.NET

aus Wikipedia, der freien Enzyklopädie

.NET [ˈdɔtnɛt] bezeichnet eine von Microsoft entwickelte Software-Plattform zur Entwicklung und Ausführung von Anwendungsprogrammen. .NET besteht aus einer Laufzeitumgebung (*Common Language Runtime*), in der die Programme ausgeführt werden, sowie einer Sammlung von Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen (*Services*). .NET ist auf verschiedenen Plattformen verfügbar und unterstützt die Verwendung einer Vielzahl von Programmiersprachen. .NET-Programme werden zunächst in eine Zwischensprache (*Common Intermediate Language*) übersetzt, bevor sie von der Laufzeitumgebung ausgeführt werden. Diese Übersetzung geschieht in der Regel mithilfe eines Just-In-Time-Compilers. Für die Entwicklung von .NET-Programmen vertreibt Microsoft die Entwicklungsumgebung Visual Studio.

.NET Framework

Microsoft .NET-Logo

Basisdaten

Entwickler	Microsoft
Aktuelle Version	4.0[1] (21. Februar 2011)
Aktuelle Vorabversion	4.5, PP[2][3]
Betriebssystem	Windows
Kategorie	Plattform
Lizenz	EULA (proprietär)
Deutschsprachig	ja

Offizielle Webseite (http://msdn.microsoft.com/de-de/netframework/default.aspx)

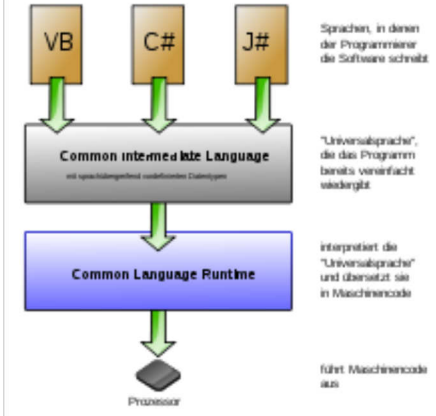
Inhaltsverzeichnis

Verfügbarkeit, Standardisierung, alternative Produkte

.NET ist im vollen Umfang nur für Windows verfügbar. Am 17. Januar 2008 veröffentlichte Microsoft Teile des Quelltextes für Windows-Entwickler.[4] Große Teile von .NET, insbesondere die Laufzeitumgebung und die Klassenbibliotheken, wurden unter dem Namen Common Language Infrastructure (CLI) als ECMA-Standard normiert. Durch die Standardisierung der Laufzeitumgebung gibt es alternative Produkte, die Software, die mit .NET erstellt wurde, ausführen beziehungsweise Software für .NET erstellen können. Viele Programme, die mit .NET erstellt wurden, laufen beispielsweise dank der durch das Open-Source-Projekt Mono zur Verfügung gestellten Software auch auf Unix-basierten Betriebssystemen wie z. B. Linux.

Konzept

Die .NET-Plattform ist die Umsetzung des Common-Language-Infrastructure-Standards (CLI) und stellt mit diesem eine Basis zur Entwicklung und Ausführung von Programmen dar, die mit unterschiedlichen Programmiersprachen auf verschiedenen Plattformen erstellt wurden. Hauptbestandteile sind die (objektorientierte) Laufzeitumgebung *Common Language Runtime* (CLR), die *Base Class Library* (BCL) sowie diverse Hilfsprogramme zum Beispiel zur Rechteverwaltung.



Das zugrundeliegende Basisprinzip

Mit .NET löste Microsoft zuvor eingesetzte Softwareentwicklungskonzepte wie das Component Object Model ab, bis es COM in einer erweiterten Form unter dem Namen Windows Runtime reaktivierte. Seitdem plant Microsoft den parallelen Einsatz beider Frameworks für die Betriebssystemgeneration um Windows 8.

CLR, CIL

Die *Common Language Runtime* (CLR) ist die Laufzeitumgebung von .NET und stellt somit den Interpreter für den standardisierten Zwischencode, die *Common Intermediate Language* (CIL), dar. Die CIL hieß früher *Microsoft Intermediate Language* (MSIL), wurde aber im Rahmen der Standardisierung durch die Ecma International umbenannt. Für sie wurde ein sprachübergreifendes System von objektbasierten Datentypen definiert, so dass alle Hochsprachen, die sich an den Common Language Infrastructure-Standard (CLI) halten, gültigen CIL-Bytecode erstellen können.

.NET wurde von Anfang an dafür entwickelt, dass Programmierer in unterschiedlichen Programmiersprachen arbeiten können. Jede dieser Hochsprachen wird von .NET dann in die CIL übersetzt.

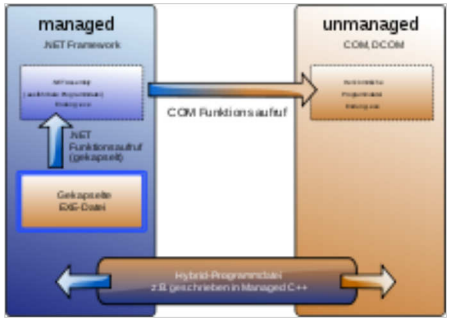
Strategie, Nutzen und Trends

Das Besondere an der CLR ist weniger die technische Innovation als vielmehr die strategische Entscheidung von Microsoft für ein laufzeitbasiertes System. Es soll unter anderem helfen, Programmierfehler zu vermindern. Gleichzeitig entschied sich Microsoft *gegen* die traditionelle, bisher angewandte direkte Kompilierung in den Maschinencode des Zielsystems. Zusammen mit der Marktmacht von Java und dem Erfolg von Skriptsprachen ist damit ein Trend zu identifizieren. Dieser stellt einen Bruch mit den direktkompilierenden Programmiersprachen (insbesondere C++ und C) dar. Damit entfernt sich die Softwareentwicklung, insbesondere die Anwendungsentwicklung, insgesamt von dem Augenmerk auf Ausführungsgeschwindigkeit, die angesichts einer immer höheren Rechengeschwindigkeit zunehmend in den Hintergrund rückt. Vielmehr bemüht man sich nun um mehr Effizienz in der Programmentwicklung.

Mittels sog. „Reflection“ ist es möglich, zur *Laufzeit* Programmcode über ein Objektmodell zu generieren und es direkt im Speicher in lauffähigen Code zu überführen.

Managed und Unmanaged

Die .NET-Terminologie unterscheidet dabei zwischen *Bytecode*, welcher von der CLR verwaltet und in Maschinensprache umgesetzt wird – sogenannter *Managed Code* (vom englischen *managed code* für „verwaltete Maschinensprache“) –, und Teilen, die nicht innerhalb der CLR ausgeführt werden (*unmanaged*). Daneben gibt es noch die Möglichkeit in .NET *unsicheren Code* zu schreiben, um weiterhin z.B. klassische Zeiger-Operationen direkt auf einem Speicherbereich durchführen zu können.



Interop-Technik

Mit Hilfe der *Interop-Technik* lassen sich alle klassischen, binär kompilierten Windows-Bibliotheken mit .NET-Kapseln (sogenannten „Wrappern“) versehen und danach deren Programmfunktionen wie *normale* .NET-Programmfunktionen aufrufen. Technisch gesehen gibt die CLR allerdings im Moment des Aufrufs einer Funktion einer nicht überwachten DLL einen großen Teil der Kontrolle über den Programmfluss ab.

Umgekehrt lassen sich auch .NET-Funktionen wie COM-Funktionen aufrufen. Damit soll eine fließende Migration von Software-Projekten auf .NET ermöglicht werden und die Integration von .NET-Modulen in eine bestehende Umgebung erleichtert werden.

Sicherheit

Eines der wichtigsten Konzepte von .NET ist die Sicherheit. Das Sicherheitskonzept beginnt bei Mechanismen, die die Identität des Programmherstellers gewährleisten sollen (Authentizität), geht über in solche zum Schutz der Programme vor Veränderung (Integrität) und reicht bis hin zu Techniken, die den Ort der Herkunft bzw. Programmausführung (zum Beispiel das Internet) einbeziehen. Es gibt sowohl ein

codebasiertes (*Code-based Security*) als auch ein nutzerbasiertes (*Role-based Security*) Sicherheitsmodell.

Attribute

Eine programmiertechnisch interessante Neuerung von .NET ist die Einführung von *Attributen*: gekennzeichnete Metadaten als Bestandteil der Programmiersprache. Beispielsweise können im Rahmen der komponentenbasierten Programmierung Komponenteneigenschaften ausgedrückt werden. Für die Verteilung, Installation und Konfiguration, für die Sicherheit, für Transaktionen und andere Programme können dem Code beschreibende Eigenschaften hinzugefügt werden.

Innerhalb eines Programmes kann mit Hilfe von Reflection auf die Attribute eines .NET-Programms, *Assembly* genannt, und die in ihr enthaltenen Elemente zugegriffen werden.

Dieses Konzept wurde später unter anderem in Java übernommen, wo es in Form sogenannter Annotations verwirklicht ist.

Verteilte Programmierung und *Web Services*

Microsoft bietet hier nicht einen übergreifenden Ansatz, sondern gleich (mindestens) drei unterschiedliche, mit jeweils eigenen Vor- und Nachteilen:

- eine moderne Implementierung des neuen Industriestandards *Webservices* (manchmal auch *„XML Web Services“* genannt): Eine Struktur für verteilte Programme, im LAN, aber mit zunehmender Bedeutung auch für nichtkommerzielle Anwendungsgebiete im Internet
- die Windows Communication Foundation
- die .NET Remoting Services
- die *.NET Enterprise Services*

Letztere sollen eine Konkurrenztechnik zu der erfolgreichen JEE werden.

Die verschiedenen Strukturen innerhalb von .NET 2.0 sind, trotz breiter Auswahl und technisch teilweise guter Ansätze, nicht ausgereift. Insbesondere die *.NET Remoting Services* sollten nur dann eingesetzt werden, wenn man über ihre Schwächen und Einschränkungen genau Bescheid weiß. In Bezug auf Web Services bietet .NET jedoch eine der technisch führenden Implementierungen an.

Mit der Fertigstellung des .NET Frameworks 3.0, mit dem auch die Windows Communication Foundation (*WCF*) Einzug in das Framework gehalten hat, wurden die geforderten Nachbesserungen implementiert. Insbesondere im Bereich des .NET Remoting wurde die Sicherheit stark verbessert und der Aufwand für die Nutzung der Dienste verringert.

Sprachneutralität und gemischtsprachige Programmierung

Die *Common Language Specification* (CLS) definiert als eine gemeinsame Untermenge den Bytecode der CIL, der von der virtuellen Laufzeitumgebung (VM) in den Maschinencode der Zielmaschine übersetzt und ausgeführt werden kann. Somit ist es möglich, .NET mit verschiedenen, an die CLR angepassten Sprachen zu programmieren. Von Microsoft zum Beispiel schon im *Visual Studio* mitgeliefert sind das, neben der von Microsoft für .NET eingeführten Sprache C#, die Sprachen C++/CLI, das proprietäre Visual Basic .NET sowie J# (Aussprache: *dschey-scharp*; eine Portierung von Microsofts veränderter Java-Implementierung) und abschließend – nicht zu verwechseln mit J# – JScript .NET. Außerdem wurde mit Visual Studio 2010 die funktionale Programmiersprache F# eingeführt.

Insbesondere das vereinheitlichte Typsystem (*Common Type System*), das eine gemeinsame Schnittmenge an Datentypen beschreibt, sorgt für ein reibungsloses Zusammenspiel beim Aufruf von in einer anderen

Sprache geschriebenen Komponenten. Das stellt einen wichtigen Fortschritt dar, da man unter Visual Basic 6 unter Umständen gezwungen war, Funktionen, die nicht in Visual Basic implementiert werden konnten, in Visual C++ zu programmieren. In diesem Fall gab es immer Schwierigkeiten bei der Zuordnung der Datentypen von Visual Basic zu den entsprechenden Datentypen unter C++. Auch bei der Programmierung von COM-Komponenten in C++ musste man als Programmierer mit einem eingeschränkten Satz von Datentypen auskommen, die zur Automation benutzt werden konnten. Außerdem wurden Zeichenketten unter C++ und Visual Basic 6 intern unterschiedlich gespeichert, was die Programmierung erschwerte.

Die Vorteile der Unterstützung gemischtsprachiger Programmierung von .NET sind nicht unumstritten. Beispielsweise ist die Wartbarkeit eines Projektes, welches in mehreren Sprachen implementiert wurde, schlechter als bei der Entwicklung mit nur einer Sprache.

Neben den von Microsoft für die .NET-Plattform angepassten Sprachen *C#*, *Visual Basic .NET* und *C++/CLI* (*Managed C++*) werden weitere .NET-Sprachen von Drittanbietern zur Verfügung gestellt (zum Beispiel Delphi Prism von Embarcadero, aber auch weniger bekannte Sprachen wie APL von Dyalog).

Die für .NET bereitgestellte IDE von Microsoft, das Visual Studio .NET, bietet die Möglichkeit, weitere Sprachen von Drittanbietern in ein Projekt einzubinden und somit deren Funktionalität zu nutzen. Dass die Entwicklung in einer konsistenten Entwicklungsumgebung stattfindet und es nicht für jede Sprache eine eigene IDE gibt, ist zusätzlich von Vorteil.

Plattformunabhängigkeit

Die angestrebte Plattformunabhängigkeit ist unter .NET grundsätzlich möglich, Microsoft selbst hatte 2002 für die erste Version von .NET eine eingeschränkte (und nicht mehr aktuelle) .NET-Variante namens *Shared Source CLI* (*SSCLI*) für Mac OS und FreeBSD zur Verfügung gestellt. 2006 folgte die Version 2.0 von SSCLI für .NET 2.0, die aber nur noch auf Windows XP SP2 lauffähig ist.

Mehrere von Microsoft quasi unabhängige Open Source-Projekte haben sich einer entsprechend flexiblen Implementierung der Rahmenkomponenten auf Grundlage des ECMA-Standards angenommen. Die beiden wohl am weitesten entwickelten Projekte sind Mono, das vom Hersteller Ximian initiiert wurde, und das dotGNU-Projekt, welches an einer Portable.NET genannten Laufzeitumgebung arbeitet.

Beide Implementierungen sind jedoch noch nicht auf dem Entwicklungsstand des heutigen .NET. Zwar hat Mono mit der aktuellen Version 2.0 einen wichtigen Meilenstein, nämlich die Kompatibilität mit den nicht-windowsspezifischen Bibliotheken von .NET 2.0, erreicht. Auf der anderen Seite gibt es zum einen viele Programme, die P-Invoke oder COM Interop benutzen, d. h. auf Bibliotheken zugreifen, die nicht in IL-Code, sondern in normalem, prozessorspezifischen Assemblercode vorliegen. Zwar kann auch Mono auf Bibliotheken zugreifen, die in C oder C++ geschrieben sind, allerdings sind die meisten dieser Bibliotheken plattformabhängig. Weiterhin hat Microsoft mit .NET 3.0 und .NET 3.5 gravierende Weiterentwicklungen des Frameworks veröffentlicht, die von Mono bzw. dotGNU bis dato noch nicht oder nur teilweise implementiert wurden, aber in Arbeit sind. Explizit ausgenommen wurde die Windows Presentation Foundation, die auf absehbare Zeit nicht reimplementiert werden wird. Allerdings wird es trotzdem Unterstützung für XAML geben.

Laufzeitumgebung

Managed Code wird, wie oben erwähnt, von der Laufzeitumgebung *Common Language Runtime* (CLR) verwaltet. Diese virtuelle Maschine übernimmt die Anforderung und Freigabe von Speicher und anderen Ressourcen (automatische Speicherbereinigung, engl. *garbage collection*) und stellt sicher, dass geschützte Speicherbereiche nicht direkt angesprochen oder überschrieben werden können. Wie oben unter Sicherheit beschrieben, können auch Zugriffe auf Dienste, Dateisystem-Funktionen oder Geräte überwacht und, sofern sie gegen Sicherheitsrichtlinien verstoßen, von der CLR abgelehnt werden.

Ausführungsgeschwindigkeit

Für den Erfolg von .NET war und ist es wichtig, die Entwicklergemeinde von C++ für .NET zu gewinnen. Daher war Geschwindigkeit bei .NET von Anfang an ein wesentliches Entwurfsziel.

Durch verschiedene Techniken wird versucht, den negativen Einfluss der CLR auf die Ausführungsgeschwindigkeit möglichst klein zu halten. Zum Beispiel wurden analog zu Java sogenannte JIT-Compiler eingeführt, die einen Mittelweg zwischen Interpretation und Kompilierung gehen. Außerdem kann man mit .NET als Neuerung auch Programme in bereits kompiliertem Code, als sogenanntes *natives Image* installieren. Das wirkt sich insbesondere auf die erstmaligen Ladezeiten bei Programmen mit größeren Klassenmengen aus. Weiterhin kann der Speicherbedarf reduziert werden, wenn mehrere Programme dieselbe Assembly nutzen bzw. das Programm mehrfach gestartet wird (Terminalserver), da die nativen Images im Gegensatz zu JIT-Code zwischen den Programmen über gemeinsam genutzten Speicher (engl. *shared memory*) geteilt werden. Der Gewinn an Ausführungsgeschwindigkeit durch native Images muss durch sorgfältige Messungen („profiling“) analysiert werden. Der Einsatz von nativen Images erfordert weitere Planungsschritte bei der Entwicklung der Software, zum Beispiel eine sorgfältige Auswahl der DLL-Basisadresse der Assemblies, um eine Relokation der DLLs zu verhindern. Schließlich müssen die Assemblies auch im GAC installiert werden, um anhand der Identität die Integrität der Images garantieren zu können. Wird das nicht beachtet, führt die Relokation bzw. die Identitätsprüfung der Assembly zu weiteren Ausführungszeiten, die den Vorteil der nativen Images wieder zunichtemachen.

Die automatische Ressourcenverwaltung und die verbesserte Sicherheit haben dennoch ihren Preis – die Ausführung von *managed code* hat einen erhöhten Ressourcenbedarf und benötigt mehr Zeit. Außerdem sind die Antwortzeiten auf Programm-Ereignisse wesentlich schwieriger zu kalkulieren und zum Teil deutlich größer, was die Anwendbarkeit für Echtzeitaufgaben stark einschränkt.

Ein Grund dafür ist die automatische Speicherbereinigung, die automatische Freigabe nicht mehr benötigten Speichers und anderer Ressourcen. Im Regelfall entscheidet der Garbage Collector, wann der Speicher aufgeräumt werden soll. Der Entwickler kann aber den Zeitpunkt der Speicherbereinigung auch selbst festlegen. Während das einerseits, durch die Zusammenfassung der Freigabeoperationen, die Ausführungsdauer von Programmläufen verringern kann, können andererseits die Antwortzeiten auf Ereignisse dadurch in Mitleidenschaft gezogen werden. Das ist natürlich besonders für kleinere Maschinen nachteilig und stellt, vor allem im Hinblick auf die Marktausrichtung zu mobilen Kleingeräten, ein Problem dar.

.NET wird inzwischen auch bei performancekritischen Programmen, zum Beispiel Computerspielen (zum Beispiel mit dem XNA Framework), Animationsprogrammen, Konstruktionsprogrammen und ähnlichen, hochaufwendigen Programmen genutzt, da viele Programmierer der Meinung sind, dass aktuelle Systeme durch ihre höhere Geschwindigkeit den durch .NET bedingten Leistungsverlust ausgleichen.

Auf der anderen Seite steht die Meinung, dass die Qualität und Effizienz der traditionellen Softwareentwicklung zu wünschen übrig lassen und dass die diesbezüglichen Vorteile durch obige Verfahren deren Nachteile in der Regel aufwiegen. Im Allgemeinen wird dabei von einer asymmetrischen Verteilung ausgegangen, dass zum Beispiel 90 Prozent einer durchschnittlichen Anwendung problemlos „managed“, das heißt, auch mit automatischer Speicherbereinigung ausgeführt werden können, und lediglich 10 Prozent (zum Beispiel einzelne Funktionen oder Klassen) optimiert werden müssen.

Dabei hat sich – analog zur Diskussion „Assembler ja oder nein“ – gezeigt, dass oftmals durch die Optimierung der zugrundeliegenden Algorithmen und Datenstrukturen eine wesentlich höhere Ausführungsgeschwindigkeit erzielt werden kann als durch die Entscheidung für oder gegen eine Programmiersprache oder Programmiertechnik. Gerade für derlei ständige Optimierungen bewährt sich eine zuverlässige automatische Speicherverwaltung oft über Jahre, bevor man hier die Grenzen erreicht hat.

Nicht zuletzt können Programme auch in Hinblick auf die Ausführungsgeschwindigkeit im Zusammenspiel

mit automatischer Speicherbereinigung optimiert werden.

Klassenbibliothek

Die *Framework Class Library* (FCL) umfasst in der aktuellen Version 3.5 etwa 11.400 Klassen und andere Datentypen, die in mehr als 300 sogenannte Namensräume (Namespaces) unterteilt sind.[5] Im Vergleich zur ersten Version 1.0 mit 3.581 Datentypen in 124 Namensräumen ist das ein deutlicher Anstieg. Die Klassen erfüllen Aufgaben wie das Formatieren von Text, das Verschicken von E-Mails, aber auch das Generieren von Code. Die Unterteilung in Namensräume dient dazu, die große Menge an Informationen übersichtlicher zu gestalten. Beispielsweise befinden sich Klassen zum Generieren von Code in dem Namensraum `System.Reflection.Emit`.

Die Dokumentation der Klassen liefert der Hersteller in seinem *Software Development Kit* (SDK) mit (siehe unten).

Siehe auch: ADO.NET und ASP.NET

Verfügbarkeit und Werkzeuge

Der Hersteller Microsoft bietet .NET in verschiedenen Formen an. Als reine Laufzeitumgebung samt benötigter Klassenbibliotheken (Framework), als kostenloses SDK für Entwickler, als kostenpflichtige integrierte Entwicklungsumgebung (IDE) in Form des *Microsoft Visual Studio .NET*. Speziell für Einsteiger und Studenten gibt es die kostenlosen *Microsoft Visual Studio Express Editions* mit Einschränkungen gegenüber den kostenpflichtigen Standard- oder Professional-Varianten. Eine ebenfalls kostenfreie IDE für .NET (und Mono) unter Windows findet man im Open-Source-Projekt SharpDevelop. Studenten bietet Microsoft weiterhin die Möglichkeit, über das *DreamSpark*-Programm kostenfrei die Professional-Variante des Visual Studios zu beziehen.

Seit *Windows Server 2003* bietet Microsoft darüber hinaus Server-Betriebssysteme an, die bereits eine .NET-Laufzeitumgebung integriert haben. Bei Vorversionen muss diese manuell installiert werden, sofern die betreffende Windows-Variante unterstützt wird. .NET ist erst ab Windows 98 einsetzbar, die Programmierung von Webanwendungen (ASP.NET) etwa läuft nur ab Windows 2000. In Windows Vista ist .NET ein Kernbestandteil des Systems. Auf Nicht-Windows-Systemen wird .NET von Microsoft offiziell nicht unterstützt – so verbleibt die Plattformunabhängigkeit in der Liste der Möglichkeiten von .NET. Allerdings existieren die bereits erwähnten Open-Source-Projekte, die .NET auch für andere Plattformen (zum Beispiel Linux) verfügbar machen, wenn sie auch nicht den vollen Funktionsumfang des .NET-Frameworks unter Windows bieten können.

Für Handhelds und Mobiltelefone, die unter Windows CE bzw. Windows Mobile laufen, existiert eine funktional reduzierte Version der .NET-Laufzeitumgebung in Form des .NET Compact Frameworks. Es lässt sich aber nur unter Verwendung des kostenpflichtigen *Visual Studio .NET 2003* oder neuer für diese Plattform entwickeln.

Für neuere Mobiltelefone die auf Windows Phone basieren stellt Microsoft die .NET Komponenten Silverlight und XNA zur Entwicklung von Applikationen kostenlos eine Express Version von Visual Studio als **Windows Phone Developer Tools**[6] zur Verfügung.

Im September 2006 stellte Microsoft zusätzlich das .NET Micro Framework vor. Es stellt eine nochmals eingeschränkte Version des .NET Frameworks speziell für Embedded-Geräte dar. Je nach Plattform soll das Framework zwischen 512 KByte und 1 MByte auf dem Gerät beanspruchen und lässt sich direkt aus dem Flash-Speicher oder dem ROM starten. In diesem Falle arbeitet das Micro Framework als Betriebssystem, es kann aber auch auf ein vorhandenes Windows-Betriebssystem aufsetzen.

Technische Einzelheiten

Programmausführung

Der Compiler für .NET-Sprachen erzeugt keinen Maschinencode, der direkt vom Betriebssystem ausgeführt werden kann. Stattdessen wird ein prozessorunspezifischer Zwischencode, der sogenannte Intermediate Language Code (IL-Code) erzeugt. Dieser besteht aus Befehlen, die auf der stackbasierten virtuellen Maschine (VM) ausgeführt werden. Die resultierenden Programme haben, wie andere Programme unter Windows, eine „.exe“-Erweiterung. Das wird durch eine kleine Routine am Anfang des Programms ermöglicht, die die virtuelle Maschine startet, welche wiederum den Zwischencode ausführt.

Wenn das Programm ausgeführt wird, übersetzt ein JIT-Compiler, der in der Laufzeitumgebung *Common Language Runtime* (CLR) enthalten ist, den Zwischencode in Maschinencode, der dann vom Prozessor direkt ausgeführt werden kann.

Da Code aus allen .NET-Sprachen in dieselbe Zwischensprache übersetzt wird, können Funktionen und Klassenbibliotheken, die in verschiedenen .NET-Sprachen geschrieben sind, problemlos gemeinsam in einem Programm verwendet werden.

Assemblies

Übersetzte Programmklassen werden als ausführbare Programme in sogenannten Assemblies zusammengefasst und bereitgestellt (vergleichbar mit Jar-Dateien in Java). Diese haben typischerweise die altbekannten Endungen .exe oder .dll, sind intern jedoch völlig anders strukturiert. Insbesondere sind im sogenannten *Manifest* alle notwendigen Metadaten aufgeführt, so dass für reine .NET-Programme in der Regel die gewohnte, aber aufwändige und fehlerträchtige Registrierung entfällt (Ausnahme zum Beispiel *COM+/Enterprise Services*).

Assemblies können entweder *privat*, *gemeinsam* (*shared*) oder *global* sein. Private Assemblies befinden sich in demselben Verzeichnis wie das auszuführende Programm. Daher wird angenommen, dass die Version des Assemblies kompatibel zum Programm ist und daher nicht von der CLR geprüft wird.

Ein gemeinsames (*shared*) Assembly kann sich in einem Verzeichnis befinden, auf das von mehreren Programmen zugegriffen wird. Daher wird für ein gemeinsames Assembly ein sogenannter *Strong Name* benötigt, bestehend aus dem Dateinamen des Assemblies, seiner Version, der Kultur und einem kryptografischen Schlüssel. Durch eine Konfigurationsdatei, die sich in dem Verzeichnis des Programms befindet, kann der Anwendung der Speicherort der gemeinsamen Assemblies mitgeteilt werden. Ein *Strong Name* kann mit Hilfe des Werkzeugs *sn* erzeugt werden.

Ein globales Assembly wird im globalen Assembly-Zwischenspeicher (Global Assembly Cache, GAC) gespeichert. Mit Hilfe des Werkzeugs *gacutil* können Assemblies dem GAC hinzugefügt werden. Innerhalb des GAC können Assemblies mit unterschiedlichen Versionen, Kulturen gespeichert werden. Mit Hilfe von Konfigurationsdateien kann festgelegt werden, welche Versionen eines Assemblies von der Anwendung benutzt werden sollen. Erfolgt keine Angabe, so wird nur die Version benutzt, die bei der Erstellung der Anwendung benutzt wurde. Wenn diese nicht vorhanden ist, wird beim Start des Programms eine Fehlermeldung ausgegeben.

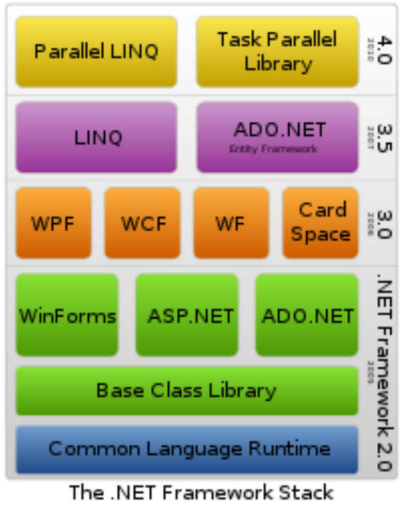
Aktuelle Windows-Versionen besitzen eine Explorer-Erweiterung, die eine aussagekräftige Anzeige des Inhalts des GAC im *Windows Explorer* ermöglicht.

Versionen

Microsoft begann mit der Entwicklung des .NET Frameworks in den späten 1990ern, ursprünglich unter

dem Namen der *Next Generation Windows Services* (NGWS). Gegen Ende des Jahres 2000 wurden die ersten Betaversionen von .NET 1.0 veröffentlicht.

Version	Versionsnummer	Datum
1.0	1.0.3705.0	5. Januar 2002
1.1	1.1.4322.573	1. April 2003
2.0	2.0.50727.42	7. November 2005
3.0	3.0.4506.30	6. November 2006
3.5	3.5.21022.8	9. November 2007
3.5 SP 1	3.5.30729.1	11. August 2008
4.0	4.0.30319	12. April 2010
4.5		-



Die .NET-Framework-Hierarchie

.NET Framework 1.0

Version 1.0 stellt die erste Veröffentlichung des .NET Frameworks dar. Es wurde am 13. Februar 2002 für Windows 98, NT 4.0, 2000 und XP veröffentlicht. Der Support von Microsoft für diese Version endete am 10. Juli 2007, der erweiterte Support lief noch bis zum 14. Juli 2009.

.NET Framework 1.1

Der erste Upgrade von .NET wurde als Installer am 3. April 2003 veröffentlicht. Es wurde gleichzeitig als integraler Bestandteil der Entwicklungsumgebung Visual Studio .NET 2003 vertrieben. Version 1.1 war die erste Version von .NET, die zusammen mit einem Betriebssystem, nämlich dem Windows Server 2003, ausgeliefert wurde. Dieser hieß bis zum Release Candidate sogar Windows Server .NET[7]. Der offizielle Support für diese Version endete am 14. Oktober 2008, der erweiterte Support endet am 8. Oktober 2013. Da .NET 1.1 eine Komponente des Windows Server 2003 darstellt, läuft der erweiterte Support zusammen mit dem Support für dieses Betriebssystem aus – derzeit wird von Microsoft der 30. Juni 2013 als Termin angegeben.

Änderungen in 1.1 im Vergleich mit 1.0

- Eingebaute Unterstützung für mobile ASP.NET-Schaltflächen. Zuvor verfügbar als Add-on für das .NET Framework, nun ein Bestandteil des Frameworks.
- Änderungen in der Sicherheitsarchitektur – Windows-Forms-Assemblies aus dem Internet werden in einer Sandbox ausgeführt, zusätzlich wurde für ASP.NET-Anwendungen die Code Access Security aktiviert.
- Eingebaute Unterstützung für ODBC- und Oracle-Datenbanken. Zuvor verfügbar als Add-on für das .NET 1.0, nun ein Bestandteil des Frameworks.
- .NET Compact Framework – eine Version von .NET für mobile Geräte.
- Einführung der Internet Protocol Version 6 (IPv6).
- Diverse Änderungen in der API.

.NET Framework 2.0

.NET 2.0 wurde zusammen mit Visual Studio 2005, Microsoft SQL Server 2005 und Microsoft BizTalk 2006 veröffentlicht.

- Das .NET-2.0-Paket wurde am 22. Januar 2006 zum Download zur Verfügung gestellt.
- Version 2.0 ohne Servicepack ist die letzte Version, die Windows 2000, Windows 98 und Windows Me unterstützt.
- Die Version wird auch mit dem Microsoft Windows Server 2003 R2 ausgeliefert (nicht standardmäßig installiert).

Änderungen in 2.0 im Vergleich mit 1.1

- Zahlreiche Änderungen in der API.
- Eine neue API für native Anwendungen, die eine Instanz der .NET-Laufzeit beherbergen wollen. Die neue API gewährleistet eine feinstrukturierte Kontrolle über das Verhalten der Laufzeit in Bezug auf Multithreading, Speicherallokation, dem Laden von Assemblies und mehr. Es wurde ursprünglich entwickelt um effizient die Laufzeit im Microsoft SQL Server zu betreiben, welcher seinen eigenen Scheduler und eine eigene Speicherverwaltung implementiert.
- Vollständige 64-Bit-Unterstützung für die x64- und alle IA64-Plattformen.
- Direkt in die .NET CLR eingebaute Sprachunterstützung für generische Typen.
- Viele Zusätze und Verbesserungen für ASP.NET-Web-Schaltflächen.
- Neue Datensteuerung mit deklarativer Datenbindung (engl. „data binding“).
- Neue personalisierende Features für ASP.NET, zum Beispiel Unterstützung von Themes, Skins und WebParts.
- .NET Micro Framework – eine Version des .NET Frameworks in Bezug auf die *Smart Personal Objects Technology*-Initiative.

.NET Framework 3.0

.NET Framework 3.0, ehemals *WinFX* genannt, erweitert die *Managed*-API, die einen integralen Bestandteil der Betriebssysteme Windows Vista und Windows Server 2008 darstellt. Seit dem 6. November 2006 ist das .NET Framework 3.0 für Windows XP ab *Service Pack 2* und für Windows Server 2003 verfügbar, um Entwicklern rechtzeitig die Entwicklung und Portierung von Programmen nach Vista zu ermöglichen. In der dritten Hauptversion von .NET wurden tiefgreifende Änderungen an der Architektur vorgenommen. Dazugekommen sind Funktionalitäten, die vor allem unter *Windows Vista* zum Einsatz kommen sollen. Das .NET Framework 3.0 greift auf die CLR aus .NET 2.0 zurück. Entgegen früheren Veröffentlichungen wurde gleichzeitig keine Version des .NET Compact Framework mit .NET 3.0 veröffentlicht.

Das .NET Framework 3.0 setzt sich aus vier Hauptkomponenten zusammen:

- *Windows Presentation Foundation* (entwickelt unter dem Codenamen *Avalon*): Eine neue Technik, Objekte mit Hilfe der eigens dafür entwickelten Beschreibungssprache XAML auf dem Bildschirm darzustellen. Dabei werden, wie bei Quartz Extreme unter Mac OS X, beispielsweise Transparenzeffekte nicht mit der CPU errechnet, sondern leistungssteigernd über die 3D-Grafikkarte. Das entlastet die CPU und lässt das System auch optisch „flüssiger“ aussehen.
- *Windows Communication Foundation* (entwickelt unter dem Codenamen *Indigo*): Eine neue dienstorientierte Kommunikationsplattform für verteilte Anwendungen. Hier will Microsoft viele Netzwerk-Funktionen zusammenführen und den Programmierern solcher Anwendungen standardisiert zur Verfügung stellen. Bei dieser Weiterentwicklung von DCOM legt Microsoft besonderen Wert auf internetbasierte Anwendungen.
- *Windows Workflow Foundation*: Infrastruktur für die einfachere Entwicklung von Workflow-Anwendungen, sowohl in geschäftlicher als auch technischer Hinsicht, aber auch für dokument- und webbasierte Workflows. Bietet zudem grafische Designer für Visual Studio (Modellierung mittels Fluss- und Zustandsdiagrammen). Funktionen davon sollen unter anderem in zukünftigen Versionen von Office (SharePoint) und BizTalk verwendet werden.
- *Windows CardSpace* (entwickelt unter dem Codenamen *InfoCard*): Identitätsmanagement-

Infrastruktur für verteilte Anwendungen. Mit *Windows CardSpace* will Microsoft einen neuen Standard für das Identitätsmanagement unter anderem im Internet etablieren. In dem eigenen Browser Internet Explorer (Version 7) schon integriert, will Microsoft für diesen Dienst auch Plug-ins für alternative Browser entwickeln, mindestens aber für Mozilla Firefox.[8]

Silverlight (vormals *WPF/E*) enthält eine stark verkleinerte Untermenge des .NET Frameworks und soll im Wesentlichen Webbrowser befähigen, reichhaltige Internetanwendungen auf Basis der WPF auszuführen. Normale Programme auf Basis der WPF sind ebenfalls „webfähig“, benötigen aber das vollständige .NET 3.0, welches derzeit nur für Windows verfügbar ist. *Silverlight* jedoch soll auch für Mac OS, ältere PCs mit Windows sowie Linux bereitgestellt werden.

Für die Vorab-Demonstration des neuen .NET Framework präsentierte Microsoft den Foto-Dienst *Microsoft Max*. Mit Herausgabe der endgültigen Version wurde der Dienst eingestellt.

.NET Framework 3.5

Version 3.5 des .NET Framework wurde am 19. November 2007 veröffentlicht. Es verwendet die CLR aus Version 2.0. Mit Version 3.5 werden gleichzeitig das *.NET Framework 2.0 SP1* und *.NET Framework 3.0 SP1* installiert. Zeitgleich mit der Version 3.5 wurde das *.NET Compact Framework 3.5* veröffentlicht.

Die Version 3.5 SP1 (11. August 2008) ergänzte die Bibliothek um das ADO.NET Entity Framework 1.0 und die ADO.NET Data Services. Mit der Version 3.5 SP1 werden gleichzeitig das *.NET Framework 2.0 SP2* und *.NET Framework 3.0 SP2* installiert. Am 18. Dezember 2008 wurde zudem ein General Distribution Release veröffentlicht, das lediglich Fehlerbehebungen beinhaltet.[9]

Der Quellcode der Klassenbibliothek (BCL) wurde teilweise unter der *Microsoft Reference Source License* freigegeben.

Änderungen seit Version 3.0

- Neue Sprachfunktionen für C# 3.0 und VB.NET 9.0
- Unterstützung von Expression Trees und Lambda-Methoden
- Extension Methoden
- Anonyme Typen
- LINQ
- Unterstützung von Paging für ADO.NET
- API für asynchrone Netzwerk-I/O
- P2P-Netzwerkstack mit verwaltetem *Peer Name Resolution Protocol Resolver*
- Verbesserte WCF- und WF-Bibliotheken
- Integration von ASP.NET AJAX
- Neuer Namensraum `System.CodeDom`
- Microsoft ADO.NET Entity Framework 1.0

.NET Framework 4

Microsoft gab Informationen zum .NET Framework 4 erstmals am 29. September 2008 und auf der *Professional Developers Conference (PDC 2008)* bekannt. Obwohl noch keine genauen Details über die zukünftige Version bekannt wurden, wurden einige allgemeine Informationen über die Pläne des Unternehmens veröffentlicht. So sollen Parallelrechner stärker unterstützt werden, insbesondere Systeme mit Mehrkernprozessoren oder Systeme, die auf verteiltes Rechnen ausgelegt sind. Weiterhin existieren Pläne, Verfahren wie PLINQ (Parallel LINQ) zu integrieren, sowie die *Task Parallel Library* (TPL), die die Parallelverarbeitung über Methodenaufrufe und Delegaten ermöglicht. Darüber hinaus hat Microsoft bekanntgegeben, eine Untermenge des .NET-Frameworks und von ASP.NET im Microsoft Windows Server

Core zu unterstützen.

Die erste Beta-Version des .NET 4 wurde am 18. Mai 2009 veröffentlicht. Am 19. Oktober 2009 folgte eine zweite Beta-Version. Ursprünglich war die Veröffentlichung des .NET-Frameworks zusammen mit der Entwicklungsumgebung Microsoft Visual Studio 2010 für den 22. März 2010 geplant. Um jedoch mehr Zeit für weitere, von Nutzern der Beta 2 des Microsoft Visual Studio 2010 geforderte, Optimierungen zu erhalten, kündigte Microsoft im Dezember 2009 eine Verschiebung des Releases von .NET 4 und Visual Studio 2010 um einige Wochen an.[10] Am 10. Februar 2010 erschien das „Release Candidate“. Die endgültige Version des .NET 4 und des Visual Studios 2010 in der englischen Sprachfassung wurde von Microsoft schließlich am 12. April 2010 veröffentlicht.[11]

Zu den wichtigsten Neuerungen[12] bei .NET Framework 4 gehören unter anderem:

- Dynamic Language Runtime
- Codeverträge
- Unterstützung für Kovarianz und Kontravarianz durch generische Schnittstellen und Delegaten
- Managed Extensibility Framework
- Unterstützung für Speicherabbilddateien
- Automatische Speicherbereinigung im Hintergrund
- Neues Programmiermodell zum Schreiben von Multithread- und asynchronem Code[13]
- Verbesserte Leistung, Skalierbarkeit und Workflow-Modellierung sowie neuer Designer bei der Windows Workflow Foundation[14]

.NET Framework 4.5

Vom in Entwicklung befindlichen .NET Framework 4.5 hat Microsoft die ersten Informationen auf der *BUILD Windows Konferenz* am 14. September 2011 bekannt gegeben. So soll in dieser Version unter anderem auf ein "Native Image" aller Assemblies verzichtet werden, um Platz bei der Installation zu sparen. Stattdessen soll die CLR die Benutzung von Managed Code protokollieren und dann bei Bedarf ein Native Image erzeugen. Des Weiteren soll der Garbage Collector für Multicore-Systeme mit mehr als vier Kernen verbessert werden.

Es gibt aber auch Neuerungen in den Sprachen C# und Visual Basic.NET. Zum Beispiel soll die asynchrone Programmierung durch neue Schlüsselwörter wie *async* vereinfacht werden.[15]

Aktuell (Stand April 2012) gibt es eine *Beta*.

Entstehungsgeschichte

Die Entwicklung der .NET-Plattform wurde als notwendig angesehen, um die in die Jahre gekommenen Konzepte von Windows durch neue zu ersetzen, war jedoch auch das Ergebnis des Rechtsstreits von Microsoft mit Sun über Java. Microsoft adaptierte das von Sun entwickelte Java-System und erweiterte es nach eigenen Bedürfnissen, was die Plattformunabhängigkeit von Java-Applikationen beeinträchtigte. Als Sun das unter anderem durch Gerichtsverfügung unterband, änderte Microsoft seine Strategie. Zudem war es Microsoft bis zur Entwicklung von .NET nicht gelungen, im lukrativen Markt für mobile Kleingeräte Fuß zu fassen.

Zudem hatten sich mit der Zeit verschiedene, zueinander inkompatible Softwaresysteme für Windows entwickelt. Die drei für Windows meistverwendeten Programmiersprachen C++, Visual Basic sowie die Microsoft-Implementierung einer Java-Syntax, J++, waren zueinander nicht kompatibel und die Zusammenarbeit über verschiedene Brücken erwies sich als sehr kompliziert.

Zeichenketten und ANSI/Unicode

Die Datentypen für Zeichenketten (engl. „strings“) waren nicht binärkompatibel zueinander. Wollte man solche über zwei Softwaresysteme hinweg schreiben, so musste man Laufzeiteinbußen wegen Konvertierungsfunktionen hinnehmen. Verschärfend kam die Koexistenz von ANSI und Unicode hinzu. Viele Programme unterstützten kein Unicode oder wurden dafür noch nicht ausgerüstet. .NET verwendet einheitlich Unicode für Zeichenketten.

Speicherverwaltung

Jede Entwicklungsplattform besaß ein eigenes System für die Verwaltung des Speichers. J++ und Visual Basic besaßen eine automatische Speicherverwaltung, das heißt der Programmierer überließ (weitgehend) dem System die Verwaltung des Speichers. Visual C++ hingegen besaß keine Speicherverwaltung, der Programmierer musste sich selber darum kümmern.

Offenlegung des Quellcodes

Am 17. Januar 2008 veröffentlichte Microsoft den Quelltext des Frameworks unter der restriktiven Microsoft Reference License. Zu diesem Schritt entschloss sich Microsoft bereits im Oktober 2007, als Sun Microsystems ihr Produkt *Java* unter der GNU GPL mit eigenen Zusatzklauseln zur Verfügung stellte.

Im Gegensatz zu *Java* stehen die Quelltexte des .NET-Frameworks jedoch nicht unter einer freien Lizenz – es ist nicht erlaubt, den Quellcode zu modifizieren oder in andere Projekte zu integrieren. Microsoft gestattet lediglich die Einsicht des Quellcodes zur Unterstützung des Debugging-Prozesses in Projekten, die für die Windows Plattform entwickelt werden.[16] Dank der Veröffentlichung des Quelltextes lassen sich in derartigen Projekten im Fehlerfall zumindest präzisere Workarounds schreiben.

Verhältnis zu Mono

Teile der Open-Source-Community[17] sehen in der Offenlegung unter der restriktiven Lizenz eine Gefahr für das Projekt Mono, welches .NET-Anwendungen unter Linux teilweise verfügbar macht. Microsoft hatte 2007 noch behauptet, das Projekt würde Quellcode aus dem .NET-Framework enthalten. Da das Framework und *Mono* gleichermaßen *.NET* implementieren, befürchtet man nun zwangsweise starke Ähnlichkeiten im Quellcode.

Durch das umstrittene Patentabkommen zwischen Microsoft und Novell[18] (dem ehemaligen Projektträger von *Mono*) werden Befürchtungen nichtig, Microsoft könnte das Projekt wegen Lizenzverstößen verklagen. Das Patentabkommen schützt derzeitig sowohl die Community unter Novell als auch Microsoft vor gegenseitigen Patentansprüchen.

Chronik der .NET-Entwicklung

Jahr	Ereignisse
2000	■ Juni – Bill Gates stellt erstmals die .NET-„Vision“ vor. ■ Juli – Auf der Entwicklerkonferenz PDC gibt es erstmals CDs mit lauffähigen Vorabversionen des *Frameworks* und von *Visual Studio .NET* ■ Oktober – C# und die CLI (siehe unten) werden (von MS, HP und Intel) bei der Europäischen Standardisierungsorganisation *European Computer Manufacturers Association (ECMA)* eingereicht, was für Microsoft einen ungewöhnlichen Schritt der

	Öffnung darstellt.
2002	■ Januar – .NET (V1.0) wird offiziell mit der zugehörigen Entwicklungsumgebung SDK (und Visual Studio .NET 2002) vorgestellt. ■ Verwirrung: Im Zuge des Marketings wird nach Microsofts Gewohnheit versucht, alle anstehenden Neuentwicklungen unter einen, den .NET-Begriff, zu fassen, wodurch selbst Fachleute einschließlich Microsoft-Mitarbeiter nicht mehr verstehen, worum es eigentlich geht. Die Programmiertechnik und Entwicklungsumgebung wird zunächst in Verbindung gestellt zu konkreten Webdiensten, die Microsoft anbietet (Codename „Hailstorm“ wird zu „.Net My Services“) und später vom Marketing wieder von .NET entkoppelt. Auch die nächste Betriebssystem-Generation von Windows wird als Bestandteil von .NET angekündigt.
2003	■ Vorstellung von .NET 1.1 und Visual Studio 2003 mit im Wesentlichen geringfügigen Verbesserungen und Erweiterungen.
2005	■ Betaversionen von *.NET 2.0* und *Visual Studio 2005* erhältlich. ■ 7. November: Visual Studio 2005 und .NET Framework 2.0. werden in den USA veröffentlicht. ■ 19. Dezember: Deutsche Version des .NET Frameworks 2.0 verfügbar.
2006	■ 6. Februar: Visual Studio 2005 wird in deutscher Sprache veröffentlicht. ■ 6. November: .NET Framework 3.0 verfügbar.
2007	■ Erste Berichte um .NET 3.5 und dem neuen Visual Studio 2008 mit dem Codenamen „Orcas“. ■ 19. November: .NET 3.5 und Visual Studio 2008 wird in den USA veröffentlicht.
2008	■ 17. Januar: Das .NET Framework wird zwecks „erleichtertem Debugging“ im Quelltext veröffentlicht. ■ Februar: Visual Studio 2008, Microsoft SQL Server 2008 und Windows Server 2008 werden veröffentlicht.
2009	■ 18. Mai: .NET 4.0 Beta 1 veröffentlicht. ■ 19. Oktober: .NET 4.0 Beta 2 veröffentlicht.
2010	■ 10. Februar: .NET 4.0 RC veröffentlicht. ■ 12. April: .NET 4.0 sowie Visual Studio 2010 in finaler Version veröffentlicht.

Siehe auch

- Liste von .NET-Sprachen

Literatur

- Holger Schwichtenberg: *Microsoft .NET 2.0 Crashkurs.* Microsoft Press, Unterschleißheim 2006. ISBN 3-86063-987-0.
- Holger Schwichtenberg: *Microsoft .NET 3.0 Crashkurs.* Microsoft Press, Unterschleißheim 2007. ISBN 3-86645-501-1.
- Jürgen Kotz, Rouven Haban, Simon Steckermeier: .NET 3.0. WPF, WCF und WF – ein Überblick. *Addison-Wesley, München Februar 2007. ISBN 3-8273-2493-9.*
- Daniel Liebhart, Guido Schmutz, Marcel Lattmann, Markus Heinisch, Michael Königs, Mischa Kölliker, Perry Pakull, Peter Welkenbach: *Architecture Blueprints* Hanser Verlag, 2007. ISBN 978-3-446-41201-9.

Weblinks

Allgemein

- .NET-Produktwebseite und Plattform Explorer (http://www.microsoft.de/net)
- Microsoft MSDN Developer Center – .NET Framework (http://msdn.microsoft.com/de-de/netframework/default.aspx)
- Alternative Implementierungen: Mono (http://www.mono-project.com/) , DotGNU (http://www.dotgnu.org/)
- Deutschsprachiges Magazin rund um das .NET Framework (http://it-republik.de/dotnet/dotnet-magazin-ausgaben/)

.NET 3.0

- Offizielle Seite .NET Framework (http://www.microsoft.com/germany/msdn/netframework/default.aspx)
- Download des Microsoft .NET Framework 3.0 Redistributable Package (http://www.microsoft.com/downloads/details.aspx?displaylang=de&FamilyID=10CC340B-F857-4A14-83F5-25634C3BF043)
- Download des Microsoft .NET Framework 3.5 Service pack 1 (Full Package) (http://www.microsoft.com/downloads/details.aspx?familyid=D0E5DEA7-AC26-4AD7-B68C-FE5076BBA986&displaylang=de)

.NET 4.0

- Download Microsoft .NET Framework 4.0 (eigenständiger Installer) (http://www.microsoft.com/downloads/details.aspx?FamilyID=0a391abd-25c1-4fc0-919f-b21f31ab88b7&displaylang=de)
- Download Microsoft .NET Framework 4.0 Client Profile (eigenständiger Installer) (http://www.microsoft.com/downloads/details.aspx?FamilyID=e5ad0459-cbcc-4b4f-97b6-fb17111cf544&displaylang=de)
- Weitere Downloads zu .NET Framework 4 (http://msdn.microsoft.com/de-de/netframework/aa569263.aspx)

Einzelnachweise

1. Microsoft .NET Framework 4 (eigenständiger Installer) (http://www.microsoft.com/downloads/de-de/details.aspx?FamilyID=0a391abd-25c1-4fc0-919f-b21f31ab88b7&pf=true) – Seite bei *Microsoft*; Stand: 22. April 2011
2. Microsoft Visual Studio 11 Developer Preview (Web Installer) (http://www.microsoft.com/download/en/details.aspx?id=27543) (englisch) – Seite bei *Microsoft*; Stand: 10. Oktober 2011
3. Jenseits von JavaScript und HTML5: Ausblick auf .NET 4.5 und Visual Studio 11.0

(http://www.heise.de/developer/artikel/Jenseits-von-JavaScript-und-HTML5-Ausblick-auf-NET-4-5-und-Visual-Studio-11-0-1355903.html) – Artikel bei *Heise online*, vom 6. Oktober 2011
4. ScottGu's Blog (Corporate Vice President in the Microsoft Developer Division) (http://weblogs.asp.net/scottgu/archive/2008/01/16/net-framework-library-source-code-now-available.aspx) (engl.)
5. Brad Abrams: Number of Types in the .NET Framework (http://blogs.msdn.com/brada/archive/2008/03/17/number-of-types-in-the-net-framework.aspx)
6. microsoft.com: *Microsoft Visual Studio 2010 Express for Windows Phone (http://www.microsoft.com/express/Phone/)* , Zugriff am 15. April 2011
7. http://www.windows-tweaks.info/html/windows-net-server.html
8. itmagazine.ch (http://www.itmagazine.ch/artikel/art_details.cfm?aid=31174) : Microsoft bringt Identity Management für Firefox
9. GDR (General Distribution Release) (http://support.microsoft.com/kb/959209)
10. Verlängerte Betaphase für Visual Studio 2010 – „Release Candidate" im Februar 2010 (http://blogs.msdn.com/vsnewsde/archive/2009/12/18/verl-ngerte-betaphase-f-r-visual-studio-2010-release-candidate-im-februar-2010.aspx)
11. Visual Studio 2010 und .NET 4 veröffentlicht (http://www.heise.de/newsticker/meldung/Visual-Studio-2010-und-NET-4-veroeffentlicht-975002.html) – Artikel bei *Heise online*, vom 12. April 2010
12. Neues in .NET Framework 4 (http://msdn.microsoft.com/de-de/library/ms171868(v=VS.100).aspx)
13. Parallele Programmierung in .NET Framework (http://msdn.microsoft.com/de-de/library/dd460693(v=VS.100).aspx)
14. Neues in Windows Workflow Foundation (http://msdn.microsoft.com/de-de/library/dd489410(v=VS.100).aspx)
15. *.NET Framework 4.5 (.NET FX 4.5) (http://www.it-visions.de/lex/6250.aspx)* . IT-Visions.de (September 2011). Abgerufen am 25. September 2011.
16. *Microsoft Reference Source Licensing*. (http://referencesource.microsoft.com/referencesourcelicensing.aspx) Microsoft, abgerufen am 20. November 2010 (englisch): „The license limits the source code release to use on the Windows platform only."
17. Microsofts Open Source Trap (http://www.eweek.com/c/a/Linux-and-Open-Source/Microsofts-OpenSource-Trap-for-Mono/) (engl.)
18. Was die Kunden wollen: Microsoft und Novell kooperieren (http://www.heise.de/open/artikel/Microsoft-und-Novell-kooperieren-222001.html) – Artikel bei *Heise open*, vom 3. November 2006