .NET-Variante von VC++ trägt den Namen "Managed C++". Die Bezeichnung "Managed" spielt dabei auf zwei unterschiedliche Dinge an: Zum einen kann mit dieser VC++-Variante "Managed Code" erzeugt werden – also Code, der von der Ausführungsumgebung des .NET-Frameworks verwaltet wird. Zum anderen ist es damit möglich, den Garbage Collector zu verwenden: Die Speicherverwaltung wird also vom Laufzeitsystem verwaltet.

Die Common Runtime erwartet, dass jeder Speicherbereich, der von mehreren Komponenten verwendet wird, von einem globalen und der Runtime verwalteten Heap alloziert wird. Dies ist im Wesentlichen nichts anderes als die Speicherverwaltung der C-Laufzeitbibliothek – allerdings mit dem Vorteil, dass nicht länger benötigter Speicher von der Laufzeitumgebung automatisch freigegeben wird. Dass dieses automatische Freigeben innerhalb einer Sprache wie C++ nicht ganz unkompliziert ist, ist offensichtlich: Schließlich ist ein C++-Destruktor nicht nur für die Freigabe von Speicher verantwortlich, sondern muss sich auch um andere Ressourcen kümmern, wie offene File-Handles und ähnliches. Dementsprechend kann man sich vorstellen, dass der Collector ein gewisses Maß an Overhead und dadurch Performance-Einbußen mit sich bringt. Aus diesem Grund ist es in Managed C++ möglich, frei zwischen "managed" und "non-managed" Code zu wählen. Eine Klasse kann sich aus Managed-Code zusammensetzen, die andere Klasse nicht.

Bei Managed C++ handelt es sich um eine Erweiterung von Standard C++, denn der Compiler muss in der Lage sein, die Features der Runtime-Umgebung zu unterstützen. Dazu braucht man neue Schlüsselwörter und "Attribute" für

Klassen, Funktionen und Daten: Wer sich also auf Managed C++ einlässt, der verlässt die Welt von Standard C++.

Bei all diesen Veränderungen stellt sich die Frage, was mit den bisherigen objekt-nahen Technologien passiert: Wo platziert sich COM und ATL im .NET Umfeld? Einfache Antwort: gar nicht.

Auch wenn man das zunächst vermuten möchte, ist .NET eben nicht auf COM basiert. Die Zeit der IUnknown Interfaces ist damit weitgehend vorbei. Dies bedeutet natürlich nicht, dass COM nicht länger unterstützt wird oder ATL völlig von der Bildfläche verschwindet. Es sind allerdings Tools, die ihren Höhepunkt bereits überschritten haben. Ihre Funktionalität wird durch .NET-Komponenten ersetzt. Bis das vollständig der Fall ist, kann der Entwickler die "COM Interop"-Schnittstelle verwenden, die bidirektionale Zusammenarbeit zwischen .NET und COM Objekten ermöglicht. Investitionen in COM Objekte sind also nicht verloren. Und man kann über Microsoft sicherlich sagen, was man will, aber Rückwärtskompatibilität ist beim Software-Riesen ganz sicher.

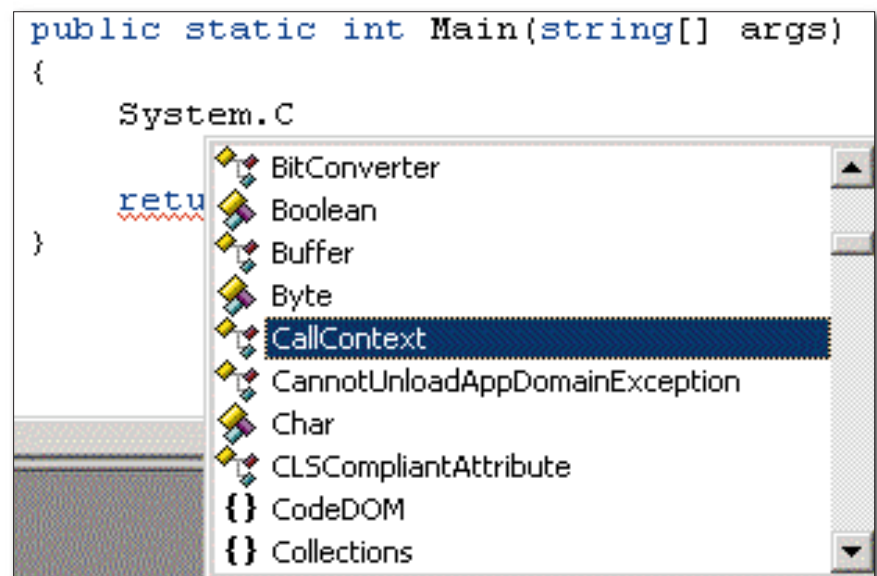
Völlig unabhängig von neuen Technologien und Entwicklungstools bietet .NET im Wesentlichen eine wirklich gute Veränderung: Es kapselt die komplette Win32 API in einer objektorientierten Schicht. Das macht .NET tatsächlich zu einer neuen Plattform. Vom Assembly-Level Interface in MS-DOS und dem C-Interface in Win16

und Win32 bietet .NET erstmals ein objektorientiertes Interface zum Betriebssystem. Allein diese Tatsache wird unausweichlich dazu führen, dass .NET mit seiner Einführung schlagartig große Bedeutung erlangen wird. Das neue Framework ändert für den Programmierer in einem Microsoft-Umfeld alles, wenn er sich darauf einlässt.

■ Einführung in C#

C# wird als Teil von Visual Studio 7 ausgeliefert werden. Wer über ein MSDN Abo verfügt kann bereits heute mit der ersten Beta-Version dieser Sprache arbeiten. Dabei kommt C# zusammen mit Visual Basic, Visual C++ sowie VBScript und JavaScript als eine Sprache daher, die das "Common Language Subset" unterstützt und dadurch auch die "Common Runtime Library" (siehe dazu der Beitrag "Der Grundgedanke von .NET" in dieser Sonderheft) verwenden kann. Im Wesentlichen bedeutet dies, dass ein Programmierer der C# verwendet Zugriff auf die allermeisten Objekte und Funktionen aus den weitaus älteren Sprachen wie VB und VC++ hat, obwohl C# selbst keine Klassenbibliothek zur Verfügung stellt.

Ähnlich wie bei C++ wird ein C# Programm in einem oder mehreren Textdateien abgelegt die nacheinander übersetzt und schließlich zu einem fertigen Programm per Linker zusammengebunden werden. Anders als bei C++ gibt es keinerlei include-Dateien und auch keine Vorwärtsreferenzen. Um Objek-



OHNE DIE KONTEXTBEZOGENE Hilfe welche die verfügbaren Eigenschaften und Objekte in einer Liste anbietet, wird die Arbeit schnell unübersichtlich: Zu groß ist das Angebot an Funktionalität.



te aus der Laufzeitbibliothek zu verwenden wird das "using" Keyword verwendet – Java Programmierern wird dies bekannt vorkommen.

Das typische "Hello World" Programm hat in C# den folgenden Aufbau:

```
using System;
class Hello {
    static void Main() {

        System.Console.WriteLine("Hello
        World");
    }
}
```

Der Ausdruck "using System" referenziert einen Namespace mit der Bezeichnung "System": Dieser wird von der Microsoftschen .NET Framework Klassenbibliothek zur Verfügung gestellt. Dieser Namespace enthält unter anderem die "Console" Klasse, die in Main() verwendet wird. Namespaces sind ein wichtiges Element von .NET und damit auch von C#. Sie stellen den primären Ordnungsmechanismus der Klassenbibliothek zur Verfügung.

Der Eintrittspunkt in ein C# Programm ist immer eine statische Funktion mit dem Namen Main() – wie bei C kann diese auch Parameter und einen Rückgabewert haben. Die Funktion Main() befindet sich allerdings nicht auf globaler Ebene innerhalb des Pro-

LISTING 1

```
Namespace ConsoleApp {
    using System;
    public class Klasse1 {
        public Klasse1 { /* konstruktor - wird hier nicht benötigt */}
        public static int Main( string[] args) {
            foreach( string s in args) {
                System.Console.WriteLine( s);
            }
        }
    }
}
```

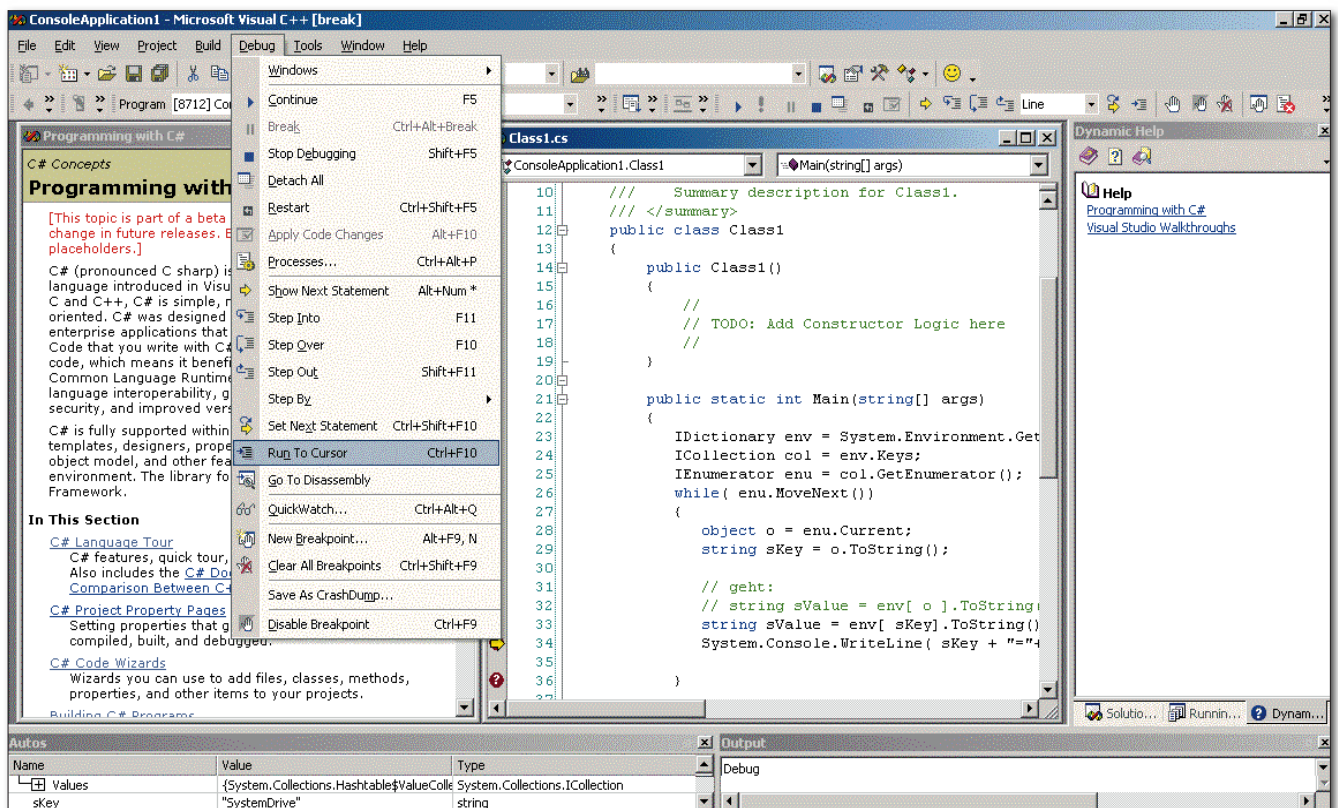
gramms – sie muss in C# immer in einer Klasse eingebettet sein. Bevor im weiteren Verlauf des Beitrage mehr auf einige Sprachdetails eingegangen wird, hier noch ein paar kurze Beispiele für C# Konsolen-Programme (Die Programmierung von Diensten, Web-Diensten und Windows Programmen ist mit C# natürlich auch möglich.).

Ein C# Programm das alle an das Programm übergebene Parameter anzeigt würde folgendermaßen aussehen (Listing 1):

In diesem Beispiel wird deutlich, das Main() als Eintrittspunkt genauso funktioniert wie dies auch in normalen C/C++ Programmen der Fall ist. Bereits anders an dieser Stelle: Der Datentyp "string" wird benutzt. Bei "string" handelt es sich um einen Typ der aus der CRT stammt und in allen .NET Spra-

chen verwendet werden kann, so auch in C#. Ebenfalls sieht man auf dem ersten Blick das "foreach" Keyword. Dabei handelt es sich um ein für C/C++ Programmierer neues Schlüsselwort: Es tut genau das, was man vermuten würde: Geboten wird eine sehr einfache Methode über eine Sammlung (hier eine Liste) zu iterieren. Ob foreach auf eine Sammlung anwendbar ist oder nicht ist vom Implementor der Sammlung abhängig. Soll foreach funktionieren, so muss dies explizit durch den Programmierer der Sammlung vorgesehen worden sein.

Ein weiteres Beispiel, bei dem noch weitere Objekte aus dem "System" Namespace verwendet werden. Das folgende Programm gibt alle Environment-Variable bzw. deren Inhalt am Bildschirm aus (Listing 2): Hier werden



C# IN AKTION – seit dem Beta 1 von Visual Studio.NET kann für die Entwicklung von C# auch die Visual Studio IDE verwendet werden.



direkt mehrere Objekte bzw. Interfaces zu diesen Objekten verwendet: IDictionary ist ein Hash-Objekt (Eine Map), ICollection eine generische Sammlung und IEnumerator ein Enumerationsobjekt mit dem über Sammlungen iteriert werden kann. Aus der Sicht eines "C" Programmiers mag der Code ein wenig verwirrend erscheinen. Einen C++ Programmierer wird auf Anhieb klar sein, was die einzelnen Elemente tun. Wichtig dabei: Keines der Objekte stammt aus einer "C#" Klassenbibliothek. Sie werden samt und sonders von der Common Runtime Library zur Verfügung gestellt. Zunächst erfragt das Programm eine Map (IDictionary) in der alle Environment-Variable enthalten sind. Die geschieht mittels des Aufrufes von GetEnvironmentVariables() und diese Funktion ist Teil des "Environment" Objektes aus dem "System" Namespace. Diese Map kann nun mittels "Keys" nach den in ihr enthaltenen Schlüsseln gefragt werden – selbige werden in einer Liste geliefert, und diese List kann ihrerseits mit einem Enumerationsobjekt (GetEnumerator()) iteriert werden. Mit neu.GetCurrent() wird dabei eine Kopie des "aktuellen" Elementes aus der Liste erfragt. Dieses wird nicht als "string" sondern als "object" (also generisch) geliefert. Dieses generische Objekt kann als Eingangswert für den Lookup in der Map (env[sKey]) verwendet werden, das wiederum als "object" gelieferte Resultat wird mit ToString() in einen String konvertiert. Schließlich wird dieser String auf der

WEITERE INFORMATIONEN

Ein Großteil der Informationen über .NET ist online verfügbar, und wöchentlich kommen neue Artikel, Werkzeuge und Hilfsmittel hinzu. Hier einige Websites, die bereits umfassende Informationen über .NET und zugehörige Dienste bieten:

- Das Preview-SDK gibt es unter <http://msdn.microsoft.com/dotnet>. Allerdings ist das SDK relativ groß. Wer auf die 80 MByte Download zunächst verzichten möchte, erhält auch an anderer Stelle Informationen zu .NET.
- Eine Art FAQ findet sich unter www.devx.com/dotnet/resources. Dort sind weitere Artikel und Hintergrundberichte zu .NET erhältlich. Eine ausführliche Beschreibung zu C# findet sich unter www.genamics.com/visualj++/csharp_comparative.htm.

Konsole ausgegeben und hier sieht der C Programmierer erneut etwas, das den C++ Programmierer nicht weiter verwundert: Die Strings können mit dem "+" (Additions) – Operator verknüpft werden. Auch hier gilt: Der Additions-Operator stammt nicht etwa aus einer #include-Datei von C#, sondern ist als Teil der "string" Klasse in der CRT definiert.

C# kommt mit einer ganzen Reihe von vordefinierten Typen, nicht zuletzt die bereits vorgestellten Referenz-Typen "object" und "string". Darüber hinaus existieren natürlich die aus C++ bekannten atomaren Typen wie "int", "char" und so weiter. Eine besondere Stellung hat "bool". Anders als man das von C++ gewohnt ist, ist "bool" nicht vom Typ "int" und kann auch nicht automatisch konvertiert werden. Dies eliminiert den folgenden häufig vorkommenden Fehler:

```
int i = Function();
if( i = 0) Function2(); // fehler:
es sollte i == 0 heissen.
```

In C# ist der Ausdruck (i = 0) vom Typ "int" – benötigt wird aber ein vom Typ "bool": Der Compiler kann also eine Fehlermeldung erzeugen.

Alle vordefinierten Typen sind in C# in Wirklichkeit nur abkürzende Schreibweisen für Typen welche die Laufzeitumgebung zur Verfügung stellt: "int" ist die Abkürzung für "System.Int32". Für alle Typen, also auch für die eingebauten, ist Operator-Overloading verfügbar und wird auch verwendet. So haben die

```
17 public static int Main(string[] args)
18 {
19     System.Console.WriteLine(
20         "1 of 18 void Console.WriteLine(string format, object[] arg)
21         format: Formatting string.
22     }
```

MIT 18 MÖGLICHEN FORMEN des Aufrufes ist die WriteLine() Methode für viele Fälle gerüstet. Gut, dass man alle möglichen Formen direkt im Quellcode-Editor anzeigen lassen kann.

Operatoren == und != in Abhängigkeit vom Typ eine unterschiedliche Semantik. Für einen Ausdruck vom Typ "int" gilt Gleichheit, wenn die verglichenen Variablen den gleichen Integer-Wert haben, während bei Objekten (object) Gleichheit gilt, sofern beide Objekt-Variable das identische Objekt referenzieren. Bei "strings" hingegen gilt Gleichheit, sofern beide Strings gleich lang sind und an allen Positionen innerhalb des Strings das gleiche Zeichen stehen haben. Wie man sieht: C++ ist gar nicht so weit entfernt.

Im großen und ganzen ist C# für einen C oder C++ Programmierer in extrem kurzer Zeit in den Griff zu bekommen. Das einzige wirkliche Problem resultiert aus einem der großen Vorteile von C#: Der Zugriff auf die schier endlose Zahl an Objekten mit den zugehörigen Methoden und Diensten macht die ansonsten sehr übersichtliche Sprache in der Benutzung zeitweilig ein bisschen unhandlich. Mit "C" konnte man Programme noch einfach so "herunterschrauben" in "C++" musste man immer wieder in die Dokumentation der verwendeten Klassenbibliothek sehen. In C# ist ein dauernd offenes Hilfefenster eigentlich Pflicht, denn die Fülle an zur Verfügung stehender Funktionalität ist ohne Volltextsuche praktisch nicht zu überschauen.

UR

LISTING 2

```
Namespace ConsoleApp {
    using System;
    using System.Collections;
    public class Klasse1 {
        public Klasse1 { /* konstruktor - wird hier nicht benötigt */}
        public static int Main( string[] args) {
            IDictionary env = System.Environment.GetEnvironmentVariables();
            ICollection col = env.Keys;
            IEnumerator enu = col.GetEnumerator();
            while( enu.MoveNext())
            {
                object o = enu.Current;
                string sKey = o.ToString();
                string sValue = env[ sKey].ToString();
                System.Console.WriteLine( sKey + "=" + sValue + "\r\n");
            }
        }
    }
}
```