

.NETBlogBook



Ausgabe 1
Jänner 2007

Copyright © 2005-2007 by DI (FH) Norbert Eder, Kai Gloth
Published 2007 by DI (FH) Norbert Eder, Graz, Austria



Autoren dieser Ausgabe



DI (FH) Norbert EDER
<http://blog.norberteder.com>

Norbert Eder ist als Software Architect und Technologieberater tätig. Zusätzlich engagiert er sich für die .NET Community und versucht sein Wissen an Hilfesuchende weiter zu vermitteln.

Sie können ihn unter me@norberteder.com erreichen.

Community Projekte

<http://www.dotnet-blog.com>
<http://www.dotnetcasts.com>



Kai Gloth
<http://blog.veloursnebel.de>

Kai Gloth arbeitet als Softwareentwickler und studiert Informatik. Er beschäftigt sich hauptsächlich mit ASP.NET und versucht die Begeisterung für diese Technologie an Einsteiger weiterzugeben.

Sie können ihn unter kai@veloursnebel.de erreichen.

1.	.NET BlogBook.....	6
1.1.	Vorwort	7
1.2.	Ziel des Projektes	7
1.3.	Was wird in diesem Buch nicht abgedeckt?.....	7
2.	.NET Framework.....	8
2.1.	Allgemein.....	8
2.1.1.	Klassen- und Namespace-Informationen zum .NET Framework erfolgreich finden	8
2.1.2.	.NET: Wo beginne ich als Anfänger?	9
2.1.3.	.NET Serialisierung und Versionierung	9
2.2.	Basis Framework.....	10
2.2.1.	ICloneable: Etwas unsauber, nicht?	10
2.2.2.	Paper zur Common Language Infrastructure (CLI)	11
2.2.3.	Objekt auf Eigenschaftsänderungen überprüfen	11
2.2.4.	Isolierter Speicher für isolierte Daten!	13
2.2.5.	Design und Implementierung der .NET Generics	14
2.2.6.	Funktionsangebot des System.Environment Namespaces	14
2.2.7.	Der Unterschied zwischen const und static readonly!.....	15
2.2.8.	Standard-Windows-Sounds abspielen.....	16
2.2.9.	Die wohl häufigste Ausnahme: NullReferenceException.....	16
2.2.10.	Vermeide: if (myBoolean == true).....	17
2.2.11.	Begriffserklärung: Boxing und Unboxing.....	18
2.2.12.	C#: Welche Datei repräsentiert einen Prozess	19
2.2.13.	Zeitmessung einfach gemacht	19
2.2.14.	Klassen als obsolet (veraltet) markieren	20
2.2.15.	Verwendung des Namespace Alias Qualifier (::).....	20
2.2.16.	Der Operator ??	21
2.2.17.	Lese- und Schreibrechte einer Datei überprüfen	21
2.2.18.	Mit Dateizugriffs-Rechten arbeiten.....	23
2.2.19.	Assemblies nur zu Reflection-Zwecken laden	24
2.2.20.	List<Person> oder doch lieber eine PersonCollection?	25
2.2.21.	Reflection Geschwindigkeitstest	25
2.2.22.	MD5-Wert eines Strings.....	27
2.2.23.	Graphics.MeasureString: Maßeinheit der Rückgabe beeinflussen	27
2.2.24.	Installierte Grafik-Codecs?.....	28
2.2.25.	JPEG Grafiken verlustlos rotieren.....	28
2.2.26.	Die grafische Länge eines Strings mit C# bestimmen	28
2.2.27.	Strong named assembly trotz Referenz auf eine ActiveX Komponente Teil 3	29
2.2.28.	Strong named assembly trotz Referenz auf eine ActiveX Komponente Teil 2	29
2.2.29.	Strong named assembly trotz Referenz auf eine ActiveX Komponente	30
2.2.30.	Object Mapping oder doch lieber DataBinding?.....	31
2.2.31.	WebRequest und SSL Zertifikate.....	31
2.2.32.	C# Beginner: Enumeratoren vs Flags.....	32
2.2.33.	System.IO.Compression - Hilfreich oder doch ein Scherz?.....	33
2.2.34.	C#-Beginner: Exception-Handling unter C#.....	34
2.2.35.	C#: Methode mit Parameter via ThreadStart aufrufen	35
2.2.36.	Connectionstrings unter C#, VB.NET.....	36
2.2.37.	Strings unter .NET.....	37
2.2.38.	Einführung Garbage Collector	37
2.2.39.	Wie die benötigten Rechte einer Assembly feststellen?	38

2.2.40.	Drucken unter C#	39
2.2.41.	AppDomains und Memory-Überwachung	40
2.2.42.	C# - Daten zwischen zwei Formularen austauschen	41
2.2.43.	C# und Properties	41
2.2.44.	AppDomains und ShadowCopies.....	42
2.2.45.	Connection-Probleme zu SQL Server 2005 Express?.....	44
2.2.46.	Wieso gibt es keine Assembly.Unload() Methode?	44
2.2.47.	Unable to debug: The binding handle is invalid.	44
2.2.48.	Dateien vergleichen	44
2.2.49.	Ressourcen schonen - Datenbanken besser öffnen und schließen	46
2.2.50.	Download einer Datei via HTTP.....	48
2.2.51.	Dateidownload ohne Filesystem	49
2.2.52.	Fehlender Namespace	50
2.2.53.	Daten aus dem Clipboard in einer Konsolenanwendung verwenden.....	51
2.2.54.	Url per WebRequest auslesen	51
2.2.55.	ApplicationPath in C# Konsolenanwendungen.....	52
2.2.56.	Read From Clipboard	52
2.2.57.	Parsing Dates and Times in .NET	52
2.2.58.	String formatting in C#	53
2.3.	Windows Forms	53
2.3.1.	ComboBox als DropDownList kann kein Text gesetzt werden	53
2.3.2.	Meine GUI friert während der Ausführung ein, was tun?.....	54
2.3.3.	Menüs dynamisch mit Hilfe einer XML Datei erstellen	55
2.3.4.	UserControls eines Namespaces finden	56
2.3.5.	.NET 2.0: ComboBox und AutoComplete	56
2.3.6.	C# Beginner: UserControl DoubleTrackBar Beispiel.....	56
2.3.7.	C# Beginner: Beispiel für den Aufbau eines Strategie-Spieles.....	57
2.4.	ASP.NET.....	58
2.4.1.	ViewState und TextBox - Control.....	58
2.4.2.	Bilder im GridView anzeigen	58
2.4.3.	Mehrere Spalten im DataTextField der DropDownList.....	59
2.4.4.	Länge eines Strings per Validation Control überprüfen.....	60
2.4.5.	Per Shift-Taste mehrere CheckBox Controls markieren	60
2.4.6.	„Wirklich löschen?“ im DetailsView	62
2.4.7.	Login Control ohne returnUrl	63
2.4.8.	Literal content is not allowed within a 'skin file'.....	64
2.4.9.	Login.aspx: Cannot convert type	65
2.4.10.	Von welcher Methode aufgerufen?	65
2.4.11.	User and Role Management in ASP.NET 2.0	66
2.4.12.	Custom Controls per Web.config registrieren.....	66
2.4.13.	Login-Cookie des Community Server verwenden	67
2.4.14.	„Wirklich löschen?“ im GridView	68
2.4.15.	Pop-Up per Response.Redirect().....	70
2.4.16.	Read-Only Datensätze im GridView, die Zweite.....	71
2.4.17.	Read-Only Datensätze im GridView.....	71
2.4.18.	Dynamischer Meta-Refresh in MasterPage.....	73
2.4.19.	Server.MapPath in Session_End ().....	73
2.4.20.	Dynamische Bilder per Generic Handler (*.ashx) anzeigen	74
2.4.21.	Using themed css files requires a header control on the page.....	75
2.4.22.	Formulare gegen SPAM schützen.....	76
2.4.23.	User Controls per Web.config registrieren.....	77

2.4.24.	Caching von Bildern verhindern	78
2.4.25.	XmlDataSource, GridView u. DataFormatString	78
2.4.26.	Controls dynamisch hinzufügen	80
2.4.27.	DropDownList - Änderung der Auswahl durch JavaScript-Dialog bestätigen	81
2.4.28.	Cache löschen.....	81
2.4.29.	Kopieren einer ASP.NET 1.1 Anwendung und VS 2003	82
2.4.30.	Controls anhand der ID rekursiv suchen.....	82
2.4.31.	AnkhSVN, TortoiseSVN und ASP.NET.....	83
2.4.32.	Reservierte ASP.NET Projektnamen	83
2.4.33.	SiteMap menu with icons	84
2.4.34.	Debug and Release Builds in ASP.NET 2.0	84
2.4.35.	Maximum request length exceeded.....	85
2.4.36.	Session lost, nachdem ein Verzeichnis umbenannt wurde.....	85
2.4.37.	ASP.NET: HttpWebRequest und Timeout-Problem.....	86
2.4.38.	Probleme mit WebUIValidation.js.....	86
2.5.	Services	86
2.5.1.	Windows Dienste mit C# und .NET 2.0 kontrollieren.....	86
2.6.	Windows Presentation Foundation.....	87
2.6.1.	Windows Presentation Foundation - Teil 2: XAML und Layouts	87
2.6.2.	WPF: Rotation und Skalierung einfach gemacht	87
2.6.3.	Windows Presentation Foundation - Teil 1: Einführung.....	89
2.7.	Sonstiges.....	89
2.7.1.	Im Batch den Ausführungserfolg einer aufgerufenen (selbst erstellten) Anwendung prüfen?	90
2.7.2.	Abhängigkeiten (Referenzen) führen zu Kompilations-Problemen.....	90
3.	Visual Studio	90
3.1.	Auflistung Visual Studio Shortcuts.....	91
3.2.	Visual Studio: Bookmarks aller geöffneten Dokumente ohne Rückfrage entfernen	91
3.3.	Interessante VS 2005 Tastenkombinationen	91
3.4.	Visual Studio 2005: Default Browser setzen	91
3.5.	Visual Studio: Anpassung Class Template	92
3.6.	Visual Studio 2005: Einfaches Einbinden von Namespaces.....	93
3.7.	Visual Studio 2005: Taskbar einblenden	94
3.8.	Visual Studio und Build Events	94
3.9.	Webprojekte mittels Firefox debuggen	94
3.10.	Codebehind file in Visual Studio 2005 hinzufügen	96
3.11.	Project Line Counter für Visual Studio.....	96
3.12.	Zur korrespondierenden Klammer springen	97
4.	Entwicklung	98
4.1.	Allgemein	98
4.1.1.	Continuous Integration - kenn ich nicht ..	98
4.1.2.	Überladen vs. überschreiben	99
4.2.	Analyse.....	100
4.2.1.	Eigenen Code analysieren	100
4.3.	Software Testing	100
4.3.1.	Das Übel Software-Testing	100
4.3.2.	Unit-Tests und Aufwand	101
4.3.3.	Grundlagen: Testgetriebene Entwicklung oder auch test-driven Development	102
4.3.4.	Grundlagen: White Box Tests	102
4.3.5.	Grundlagen: Black Box Tests	102

4.3.6.	[Tutorial] Unit-Tests mit Visual Studio .NET	103
4.3.7.	Nicht ausgeführte UnitTests mit TestDriven.NET.....	106
4.3.8.	Externes Configuration File benutzen.....	106
4.3.9.	ToolTips einzelner Items einer CheckBoxList setzen.....	107
4.4.	Deployment	107
4.4.1.	Deploying unter .NET 3	108
4.5.	Design Patterns.....	108
4.5.1.	Was sind Design Patterns?	108
4.5.2.	Design Patterns: Die Geschichte	108
4.5.3.	Patterns: Command Pattern	109
4.5.4.	Singleton Pattern	111
5.	Tools.....	112
5.1.	Tools: DotLucene - Fulltext Search Engine for .NET.....	112
5.2.	GUI für Windows Installer XML (WiX)	113
5.3.	Sandcastle Helferleins	113
5.4.	ReSharper UnitRun: kostenloser Testdriven.NET Gegenspieler	114
5.5.	CCTray: CruiseControl.NET Build-Fortschritt im Überblick	114
5.6.	CCNetConfig: CruiseControl.NET Konfigurationsdateien mittels Win-Forms-Anwendung erstellen.....	114
5.7.	WMI Code Creator	115
5.8.	Tipp: Guidande Explorer.....	116
5.9.	Documentation Generator: CodeDoc	116
5.10.	LINQ - Kennst du schon?.....	117
5.11.	ASP.NET Deployment Tool	117
5.12.	Lokalisierung des Community Servers	118
5.13.	Community Server Installation HowTo	119
5.14.	SQL Server Web Data Administrator	124
6.	Microsoft Office	124
6.1.	Word 2007: Custom Ribbons erstellen	124
6.2.	MS Outlook - Makros reloaded.....	125
6.3.	MS Outlook - Makros reloaded 2.....	126
6.4.	MS Outlook: Mails mit Shortcut als gelesen markieren	127
7.	Datenbank-Management-Systeme	128
7.1.	SQL Server 2000 Replikation und Error 18483	128
7.2.	SQL Server 2005: Output-Klausel	129
7.3.	Objekte und relationale Daten in einer Datenbank	130
7.4.	SQL Server 2005: Erstellung einer Replikation.....	131
7.5.	SQL Server 2005: Managed Code.....	131
7.6.	SQL Server 2005: Custom Datatypes	131
7.7.	C# und SQL Server 2005	131
7.8.	OODBMS- Object Oriented DataBase Management Systems	134
7.9.	SA User unter SQLServer 2005 umbenennen.....	135
7.10.	SQL Server 2000: Felder zur Replikation hinzufügen.....	135
7.11.	SQL Server 2000: Einschränkungen bei Replikation	136
7.12.	SQL Server Replizierung: Alt oder doch neu?.....	136
7.13.	Autoinkrement-Wert nach INSERT INTO auslesen, die Zweite.....	137
7.14.	Zufälligen Datensatz ausgeben.....	138
7.15.	Autoinkrement-Wert nach INSERT INTO auslesen.....	138
7.16.	Import einer CSV Datei mit Hilfe von BULK INSERT	139
8.	Sonstiges.....	140
8.1.	Verwendete Ports von Microsoft Produkten	141

8.2.	IIS Fehler: Fehler beim Zugriff auf die IIS-Metabasis. Was tun?	141
8.3.	C#: Google Web API schon getestet?	142
8.4.	Recent Projects in VS 2005	143
8.5.	IE 6.0 Contextmenü.....	143
9.	Abbildungsverzeichnis	143

1. .NET BLOGBOOK

1.1. Vorwort

Dies ist die erste Ausgabe des .NET BlogBooks. Es handelt sich hierbei um eine Sammlung und Überarbeitung von Blog-Beiträgen der in dieser Ausgabe genannten Autoren. Im Laufe der Zeit sammeln sich viele hilfreiche Artikel an, die an sich nur online zur Verfügung stehen. In vielen Fällen stellt sich jedoch eine Offline-Variante als sehr hilfreich und sinnvoll heraus. Hauptsächlich bei Programmierarbeiten beim Kunden der aufgrund Sicherheitsrichtlinien keinen Zugriff in das Internet erlaubt, oder während einer Zugfahrt mit vergessener UMTS-Karte, oder aber auch einfach nur zum Ausdrucken und Durchstöbern.

Derzeit umfasst dieses BlogBook die Einträge von zwei Weblogs zu unterschiedlichen .NET Themen. Um die Inhalte aktuell zu halten, werden diese einmal im Quartal erweitert und überarbeitet um so neue Inhalte zu schaffen, als auch bestehende Inhalte optimaler aufzubereiten.

Folgende Erscheinungstermine werden anvisiert:

- 15. April
- 15. Juni
- 15. September
- 15. Jänner

Neue Versionen können unter <http://www.dotnetcasts.com> bezogen werden und stehen kostenlos zur Verfügung.

1.2. Ziel des Projektes

Wie die meisten Projekte wurde auch für das .NET BlogBook ein Ziel definiert. Unterschiedliche .NET Inhalte (von grundlegenden Beispielen zum .NET Core Framework, über designtechnische Grundlagen bis hin zur Webentwicklung) sollen einfach aufbereitet und übersichtlicher Form vermittelt werden. Das Zielpublikum wird sowohl durch den .NET-Einsteiger, als auch durch den Profi geprägt.

Ebenfalls wird ein weiterer Ausbau dieses Projektes ins Auge gefasst. Vordergründig wird jedoch dieses BlogBook stark erweitert und gepflegt. Durch zusätzliche Autoren und ständige Pflege soll die Qualität auf eine hohe Stufe gehoben werden.

Kurzdefinition: Ein umfassendes How-To-Werk für alle .NET Entwickler und solche, die es noch werden wollen.

1.3. Was wird in diesem Buch nicht abgedeckt?

Kein Ziel ist es, absolute Grundlagen-Themen zu vermitteln, die in einschlägigen Büchern in den ersten Kapiteln zu finden sind. Die ersten Kapitel in diesem Buch beschäftigen sich dementsprechend bereits mit hilfreichen Tipps und Ratschlägen. Es gibt keine Einführung in Schleifen, Verzweigungen und grundlegenden Informationen zum .NET Framework.

Dieses Buch ist auch nicht geeignet, von A bis Z durchgelesen zu werden. Vielmehr werden Tipps und Tricks zu bestimmten Gebieten angeboten. Ein Nachschlagewerk soll es sein.

2. .NET FRAMEWORK

Dieses Kapitel behandelt das .NET Framework. Dies bedeutet hier sind zahlreiche Hilfen, Ratschläge und Diskussionen zu sämtlichen Framework-Themen aufgelistet. Es finden sich Informationen zu Windows Forms, zum Basis-Framework, zu ASP.NET und allem was noch zum .NET Framework gehört.

2.1. Allgemein

2.1.1. [Klassen- und Namespace-Informationen zum .NET Framework erfolgreich finden](#)

Aufgrund eines Kommentars zu meinem Beitrag [C#: Welche Datei repräsentiert einen Prozess](#) möchte ich hier kurz aufführen, wie man Informationen zu Klassen des .NET Frameworks einfach finden kann. Hierfür gibt es unterschiedliche Wege:

Object Browser

Dieser ist äußerst hilfreich, wenn beispielsweise nach Klassen gesucht wird, jedoch der Namespace nicht bekannt ist. Alle Klassen, die in bereits referenzierten Assemblies vorhanden sind, können so gefunden werden. Dies betrifft alle Klassen der .NET Framework Core. Zusätzlich können Methoden-, Property- und Vererbungs-Informationen bezogen werden.

Microsoft Developer Network (MSDN)

Microsoft steckt sehr viel Aufwand und Mühe in das Microsoft Developer Network. Darin finden sich unter zahlreichen Artikeln und Hilfestellungen auch die Dokumentationen zu den einzelnen .NET Framework Versionen. Jede Klasse des .NET Frameworks ist darin aufgelistet und großteils mit Beispielen versehen. So lassen sich die entsprechenden Namespaces finden, als auch Hinweise wie die Klassen verwendet werden, ob sie threadsicher sind und viele weitere Informationen. Ein Muss für jeden .NET Entwickler.

Weitere Ressourcen

Zusätzlich finden sich eine Menge weiterer Ressourcen zum Thema .NET im Internet. Wer Beispiele sucht ist auf [CodeProject](#) gut aufgehoben. Wer ständig aktuelle Informationen, Informationen zu neuen Technologien und/oder Erfahrungsberichte sowie kurze Code-Beispiele sucht, der sollte sich auf [.NET Heute](#) umsehen. Zusätzlich finden sich eine Menge Foren, wie die [MSDN Foren](#).

Zu guter Letzt finden sich viele Personen aus der Community, die doch meistens ein offenes Ohr für den .NET Nachwuchs besitzen. Wer ein wenig guten Willen zeigt, wird sicher nicht abgewiesen.

2.1.2. [.NET: Wo beginne ich als Anfänger?](#)

Diese Frage wird mir immer wieder gestellt. Womit soll begonnen werden, wie beginnt man ein Projekt, was ist zu beachten und viele weitere Fragen warten darauf, beantwortet zu werden.

Nun, im Grunde ist es nicht ganz so einfach und Anfänger haben oft das Gefühl, von der Informationsflut quasi erschlagen bzw. ertränkt zu werden. Doch es ist nicht ganz so kompliziert.

Zu unterscheiden gilt:

Kann ich alles, um ein bestimmtes Projekt erfolgreich abzuschließen.

Welches Wissen fehlt mir?

Wie plane ich ein Projekt von A bis Z?

Die erste Frage muss jeder für sich selbst beantworten. Wenn man etwas nicht kann, muss einfach die notwendige Zeit aufgewandt werden. Komme es wie es wolle. Viele Tests, viele Dokumentationen lesen und mit vielen Menschen sprechen. Das quasi offene Geheimrezept.

Es ist zudem sehr wichtig zu wissen, welche Möglichkeiten man nicht hat, wo Wissen zu diversen Themengebieten fehlt und vor allem natürlich: Wie komme ich an dieses Wissen.

Zur dritten Frage gibt es jede Menge Bücher zum Thema Projektmanagement. Hier möchte ich mit meinen Ergüssen keine Verwirrung stiften. Eher besser ein allgemein anerkanntes Buch schnappen und los geht's.

Im Internet lassen sich zahlreiche Ressourcen – vor allem auch für Anfänger – finden. Es macht sich bezahlt, diese zu konsultieren.

Hier ein paar Starthilfen:

[Galileo Computing - <openbooks>](#)

[The CodeProject](#)

[Microsoft CodePlex](#)

[NET Heute](#)

2.1.3. [.NET Serialisierung und Versionierung](#)

Bei Verwendung der .NET Serialisierung stellt sich bei einer Änderung von zu serialisierenden Objekten oft die Frage, wie diese auf der Gegenseite behandelt werden. Um Probleme beim Deserialisieren auf der Gegenseite zu vermeiden, können die zusätzlichen Eigenschaften mit unterschiedlichen Attributen versehen werden, um dieser Falle Herr zu werden.

Hier die einzelnen Attribute und welche Bedeutung ihnen zukommt:

[NonSerialized()]

Dieses Attribut gibt an, dass die entsprechende Eigenschaft nicht serialisiert wird.

[OptionalField]

Damit wird die entsprechende Eigenschaft als optional gekennzeichnet. Sendet nun

beispielsweise eine ältere Anwendungsversion das Objekt serialisiert zu einer neueren Version, wird das Fehlen der Eigenschaft einfach ignoriert.

[OnDeserializing]

Dieses Attribut wird nicht bei Eigenschaften gesetzt, sondern nur bei einer Methode - und zwar bei ausschließlich einer Methode pro Klasse. Dies wird hauptsächlich für Versionierungszwecken getan. Die Damit gekennzeichnete Methode hat nun die Möglichkeit in den Serialisierungsprozess einzugreifen und die fehlenden Werte zu setzen.

Weitere Informationen und auch Beispiele zu diesen Attributen können durch den MSDN Artikel [Version Tolerant Serialization](#) bezogen werden.

2.2. Basis Framework

2.2.1. [ICloneable: Etwas unsauber, nicht?](#)

Das Interface `ICloneable` wird für gewöhnlich implementiert, wenn das entsprechende Objekt geklont werden soll. Nun ergibt sich aus meiner Sicht hier ein kleines Problem:

`ICloneable` stellt eine Methode `Clone` zur Verfügung. Diese wird für die Klon-Implementierung verwendet. Nun gibt es aber zwei Möglichkeiten zu klonen:

Deep Copy: Alle Objekte werden dupliziert.

Shallow Copy: Nur Objekte oberster Ebene werden vervielfältigt. Alle anderen Objekte stellen Verweise auf die tatsächlichen Objekte dar.

Das eigentliche Problem dieser Schnittstelle besteht nun in folgender MSDN-Aussage:

Clone can be implemented either as a deep copy or a shallow copy.

Dadurch ist nicht klar definiert, wie die `Clone`-Methode nun zu implementieren ist. Bei der Entwicklung eines Frameworks wirft dies das Problem auf, dass der Verwender dieses Frameworks (abgesehen von der möglicherweise recht dürftigen Dokumentation) nicht weiß, ob sein dupliziertes Objekt nun eine vollständige Kopie darstellt, oder eben nicht.

Aus meiner Sicht empfehle ich daher, `ICloneable` nicht zu verwenden und stattdessen eine eigene Implementierung vorzunehmen, die hier eindeutiger ist. Anbieten würde sich die Erstellung zweier Interfaces nach folgendem Muster:

```
public interface IShallowCloneable<T>
{
    T ShallowClone();
}
```

Durch die Implementierung des Interfaces `IShallowCloneable` geht nun eindeutig hervor, dass dabei ein Shallow Copy durchzuführen ist.

```
public interface IDeepCloneable<T>
{
    T DeepClone();
}
```

Ebenso verhält es sich beim Interface `IDeepCloneable`.

Zu guter Letzt noch ein Hinweis auf [Object.MemberwiseClone\(\)](#): `MemberwiseClone` erstellt eine flache Kopie (Shallow Copy) des aktuellen Objektes und könnte für diesen Zweck benutzt werden. Auch hier ein kurzer Auszug aus dem MSDN:

MemberwiseClone kann nicht überschrieben und nur über diese oder eine abgeleitete Klasse aufgerufen werden. Verwenden Sie eine Klasse, die die `ICloneable`-Schnittstelle implementiert, wenn eine tiefe oder flache Kopie eines Objekts öffentlich für die Benutzer bereitgestellt werden muss.

Bei diesem Thema scheint man sich also selbst bei Microsoft nicht 100%ig einig zu sein. Daher also mein Rat für eigene Frameworks: Für den Zweck des Kopierens sollten eigene Interfaces bereitgestellt werden. Dadurch können jegliche Zweifel aus dem Weg geräumt werden. In Kombination mit einer eindeutigen Dokumentation sieht sich der Konsument des Frameworks dadurch mit keinerlei Fehl-Information konfrontiert.

2.2.2. [Paper zur Common Language Infrastructure \(CLI\)](#)

Die CLI-Spezifikation gibt es ja schon einige Jahre (klarerweise), doch haben sich die meisten .NET Entwickler die Paper zur CLI gespart. Hier einfach die Links zu den entsprechenden Dokumenten. Ein Durchlesen erhöht auf jeden Fall das Verständnis für .NET und die Arbeitsweise im Hintergrund.

[CLI Partition I - Concepts and Architecture](#)

[CLI Partition II - Metadata Definition and Semantics](#)

[CLI Partition III - Common Intermediate Language \(CIL\) Instruction Set](#)

[CLI Partition IV - Profiles and Libraries](#)

[CLI Partition V - Annexes](#)

2.2.3. [Objekt auf Eigenschaftsänderungen überprüfen](#)

Es kommt dann doch vor, dass es notwendig ist festzustellen, ob sich Eigenschaften eines Objektes verändert haben. Beispielsweise um beim User nachzufragen, ob er die Änderungen auch tatsächlich speichern möchte.

Nun ist es ziemlich öd, sich alle alten Werte zu merken und mit denen vor der Speicherung zu vergleichen ob sich hier Unterschiede finden.

Stattdessen bietet sich die Implementierung des Interfaces `INotifyPropertyChanged` an.

Dadurch erhält das eigene Objekt ein Event *PropertyChanged* welches ausgelöst wird, wenn eine der Eigenschaften verändert wurde. So fällt jegliche lästige Abfragerei flach und man bekommt genau das was man möchte. Eine Benachrichtigung, wenn sich mindestens eine Eigenschaft verändert hat. Ist doch mal was.

Ein Beispiel aus der Praxis kann dann so aussehen:

```
public class ProgramProperties : BasePropertyClass,
INotifyPropertyChanged
{
    #region Members

    private bool _checkUpdateStartup = false;
    private bool _receiveConfigurationsFromServer = false;
    private string _serverUri = null;

    #endregion

    #region Properties

    public bool CheckUpdateStartup
    {
        get { return this._checkUpdateStartup; }
        set
        {
            if (this._checkUpdateStartup != value)
                NotifyPropertyChanged("CheckUpdateStartup");

            this._checkUpdateStartup = value;
        }
    }

    public bool ReceiveConfigurationsFromServer
    {
        get { return this._receiveConfigurationsFromServer; }
        set
        {
            if (this._receiveConfigurationsFromServer != value)
                NotifyPropertyChanged("ReceiveConfigurationsFromServer");

            this._receiveConfigurationsFromServer = value;
        }
    }

    public string ServerUri
    {
        get { return this._serverUri; }
        set
        {
            if (this._serverUri != value)
                NotifyPropertyChanged("ServerUri");

            this._serverUri = value;
        }
    }

    #endregion
}
```

```
#region INotifyPropertyChanged Members

public event PropertyChangedEventHandler PropertyChanged;

#endregion

#region Private Members

private void NotifyPropertyChanged(String info)
{
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new
PropertyChangedEventArgs(info));
    }
}

#endregion
}
```

2.2.4. Isolierter Speicher für isolierte Daten!

Durch den Namespace *System.IO.IsolatedStorage* ist es möglich, Daten in einem isolierten Speicher abzulegen. Der Vorteil liegt hier klar auf der Hand: Auf diese Art und Weise können Daten gelesen und geschrieben werden, auf die wenig vertrauenswürdiger Code keinen Zugriff erhalten soll. Somit werden vertrauliche Daten nicht zugänglich gemacht und werden daher zusätzlich abgesichert.

Ein weiterer Vorteil: Diese Variante kann zusammen mit servergestützten Benutzerprofilen verwendet werden. So ist der isolierte Speicher des Benutzers dort verfügbar, wo er sich an der Domäne anmeldet.

Wie kann nun in einen isolierten Speicher geschrieben werden?

```
IsolatedStorageFile isolatedFile =
    IsolatedStorageFile.GetStore
        (IsolatedStorageScope.User |
IsolatedStorageScope.Assembly, null, null);
if (isolatedFile != null)
{
    StreamWriter newFile =
        new StreamWriter(
            new IsolatedStorageFileStream("IsolatedStorageTest.txt",
 FileMode.OpenOrCreate));
    if (newFile != null)
    {
        newFile.WriteLine("This is an IsolatedStorage Test");
        newFile.Close();
    }
}
```

Vorerst wird der isolierte Speicher basierend auf eine Scope-Angabe geladen. Der Scope beschreibt hier die gültigen Ebenen, die auf die Informationen zugreifen dürfen. In der weiteren Folge wird dem *StreamWriter* ein *IsolatedStorageFileStream* übergeben, der die Daten in diesen isolierten Speicher schreibt. Und das war es dann auch schon wieder.

Gelesen werden können die Daten folgendermaßen:

```
StreamReader existingFile =  
new StreamReader(new  
IsolatedStorageFileStream("IsolatedStorageTest.txt",  
FileMode.OpenOrCreate));  
if (existingFile != null)  
{  
    Console.WriteLine(existingFile.ReadToEnd());  
    existingFile.Close();  
}
```

Die Verwendung von isoliertem Speicher ist - wie oben zu sehen - also doch sehr einfach gehalten. Dementsprechend empfiehlt es sich auch, diesen tatsächlich zu nutzen, wenn sensible Daten im Spiel sind.

Weitere Informationen zu diesem Thema finden sich natürlich im MSDN: [Verwenden der isolierten Speicherung](#).

2.2.5. [Design und Implementierung der .NET Generics](#)

Wer sich ausführlicher mit der Implementierung der Generics (Parametric Polymorphism) in der CLR auseinandersetzen möchte, dem sei nachfolgender Artikel für den Start (quasi als Entry point) ans Herz gelegt:

[Design and Implementation of Generics for the .NET Common Language Runtime](#)
(PDF, 134 KB)

Weiters empfiehlt sich der MSDN Artikel [Introducing Generics in the CLR](#).

Und zum Schluss noch ein Link, der in die Generics-Programmierung noch ein wenig Licht bringt: [Generics \(C# Programming Guide\)](#).

Wer jetzt noch immer nicht genug von Generics hat, dem sei noch das Projekt [Generics.NET](#) auf [CodePlex](#) empfohlen.

2.2.6. [Funktionsangebot des System.Environment Namespaces](#)

Der *System.Environment*-Namespace bietet einige nette Hilfsmittel, die durchaus nützlich sind. Dabei handelt es sich quasi um eine Allerlei-Sammlung. Die angehängte Beispielanwendung (.NET 2.0) gibt alle verfügbaren Daten in einer *ListView* aus. Dabei handelt es sich um folgende Daten:

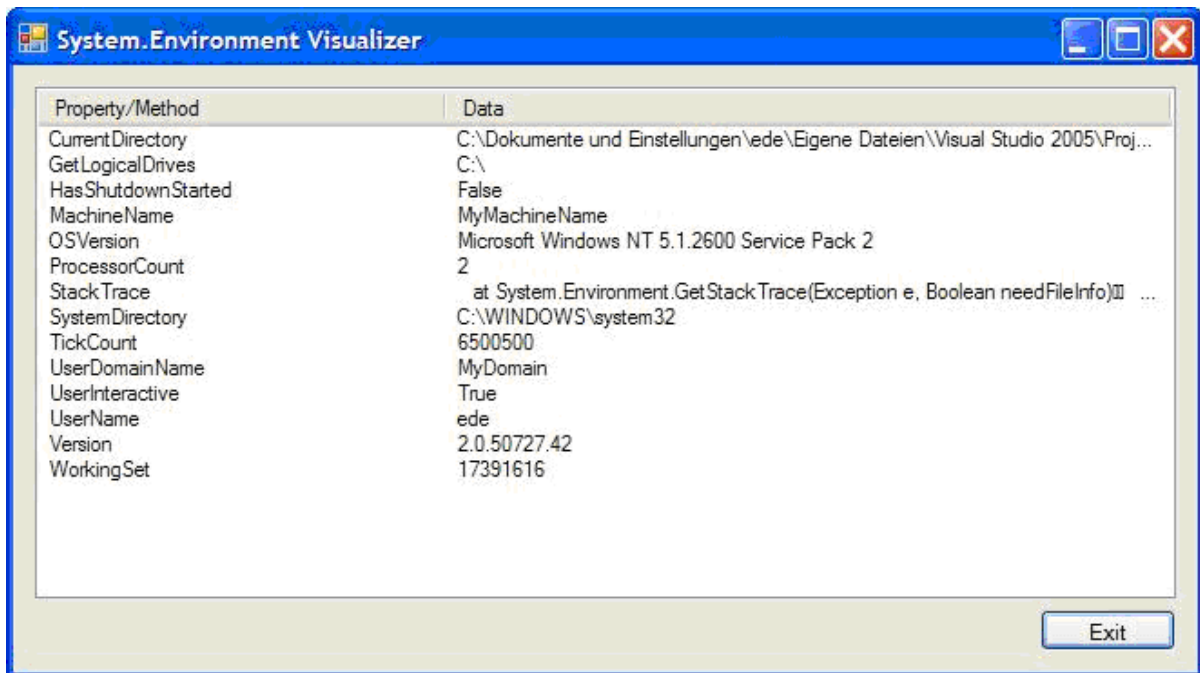


Abbildung 1: System.Environment Visualizer

Zu beachten ist, dass der Namespace auch einige Methoden besitzt, die recht interessant sind:

- Exit
- ExpandEnvironmentVariables
- FailFast
- GetCommandLineArgs
- GetEnvironmentVariable
- GetEnvironmentVariables
- GetFolderPath
- GetLogicalDrives
- SetEnvironmentVariable

Zusammen also eine nette und vor allem hilfreiche Liste. Eine genaue Beschreibung der genannten Methoden findet sich im MSDN unter [System.Environment - Methods](#).

Und hier nun der Download: [System Environment Visualizer](#)

2.2.7. [Der Unterschied zwischen const und static readonly!](#)

const

Mit `const` definierte Werte werden mit der Kompilierung als Konstanten ausgewiesen und sind danach nicht mehr veränderbar.

static readonly

Diese "Markierung" erfolgt zur Laufzeit. Damit gekennzeichnete Werte können zur Laufzeit innerhalb der eigenen Klasse modifiziert werden.

`static readonly` wird hauptsächlich verwendet, wenn der gewünschte Typ nicht als `const` deklariert werden darf, oder wenn der Wert erst zur Laufzeit bekannt ist.

2.2.8. [Standard-Windows-Sounds abspielen](#)

Seit .NET 2.0 gibt es den Namespace `System.Media` und die Klassen `SystemSound` sowie `SystemSounds`. Damit ist es möglich, Standard-Windows-Sounds abzuspielen um den User auf bestimmte Ereignisse, Fehleingaben etc. hinzuweisen.

Der klassische Beep kann mit nachfolgendem Code abgespielt werden.

```
System.Media.SystemSounds.Beep.Play();
```

Natürlich stehen noch weitere Möglichkeiten zur Verfügung. Einfach mal die entsprechenden Klassen näher betrachten.

Zusätzliche Assemblies müssen hierzu nicht eingebunden werden.

2.2.9. [Die wohl häufigste Ausnahme: NullReferenceException](#)

Wer kennt sie nicht, die nachfolgenden Meldungen:

System.NullReferenceException: Object reference not set to an instance of an object
System.NullReferenceException: Der Objektverweis wurde nicht auf eine Objektinstanz festgelegt

Was ist passiert?

Es wurde versucht auf ein Objekt zuzugreifen, welches `null` ist. Im Gegensatz zu Wertetypen müssen Referenztypen instanziiert werden. Erst dann kann ein Zugriff darauf stattfinden.

Beispiel

```
MyObject o = null;  
o.CallMethod(); // NullReferenceException
```

Hier wurde das Objekt nicht instanziiert und führt daher zu einer `NullReferenceException`.

```
MyObject o = new MyObject();  
o.CallMethod(); // keine Exception
```

Da in diesem Beispiel das Objekt instanziiert wurde, wird der Methoden-Aufruf ohne Exception ausgeführt.

Um eine `NullReferenceException` zu vermeiden ist es sinnvoll, vor dem Zugriff auf ein Objekt zu überprüfen ob es `null` ist. Ist dem nicht so, kann "weitergearbeitet" werden.

```
MyObject o = null;
if (o != null)
{
    o.CallMethod();
}
```

Damit wird sichergestellt, dass der Code innerhalb der Bedingung nur ausgeführt wird, wenn das Objekt erfolgreich instanziiert wurde. Und so ganz nebenbei fliegen dem Benutzer weit weniger Exceptions um die Ohren.

Womit wir schon beim nächsten Thema wären (Danke Frank für den Hinweis): Exception-Handling. Natürlich muss nicht nur abgefragt werden, ob das Objekt `null` ist, sondern diese Tatsache ist auch entsprechend zu behandeln. So kann beispielsweise die restliche Aufgabe ohne dieses Objekt nicht korrekt ausgeführt werden. Im Falle einer `if`-Bedingung ist hier ein entsprechender `else`-Zweig notwendig. Besser eignet sich ein entsprechender `try-catch`-Block um diese Ausnahme zu behandeln. So ein Block könnte so aussehen:

```
MyObject o = null;
try
{
    o.CallMethod();
}
catch (NullReferenceException nullException)
{
    // Fehlerausgabe
    // Sonstige Behandlungen
    // Zurücksetzen des Workflows
    // etc.
}
catch (Exception ex)
{
    // eine andere Exception ist aufgetreten
    // diese muss entsprechend behandelt werden
}
finally
{
    // hier Code für Aufräumarbeiten
}
```

2.2.10. [Vermeide: if \(myBoolean == true\)](#)

Oft sieht man Code á la

```
if (myBoolean == true)
```

oder

```
if (myBoolean != true)
```

und jeder hat es zeitweise selbst irgendwo so verwendet. Nichts desto trotz verhält es sich mit diesen Bedingungen wie mit grünem Gras. Oh, Moment. Gras ist ja ohnehin für gewöhnlich grün. Oder war es ein weißer Schimmel, oder weißer Schnee? Nun gut, wieder zurück zum Thema.

```
if (myBoolean)
```

oder

```
if (!myBoolean)
```

ist der saubere Weg dies zu tun - und zeugt auch vom Verständnis von booleschen Variablen.

2.2.11. [Begriffserklärung: Boxing und Unboxing](#)

Für die einen ein alter Hut, für andere ein Grund um auf die Suche nach Informationen und Erklärungen zu gehen. Nun, ein wenig Licht ins Dunkel kann ich hiermit bringen. Es erfolgt nicht nur eine Erklärung der beiden Begriffe, sondern auch Hinweise wann diese Techniken eingesetzt werden sollen/können und wann dies zu vermeiden sind.

Boxing ist die Konvertierung eines Werttypen in einen Verweistyp. Beispiel:

```
int i = 999; // Werttyp  
object o = (object)i; // Konvertierung in einen Verweistyp
```

Unboxing ist die Konvertierung eines Verweistypen in einen Werttyp. Beispiel:

```
object o = 999; // Wert in einem Verweistyp  
int i = (int)o; // Konvertierung in einen Werttype
```

Boxing und Unboxing verursachen zusätzlichen Aufwand, daher sollten diese Operationen vermieden werden. Je häufiger der entsprechende Code-Bereich ausgeführt wird, desto weniger empfiehlt sich die Verwendung.

Es sei an dieser Stelle bemerkt, dass auch der Aufruf von virtuellen Methoden, die Strukturen von `System.Object` erben, unter den Begriff Boxing fallen. (Beispiel: `ToString`).

Empfohlene Vorgehensweisen

Um Boxing und Unboxing zu Vermeiden (und somit auch etwaige Performanceprobleme), empfiehlt es sich, einige Punkte einzuhalten. Hier eine (unvollständige) Liste:

- Werden Strukturen definiert, sollten die Methoden `GetHashCode`, `Equals` und `ToString` überschrieben werden

- Bestehen Methoden die Parameter vom Typ `object` besitzen, die jedoch zur Übergabe von Werttypen verwendet werden, empfiehlt es sich, Überladungen zu definieren, die auf den jeweiligen Werttyp abgestimmt sind.

- Anstatt `object`-Parameter zu verwenden, empfiehlt sich die Verwendung von Generika.

Wann soll Boxing/Unboxing eingesetzt werden

Manche mögen mir jetzt vielleicht widersprechen, aber ich persönlich empfehle Boxing und Unboxing nicht einzusetzen. Selbst bei Code-Teilen die nur selten ausgeführt werden empfiehlt es sich, beispielsweise auf Generika zu setzen oder Überladungen für die entsprechenden Werttypen zu schaffen. Dadurch erhöhen sich zum einen die Übersichtlichkeit, die Verständlichkeit und vor allem auch die Performance.

2.2.12. [C#: Welche Datei repräsentiert einen Prozess](#)

Da manche Prozesse mehrfach ausgeführt werden bzw. werden müssen, ist es manchmal gut zu wissen, welche DLL oder Anwendung der Ursprung eines Prozesses ist. Das nachfolgende Beispiel zeigt, wie eine die Informationen zur gesamten Prozessliste ausgegeben werden. Achtung: Die Prozesse `idle` und `System` besitzen kein `main module`.

```
Process[] processes = Process.GetProcesses();
foreach (Process p in processes)
{
    try
    {
        Console.WriteLine(p.ProcessName + " - " +
p.MainModule.FileName);
    }
    catch (Exception ex)
    {
        Console.WriteLine(p.ProcessName);
        //Console.WriteLine(ex.Message);
    }
}
```

2.2.13. [Zeitmessung einfach gemacht](#)

Soll ein Vorgang performant sein, empfiehlt es sich, unterschiedliche Ansätze zu testen. Diese müssen dann natürlich nicht nur auf Ressourcen-Verbrauch, sondern auch in zeitlicher Hinsicht getestet werden. Dies kann natürlich mit Hilfe von `DateTime` und `TimeSpan` erledigt werden. Das .NET Framework 2.0 enthält hier jedoch auch noch andere Mittel: `Stopwatch` aus dem `System.Diagnostics` Namespace.

`Stopwatch` kann dazu verwendet werden, einfach verbrauchte Zeiten zu messen. Dafür einfach instanzieren und los geht's. `Start` und `Stop` sind die entsprechenden Methoden.

Die Besonderheiten

1. Bei einem `Stop` wird die bereits verbrauchte Zeit nicht zurückgesetzt. D.h. bei einem neuerlichen `Start` beginnt der Zähler nicht bei Null, sondern inkludiert die bereits verbrauchte Zeit. Mit `Reset` kann der Wert auf Null zurückgesetzt werden.
2. Wem die Präzision und die Auflösung der `Stopwatch`-Implementierung nicht genügt, dem seien die Eigenschaften `Frequency` und `IsHighResolution` ans Herz gelegt.

Weitere Besonderheiten und generelle Informationen finden sich unter [1].

Hier noch ein kleines (sinnloses) Beispiel:

```
private void PerformanceTest()
{
    System.Diagnostics.Stopwatch stopWatch = new
System.Diagnostics.Stopwatch();
```

```
stopWatch.Start();  
StringBuilder sb = new StringBuilder();  
for (int i = 0; i < 1000000; i++)  
{  
    sb.Append(" ");  
}  
stopWatch.Stop();  
Console.WriteLine("Milliseconds used: " +  
stopWatch.ElapsedMilliseconds);  
}
```

[1] [MSDN: Stopwatch-Klasse \(System.Diagnostics\)](#)

2.2.14. [Klassen als obsolet \(veraltet\) markieren](#)

Vor allem bei der Entwicklung von Frameworks kommt es immer wieder vor, dass bestimmte Klassen durch andere ersetzt werden, da ein anderes Pattern eingezogen wurde, oder einfach nur eine bessere Implementierung gefunden wurde. Nun sollten diese Klassen nicht sofort aus dem Framework entfernt werden - aus Gründen der Kompatibilität zu älteren Versionen.

Hier bietet es sich an, diese Klassen als obsolet zu markieren. Dies kann folgendermaßen erreicht werden:

```
[Obsolete("Beschreibungstext", false)] // Keine Fehlermeldung  
public class Test { }  
  
[Obsolete("Beschreibungstext", true)] // Fehlermeldung  
public class Test { }
```

Es stellt sich nun lediglich die Frage, wie lange diese Klassen im Framework erhalten bleiben sollten. Ich handhabe dies so, dass obsolete Klassen beim übernächsten Major-Release entfernt werden. Alle Minor-, Build-, oder gar Revision-Versionen werden nicht mit einbezogen. Dies bedeutet nun anhand eines Beispiels:

In der Version 2.0 eines Frameworks wird eine Klasse als obsolete markiert, dann fliegt diese in Version 4.0 tatsächlich hinaus. So bleibt genügend Zeit, auf die neue Klassenstruktur umzustellen.

2.2.15. [Verwendung des Namespace Alias Qualifier \(::\)](#)

Gehen wir von folgendem Code aus:

```
class Program  
{  
    public class System { }  
  
    const int Console = 1;  
    const int number = 2;  
  
    static void Main(string[] args)  
    {  
        Console.WriteLine(number); // Problem  
        System.Console.WriteLine(number); // Problem  
    }  
}
```

```
    }
```

Hier wird eine Klasse `System` erstellt, ohne den System-Namespace des .NET Frameworks zu beachten. Zusätzlich gibt es einen privaten Member namens `Console`. Davon ausgehend können die zwei Aufrufe der `Main`-Methode nicht mehr funktionieren. Wie dem Abhilfe schaffen?

Dafür gibt es den Namespace Alias Qualifier (`::`).

```
    global::System.Console.WriteLine(number); // Funktioniert
```

Und schon funktioniert es wieder.

Der Sinn? Vor allem bei Frameworks kann es passieren, dass Member eines globalen Namespaces (unabsichtlich) ausgeblendet werden. Durch den Namespace Alias Qualifier können diese jedoch ganz normal verwendet werden.

2.2.16. [Der Operator ??](#)

Als ich die Überschrift des Beitrages [Der Operator ?? in C#](#) von [Thomas Woelfer](#) gelesen habe, dachte ich mir zuerst einfach nur: Häh?

Nachdem ich allerdings bei den Vorbereitungen zu einer Zertifizierung genau über diese Frage gestolpert bin, sage ich auf diesem Weg einfach mal Danke ;-) Ist ausserdem ganz nett zu verwenden.

Hier das Beispiel von Thomas, ich hoffe er ist mir nicht böse, dass ich es einfach so mal verwende:

```
void foo( string v)
{
    string r = ( v == null ) ? "null" : v;
}
```

Variante mit dem `??`-Operator:

```
void foo( string v)
{
    string r = v ?? "null";
}
```

2.2.17. [Lese- und Schreibrechte einer Datei überprüfen](#)

Kaum ein Entwickler überprüft bei Dateizugriffe, ob der aktuell angemeldete User auch tatsächlich über die entsprechenden Rechte verfügt. Hintergrund ist wohl, dass die meisten User mit einem Administrator-Account (was zumindest Windows betrifft) angemeldet sind. Hier eine einfache Möglichkeit, die Lese- bzw. Schreibrechte einer Datei abzufragen:

```
public class FileRightsReader
{
```

```
public static bool IsReadable(string filename)
{
    WindowsIdentity principal = WindowsIdentity.GetCurrent();
    if (File.Exists(filename))
    {
        FileInfo fi = new FileInfo(filename);
        AuthorizationRuleCollection acl =
fi.GetAccessControl().GetAccessRules(true, true,
typeof(SecurityIdentifier));
        for (int i = 0; i < acl.Count; i++)
        {
System.Security.AccessControl.FileSystemAccessRule rule =
(System.Security.AccessControl.FileSystemAccessRule)acl;
            if
(principal.User.Equals(rule.IdentityReference))
            {
                if
(System.Security.AccessControl.AccessControlType.Deny.Equals
(rule.AccessControlType))
                {
                    if (((int)FileSystemRights.Read) &
(int)rule.FileSystemRights) == (int)(FileSystemRights.Read)
                    return false;
                }
                else if
(System.Security.AccessControl.AccessControlType.Allow.Equals
(rule.AccessControlType))
                {
                    if (((int)FileSystemRights.Read) &
(int)rule.FileSystemRights) == (int)(FileSystemRights.Read)
                    return true;
                }
            }
        }
    }
    else
    {
        return false;
    }
    return false;
}

public static bool IsWritable(string filename)
{
    WindowsIdentity principal = WindowsIdentity.GetCurrent();
    if (File.Exists(filename))
    {
        FileInfo fi = new FileInfo(filename);
        if (fi.IsReadOnly)
            return false;

        AuthorizationRuleCollection acl =
fi.GetAccessControl().GetAccessRules(true, true,
typeof(SecurityIdentifier));
        for (int i = 0; i < acl.Count; i++)
        {
System.Security.AccessControl.FileSystemAccessRule rule =
(System.Security.AccessControl.FileSystemAccessRule)acl;
            if
(principal.User.Equals(rule.IdentityReference))
```



```
        {  
            if  
(System.Security.AccessControl.AccessControlType.Deny.Equals  
            (rule.AccessControlType))  
            {  
                if (((int)FileSystemRights.Write) &  
(int)rule.FileSystemRights) == (int)(FileSystemRights.Write))  
                    return false;  
            }  
            else if  
(System.Security.AccessControl.AccessControlType.Allow.Equals  
            (rule.AccessControlType))  
            {  
                if (((int)FileSystemRights.Write) &  
(int)rule.FileSystemRights) == (int)(FileSystemRights.Write))  
                    return true;  
            }  
        }  
    }  
    }  
else  
{  
    return false;  
}  
return false;  
}  
}
```

2.2.18. [Mit Dateizugriffs-Rechten arbeiten](#)

In einigen Fällen muss mit den Dateizugriffs-Rechten gearbeitet werden. So muss die ACL (Access Control List) ausgelesen und angepasst werden. Hier ein Code-Snipet, welches den Umgang mit den entsprechenden Klassen aus dem .NET Framework 2.0 zeigt. Kurze Beschreibungen gibt es als Code-Comment.

```
// Creating a testfile  
Console.WriteLine("Creating XML File");  
  
XmlDocument doc = new XmlDocument();  
XmlNode root = doc.CreateElement("root");  
doc.AppendChild(root);  
doc.Save(@"C:temptest.xml");  
  
Console.WriteLine("Getting FileSecurity");  
  
// Getting Access Control List (ACL) of the file  
FileSecurity fSec = File.GetAccessControl(@"C:temptest.xml");  
  
// Get Access Right Type  
Type accessRightType = fSec.AccessRightType;  
Console.WriteLine("AccessRightType: " +  
accessRightType.FullName);  
  
// Get Owner of the file  
string fileOwner =  
fSec.GetOwner(typeof(System.Security.Principal.NTAccount)).Value;
```

```
Console.WriteLine("Owner: " + fileOwner +
System.Environment.NewLine);

// Get Access Rules of the file
AuthorizationRuleCollection authRuleColl =
fSec.GetAccessRules(true, true,
typeof(System.Security.Principal.NTAccount));

// Iterate through all Access Rules
foreach (FileSystemAccessRule rule in authRuleColl)
{
    Console.WriteLine("Control Type : " +
rule.AccessControlType.ToString());
    Console.WriteLine("Identity : " +
rule.IdentityReference.Value);
    Console.WriteLine("Inheritance Flags: " +
rule.InheritanceFlags.ToString());
    Console.WriteLine("Is Inherited : " +
rule.IsInherited.ToString());
    Console.WriteLine("Propagation Flags: " +
rule.PropagationFlags.ToString());
    Console.WriteLine("File System Right: " +
rule.FileSystemRights.ToString());
    Console.WriteLine(System.Environment.NewLine);
}

Console.WriteLine("Adding new Rule");

// Adding a new rule to the file's Access Control List
fSec.AddAccessRule(new
System.Security.AccessControl.FileSystemAccessRule(
System.Security.Principal.WindowsIdentity.GetCurrent().Name,
System.Security.AccessControl.FileSystemRights.Read &
System.Security.AccessControl.FileSystemRights.Write,
System.Security.AccessControl.AccessControlType.Allow));

Console.WriteLine("Setting nun FileSecurity");

// Save Access Control List
File.SetAccessControl(@"C:temptest.xml", fSec);

Console.ReadKey(false);
```

2.2.19. [Assemblies nur zu Reflection-Zwecken laden](#)

Die Assembly-Klasse bietet die Methoden `ReflectionOnlyLoad` und `ReflectionOnlyLoadFrom` um Assemblies in einen eigenen Reflection-Context zu laden.

Damit können Assemblies einfach geladen werden um per Reflection Informationen aus diesen zu beziehen.

Nicht möglich ist damit das Instanzieren von Objekten, ebensowenig werden Abhängigkeiten automatisch mitgeladen. Diese müssen separat nachgeladen werden.

Hinweis: Nur .NET >= 2.0

2.2.20. [List<Person> oder doch lieber eine PersonCollection?](#)

In der letzten Zeit ergeben sich immer wieder des Abends Kurzdiskussionen zu bestimmten Themen. Hier nun die Frage bezüglich typisierter Collections (strongly-typed collections) und wie diese in der Praxis anzuwenden sind.

Anstatt Generics via `List<Person>` zu verwenden, würde sich anbieten, eine `PersonCollection` mittels

```
public class PersonCollection : List<Person> { }
```

zu erstellen und diese zu nutzen. Ist das sinnvoll?

Persönlich bevorzuge ich die Generics-Variante, da hier auf den ersten Blick ersichtlich ist, was auch tatsächlich gemeint ist (Lesbarkeit und Verständnis des Codes). Die zweite Variante würde ich vorziehen, wenn zusätzliche Funktionen implementiert werden sollen.

2.2.21. [Reflection Geschwindigkeitstest](#)

Da ich mir heute wieder mal einen O/R Mapper genauer angesehen habe (Name der Redaktion bekannt), ist mir ein sehr wesentlicher Punkt aufgefallen. Natürlich arbeiten diese Frameworks mit Reflection. Selten wird jedoch tatsächlich auf Geschwindigkeit gesetzt, so eben das Framework, welches ich heute in die Finger bekommen habe. Daraufhin musste ich gleich einen Test machen.

Der Testfall ist ein sehr einfacher, der im Normalfall keine wesentliche Last verursacht. Es wird aus einer Assembly lediglich ein bestimmter Typ aufgrund seines Namens geladen und alle Properties ausgegeben. Die ganze Übung wird 200.000 Mal durchgeführt, was in einer mittelgroßen Anwendung nicht sehr viel ist.

Wenn man davon ausgeht, dass die Engpässe in der Reflection bei quasi rekursiven Durchläufen erst so richtig entstehen, dann sollte mein Testfall nicht sehr gravierend ausfallen. Hier jedoch das Testergebnis auf meinem Rechner:

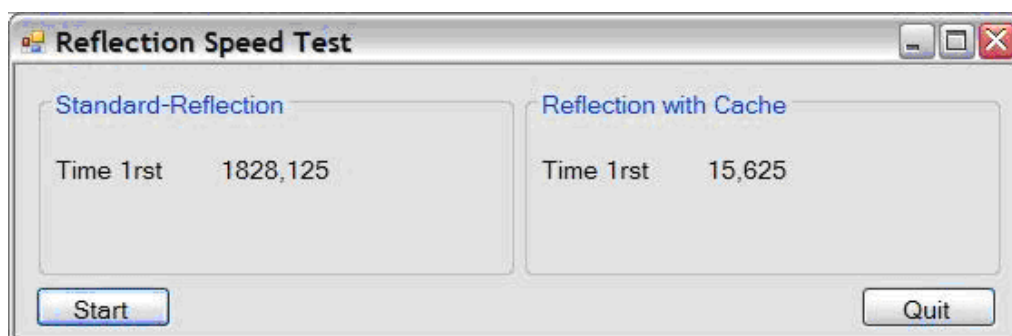


Abbildung 2: Reflection Speed Test

Was genau bedeutet dies? Nun, im ersten Durchlauf wird der nachfolgende Code aufgerufen.

```
for (int i = 0; i < 200000; i++)
{
    Assembly a = Assembly.GetExecutingAssembly();
    if (a != null)
    {
        Type t = a.GetType("ReflectionSpeedTest.Person");
        FieldInfo[] fis = t.GetFields();
        foreach (FieldInfo fi in fis)
        {
            Console.WriteLine(fi.Name);
        }
    }
}
```

Im Vergleich dazu, werden die Feld-Informationen in der zweiten Variante gecached. Das hat zwar den Nachteil, dass der Speicherverbrauch ansteigt, jedoch die Geschwindigkeit um ein Vielfaches erhöht wird - und das bei dieser wirklich sehr einfachen Aufgabe.

```
this.fieldInfos = new Hashtable();

Assembly a = Assembly.GetExecutingAssembly();
if (a != null)
{
    Type t = a.GetType("ReflectionSpeedTest.Person");
    FieldInfo[] fis = t.GetFields();
    foreach (FieldInfo fi in fis)
    {
        this.fieldInfos.Add(fi.Name, fi);
        Console.WriteLine(fi.Name);
    }
}

for (int i = 0; i < 199999; i++)
{
    IDictionaryEnumerator en = this.fieldInfos.GetEnumerator();
    while (en.MoveNext())
    {
        Console.WriteLine(en.Key.ToString());
    }
}
```

Insgesamt ebenfalls wieder 200.000 Aufrufe. Über 1.800 Millisekunden im Vergleich zu knapp mehr als 15 Millisekunden sprechen hierbei schon Bände. Man bedenke, dass oft aufwändigere Probleme mittels Reflection gelöst werden, daher sollte ein Caching durchaus überlegt werden. An sich sehr einfach, jedoch eine große Wirkung.

2.2.22. [MD5-Wert eines Strings](#)

Immer wieder taucht die Frage auf (obwohl an vielen Stellen im Internet auffindbar), wie denn ein String in einen MD5-Hash umgewandelt werden kann. Hier ein kleines Code-Snippet dazu:

```
public static string GetOneWayHash(string val)
{
    byte[] data = System.Text.ASCIIEncoding.ASCII.GetBytes(val);
    MD5 md5 = new MD5CryptoServiceProvider();
    byte[] res = md5.ComputeHash(data);
}
```

```
    return System.Convert.ToBase64String(res, 0, res.Length);  
}
```

Ich möchte hierzu jedoch einen sehr wichtigen Punkt loswerden, der oftmals falsch interpretiert wird:

Ein MD5 ist ein Hashverfahren und keine Verschlüsselung. Dies bedeutet, dass das Zurückgewinnen des ursprünglichen Wertes im Normalfall nicht möglich ist. Warum nur im Normalfall: Beispielsweise sollten Passwörter nie in ihrem originalen Wert abgespeichert werden. Hierfür bieten sich Hashverfahren an, die einen Hashvalue errechnen und diesen anstatt des ursprünglichen Passwortes ablegen. Aus dem resultierenden Wert soll es nicht möglich sein, das eigentliche Passwort zu errechnen. Im Gegensatz zur Verschlüsselung. Hier ist es von absoluter Wichtigkeit, die ursprünglichen Daten in derselben Form zurück zu erhalten.

2.2.23. [Graphics.MeasureString: Maßeinheit der Rückgabe beeinflussen](#)

Immer wieder taucht die Frage auf, in welcher Einheit die Rückgabe der Methode `MeasureString` (Graphics-Object) gehalten ist. Hier die Antwort.

Die Maßeinheit wird über die `PageUnit`-Eigenschaft des Graphics-Objektes beschrieben. Hierzu wird die Enumeration `GraphicsUnit` verwendet. Diese besitzt folgende Werte:

- Display
- Document
- Inch
- Millimeter
- Pixel
- Point
- World

Um also die Breite eines Strings in Pixel zu erfahren, ist vor dem Aufruf von `MeasureString` die Eigenschaft `PageUnit` des Graphics-Objektes auf `GraphicsUnit.Pixel` zu stellen.

[MSDN: Graphics Class](#)

[MSDN: MeasureString Method](#)

[MSDN: PageUnit Property](#)

2.2.24. [Installierte Grafik-Codexs?](#)

Installierte Grafik-Codexs können mit nachfolgendem Code ermittelt werden:

```
ImageCodecInfo[] encoders = ImageCodecInfo.GetImageEncoders();  
foreach (ImageCodecInfo codecInfo in encoders)  
    Console.WriteLine(codecInfo.MimeType);
```

2.2.25. [JPEG Grafiken verlustlos rotieren](#)

Der nachfolgende Code zeigt, wie JPEG Grafiken (und ebenso auch andere Formate) verlustlos rotiert werden können:

```
Image i = Image.FromFile(this.imageFilename);
ImageCodecInfo usedIC = this.GetEncoderInfo("image/jpeg");

System.Drawing.Imaging.Encoder encoder =
System.Drawing.Imaging.Encoder.Transformation;

EncoderParameters encparams = new EncoderParameters(1);
EncoderParameter encparam =
new EncoderParameter(encoder,
(long)EncoderValue.TransformRotate270);
encparams.Param[0] = encparam;

i.Save("filename.jpg", usedIC, encparams );

i.Dispose();
i = null;
GC.Collect();
```

2.2.26. [Die grafische Länge eines Strings mit C# bestimmen](#)

In manchen Fällen (bei der Erstellung von UserControls oder der Verwendung von GDI+) ist es notwendig, die grafische Länge eines Strings zu kennen (also nicht nur die Anzahl der Zeichen). Nachfolgender Code zeigt, wie dies bewerkstelligt werden kann:

```
string test = "This is a test!";

Font font = new Font("Arial", 10.0F);
Graphics g = this.CreateGraphics();
SizeF sizeInfo = g.MeasureString(test, font);
```

this ist in diesem Fall eine Form, kann jedoch genausogut eine PictureBox etc. sein.

2.2.27. [Strong named assembly trotz Referenz auf eine ActiveX Komponente Teil 3](#)

Im zweiten Teil dieser Serie erwähnte ich das Tool AxImp.exe. Ich fand jedoch einen wesentlich einfacheren Weg heraus, diese Aufgabe mit Visual Studio 2003 zu erledigen.

Hierzu ist lediglich das Projekt-Eigenschaften-Fenster des aktuellen Projektes zu öffnen. Unter Common Properties/General finden sich die beiden Einträge

Wrapper Assembly Key File und
Wrapper Assembly Key Name

Im ersten Feld ist der Pfad zum Keyfile anzugeben, im zweiten ein entsprechender Name. Dadurch werden die Wrapper-Klassen automatisch signiert.

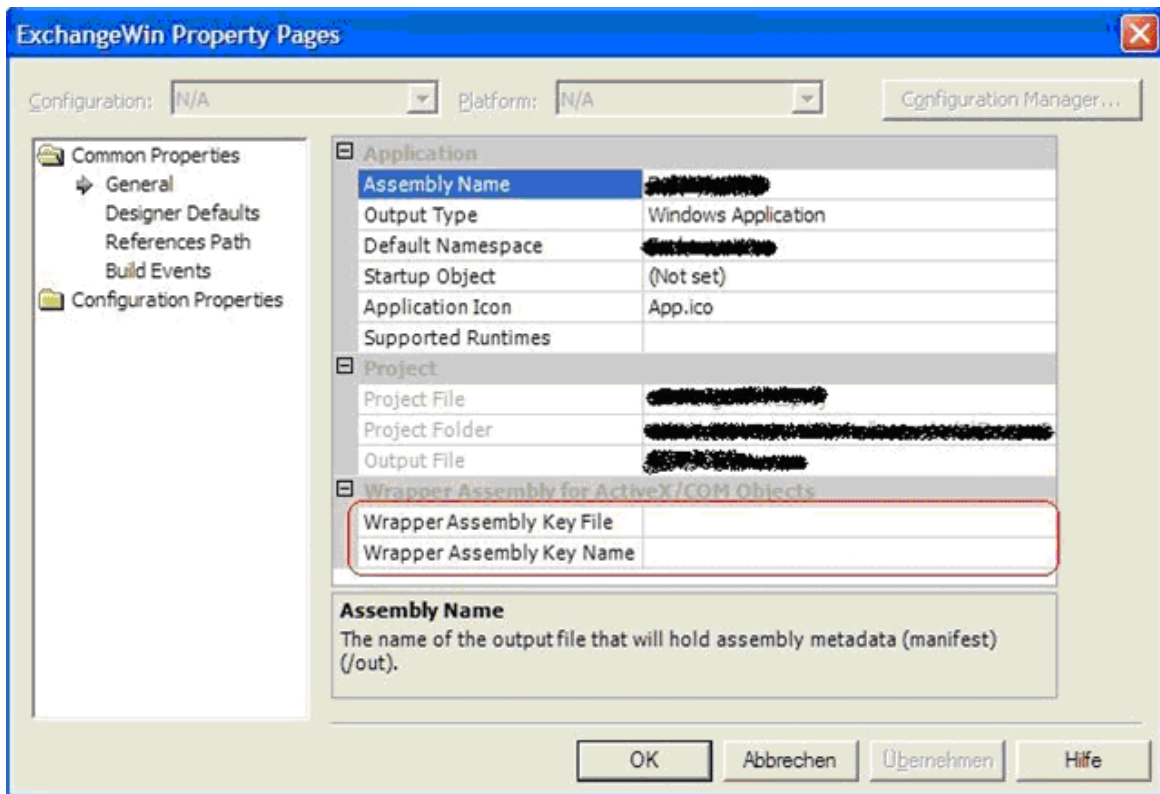


Abbildung 3: Assembly Key File

2.2.28. [Strong named assembly trotz Referenz auf eine ActiveX Komponente Teil 2](#)

Ebenfalls sehr interessant ist das Tool `AxImp.exe`.

Das ActiveX Control Importer Tool konvertiert Typ-Informationen einer COM-Typ-Library in ein Windows Forms Control.

Unter Windows Forms ist es nur möglich Controls zu hosten, die von der Control-Klasse abgeleitet sind. `AxImp.exe` generiert nun einen Wrapper für ActiveX Komponenten welche unter den Windows Forms verwendet werden können.

Um also ein ActiveX Control hosten zu können, muss ein entsprechendes Wrapper-Control erstellt werden. Dieses muss zudem von `AxHost` abgeleitet sein. Das Wrapper-Control enthält dabei eine Instanz des zugrundeliegenden ActiveX Controls und bietet die Möglichkeit darauf zuzugreifen. Alle Methoden, Eigenschaften und Events können genutzt werden.

Um beispielsweise einen entsprechenden Wrapper für das Internet Explorer Control zu erstellen, sind folgende Befehle auszuführen:

```
sn -k shdocvw.snk  
AxImp %WINDIR%\System32\shdocvw.dll /keyfile:shdocvw.snk
```

Die beiden benötigten Dateien `AxSHDocVw.dll` und `SHDocVw.dll` werden daraufhin erstellt.

Für Informationen über `tlbimp.exe` verweise ich auf den ersten Teil [1] dieses Themas.

[1] [Informationen über tlbimp.exe](#)

2.2.29. [Strong named assembly trotz Referenz auf eine ActiveX Komponente](#)

Will man eine Assembly per Strong Name signieren und hält diese Assembly eine Referenz auf eine ActiveX Komponente (beispielsweise `AxInterop.SHDocVw`), dann kommt es zu folgendem Fehler:

```
Assembly generation failed — Referenced assembly 'AxInterop.SHDocVw' does not have a strong name
```

Dieser Fehler ist jedoch recht einfach zu umgehen.

Durchzuführende Schritte

Hierzu benötigen wir den Pfad zur ursprünglichen ActiveX Komponente (hier namentlich als `MyActiveX.dll` geführt). Der Pfad kann mittels der bereits vorhandenen Referenz (Eigenschaften) ermittelt werden. Die ActiveX-Komponente ist nun mittels `tlbimp.exe`.

zu importieren:

```
tlbimp.exe MyActiveX.dll /out:Interop.MyActiveX.dll  
/keyfile:MyKeyFile.snk
```

Nach diesem Vorgang ist die vorhandene Referenz auf die Interop-Assembly zu entfernen und die neu erstellte `Interop.MyActiveX.dll` einzubinden. Ein Rebuild sollte nun fehlerlos durchlaufen.

2.2.30. [Object Mapping oder doch lieber DataBinding?](#)

Oft bekommt man in diversen Foren die Frage zu sehen, ob denn nun im eigenen Projekt ein Object Mapping oder doch ein DataBinding verwendet werden soll. Hier ein paar Punkte - aus meiner Sicht - um diese Frage zu beantworten.

Aus meiner Erfahrung sollte Object Mapping dann verwendet werden, wenn untenstehende Fragen mit Ja beantwortet werden können.

Gibt es jede Menge Data-Objects welche auf ebensolche Tabellen gemappt werden sollen und stammen diese auch alle von der gleichen Basis-Klasse ab?

Könnte der Fall eintreten, dass das zugrunde liegende Datenbank Management System (DBMS) ausgetauscht wird? Sollte das Projekt generell unterschiedliche DBMSs unterstützen?

Sind für die Entwicklung der Lösung mehr als 15 Manntage notwendig?

Soll die Lösung von vielen unterschiedlichen Usern eingesetzt werden? (Open Source Projekt, kostenlose Webanwendung)

Können alle Fragen mit einem klaren Ja beantwortet werden, würde ich persönlich zu einem Object Mapping (beispielsweise NHibernate [1]) raten. In anderen Fällen würde ich dann doch eher ein simples DataBinding vorziehen.

Aber Achtung: Immer gründlich die Zukunft im Auge behalten und nicht immer nur von der Jetzt-Situation ausgehen. Dinge können sich ändern. Wurde einmal eine Entscheidung getroffen, kann diese meist nur mehr sehr schwer geändert werden.

Bei Anregungen oder einfach Dingen die ich nicht bedacht habe, bitte ich einen Kommentar zu hinterlassen.

[1] [NHibernate](#)

2.2.31. [WebRequest und SSL Zertifikate](#)

Muss man via .NET auf Webseiten zugreifen, die per SSL gesichert sind, kommt es hin und wieder zu Problemen, wenn die Zertifikate nicht verifiziert werden können. Diesem Problem kann man aus dem Weg gehen. Dazu einfach die folgende Klasse implementieren:

```
public class TrustAllCertificatePolicy
: System.Net.ICertificatePolicy
{
    public TrustAllCertificatePolicy()
    {}

    public bool CheckValidationResult(ServicePoint sp,
        X509Certificate cert, WebRequest req, int problem)
    {
        return true;
    }
}
```

Danach ist die Policy nur mehr zu aktivieren. Dies kann folgendermaßen erreicht werden:

```
System.Net.ServicePointManager.CertificatePolicy =
new TrustAllCertificatePolicy();
```

Und schon sollte der Zugriff auf diese Webseiten funktionieren.

2.2.32. [C# Beginner: Enumeratoren vs Flags](#)

Enumeratoren sind ja den meisten C# Entwicklern durchaus bekannt. Flags allerdings werden nicht sehr oft eingesetzt. Diese Erfahrung habe ich in diversen Foren gemacht und daher möchte ich dazu ein paar Worte verlieren.

Enumeratoren

Ein Enumerator kann bestimmte vordefinierte Werte enthalten und ist vor allem für Aufzählungen sehr praktisch. Ein Beispiel wäre hierfür der Color-Enumerator. Dieser ermöglicht das einfache Auswählen von Farbwerten. Ein eigenes Beispiel würde wie folgt aussehen:

```
public enum TestEnum
{
    EnumValue0 = 0,
    EnumValue1 = 1
}
```

Für einzelne Items kann hier ein bestimmter Wert definiert werden, muss jedoch nicht. Standardmäßig repräsentiert ein Enumerator einen Int32-Value und beginnt bei 0, ausser anders definiert. Eine Zuweisung sieht beispielsweise wie folgt aus:

```
private TestEnum testEnum = TestEnum.EnumValue0;
```

Zu beachten ist, dass `testEnum` immer nur einen Wert enthalten kann. Dies führt uns nun zur Frage: "Was tun, wenn ich jedoch mehrere Werte speichern möchte?". Ganz einfach:

Flags

Flags können mehrere Werte enthalten. Zuerst jedoch ein Beispiel für die Definition eines Flags:

```
[Flags]
public enum TestFlag
{
    FlagValue1 = 1,
    FlagValue2 = 2,
    FlagValue4 = 4,
    FlagValue8 = 8,
    FlagValue16 = 16
}
```

Im Grunde handelt es sich hierbei um einen Enumerator, dem zusätzlich das Attribut `[Flags]` verpasst wird. Zusätzlich sind die einzelnen Werte entsprechend des Dualsystems (1, 2, 4, 8, 16, 32, ...) anzugeben, da diese bei Mehrfachauswahl durch ein logisches Oder verknüpft werden.

```
TestFlag testf = TestFlag.FlagValue1;
testf := TestFlag.FlagValue2;
```

In diesem Fall hat `testf` den Wert 3. Dieser repräsentiert `FlagValue1` und `FlagValue2`. Durch ein

```
testf -= TestFlag.FlagValue1;
```

reduziert sich der Wert auf 2, wodurch auch nur mehr dieses Flag gesetzt ist. Abfragen können nun folgendermaßen gemacht werden:

```
if ( (testf & TestFlag.FlagValue1) > 0) {}
```

Trifft diese Bedingung zu, ist das Flag gesetzt, andernfalls würde das Ergebnis 0 sein und die Bedingung würde folgerichtig nicht zutreffen.

```
if ( (testf & TestFlag.FlagValue1) == 0) {}
```

Auf diese Weise kann festgestellt werden, ob ein bestimmtes Flag nicht gesetzt ist. Trifft die Bedingung zu ist es nicht gesetzt, andernfalls schon.

Fazit

Dies sollte einen kurzen Einblick in die Welt der Enumeratoren und Flags bieten. Flags bieten in vielen Fällen eine einfache Lösung für Mehrfach-Auswahlen und können auch entsprechend in UserControls abgebildet werden, um dem User eine vereinfachte Darstellung zu bieten.

2.2.33. [System.IO.Compression - Hilfreich oder doch ein Scherz?](#)

Unter dem .NET Framework 2.0 gibt es ja den *System.IO.Compression*-Namespace. Dieser beinhaltet Klassen um Dateien zu zippen. Vorgangswise sieht so aus, dass Daten in einen Stream geschrieben werden (beispielsweise einem GZipStream). Dieser zippt danach die Daten, welche in weiterer Folge in einer Zip-Datei abgelegt werden können.

Dies funktioniert wenn man eine einzelne Datei zippen möchte. Was ist, will ein gesamtes Verzeichnis gepackt werden? Gute Frage. Bis dato konnte ich noch keine Art Zip-Container finden. Dies bedeutet, man muss sich die einzelnen Positionen der Dateien innerhalb des Streams merken bzw. an einer Stelle vermerken, um diese wieder entpacken zu können. Ein handelsübliches Zip-Programm ist damit natürlich überfordert. Dies läßt die Frage offen, ob damit proprietäre Formate unterstützt werden sollen ...

Fazit

Ich hätte mir unter diesem Namespace doch wesentlich mehr vorgestellt. So ist er für mich nicht wirklich zu gebrauchen. Schade drum.

Weiterführende Informationen können unter folgendem Link bezogen werden:
[.NET System.IO.Compression and zip files](#)

2.2.34. [C#-Beginner: Exception-Handling unter C#](#)

Das Thema Exception-Behandlung scheint bei vielen .NET Programmierern noch nicht richtig angekommen zu sein. Immer wieder finden sich in diversen Beispielen und Fragen traurige Konstrukte, die hauptsächlich negative Erscheinungen zu Tage fördern. Also Beispiel sei hier ein neulich gesichteter Code gezeigt werden (nicht kopieren!!!!):

```
try
{
    StreamReader sr = new StreamReader("path");
    string text = sr.ReadToEnd();
    sr.Close();
}
```

```
    }  
    catch (Exception ex) {}
```

Das Ergebnis? Nun, ist die Datei nicht vorhanden wird eine Exception geworfen und auch abgefangen, aber es passiert damit nichts. Es erfolgt weder eine Meldung an den User, noch ein Log-Eintrag, um etwaige Fehler zu einem späteren Zeitpunkt nachvollziehen zu können. Ein weiterer Effekt ist, dass beim "Testen" durch den Entwickler "alles funktioniert" - was natürlich nicht stimmt.

Nun gut, aber wie soll das Exception Handling dann wirklich umgesetzt werden? Ganz einfach. Der grundlegende try-catch-Block sieht so aus:

```
try {  
    // Implementierung  
} catch (IOException ex) {  
    // Fehlerbehandlung für IO-Fehler  
} catch (Exception ex) {  
    // Fehlerbehandlung für andere Fehler  
} finally {  
    // Abschlussarbeiten  
}
```

Hier noch eine genaue Beschreibung:

try: Im try-Teil des gesamten Blockes erfolgt die Implementierung der tatsächlichen Funktion.

catch: Hier ist die Fehlerbehandlung zu implementieren. Dies kann realisiert werden, indem die Fehlermeldungen bzw. zusätzliche Einträge in eine Log-Datei geschrieben werden oder eine User-Interaktion verlangt wird. Wie oben gezeigt, können mehrere Exceptions gezielt abgefangen und behandelt werden. Im MSDN finden sich zu allen Methoden auch Angaben darüber, welche Exceptions geworfen werden. Prinzipiell ist mit diesen Exceptions zu arbeiten und nicht direkt mit Exception selbst.

finally: Dieser Teil des Blockes wird in jedem Fall ausgeführt, also sowohl nach erfolgreichem Durchlauf des try-Teiles, als auch im Falle eines Fehlers. Dadurch bietet es sich an, im finally-Block Aufräumarbeiten durchzuführen. Dies kann beispielsweise das Schließen einer Datenbank-Verbindung sein.

Beim catch-Block muss nicht zwingend ein Typ angegeben werden. Ist dies nicht der Fall, dann werden sämtliche Exceptions behandelt. Von dieser Schreibweise würde ich jedoch eher abraten:

```
try {  
} catch {  
}
```

Des Weiteren sollten Exception nicht für die Ablaufsteuerung einer Anwendung verwendet werden. Darunter wird verstanden, dass gezielt auf Exceptions abgefragt wird, um aufgrund des Exception-Typs zu entscheiden, welcher weitere Code anschließend ausgeführt wird. Exceptions sind sehr "teuer". Dies bedeutet, dass dadurch viele Ressourcen verbraucht werden. Daher sind Exceptions auch als solche zu behandeln.

Natürlich besteht auch die Möglichkeit eigene Exceptions zu implementieren. Dies macht vor allem bei der Entwicklung von größeren Frameworks (die auch von anderen Entwicklern benutzt werden) Sinn. Folgendes Beispiel soll eine eigene Exception verdeutlichen:

```
using System;
class MyException : ApplicationException
{
    public MyException(string str)
    {
        Console.WriteLine(str);
    }
}
```

Wie zu erkennen ist, ist von der Basisklasse `ApplicationException` abzuleiten. Der Konstruktor erhält einen Parameter und damit ist die einfachste Variante einer benutzerdefinierten Exception fertig. Weitere Möglichkeiten können aus dem MSDN bezogen werden.

Zum Schluss möchte ich noch kurz ansprechen, wie der Programmierer selbst Exceptions werfen kann. Dies wird mittels des Schlüsselwortes `throw` getan:

```
throw new MyException("Eine benutzerdefinierte Exception ist aufgetreten");
```

Eine so geworfene Exception muss natürlich auch entsprechend behandelt werden.

Referenzen und weiterführende Artikel:

[1] [Exception Management Architecture Guide](#)

[2] [Exception Class](#)

2.2.35. [C#: Methode mit Parameter via ThreadStart aufrufen](#)

Im heutigen Beitrag zum Thema "C# Beginner" möchte ich ein Beispiel zeigen, wie mittels `ThreadStart` eine Methode inkl. Parameter aufgerufen werden kann.

Dazu ist einfach die entsprechende Methode in eine eigene Klasse mit den notwendigen Properties auszulagern:

```
private class TestClass {
    private string parameter = null;

    public string Parameter {
        get { return this.parameter; }
        set { this.parameter = value; }
    }

    public void Start() {
        // code goes here
    }
}
```

In der ursprünglichen Klasse wird der Thread wie folgt gestartet:

```
TestClass tc = new TestClass();  
tc.Parameter = "test";  
Thread t = new Thread(new ThreadStart(tc.Start)).Start();
```

Das wars dann auch schon wieder.

2.2.36. [Connectionstrings unter C#, VB.NET](#)

Immer wieder wird nach den richtigen Connectionstrings für die unterschiedlichsten Datenbank-Systeme gefragt. Daher mein Tipp: einfach auf <http://www.connectionstrings.com/> nachsehen. Da sollte das meiste zu finden sein.

Und wie wird dieser dann in C# bzw. VB.NET verwendet? Gut, hierfür kann ich ein kleines Beispiel geben (gilt für den Microsoft SQL Server):

C#

```
using System.Data.SqlClient;  
...  
SqlConnection conn = new SqlConnection();  
conn.ConnectionString = "Data Source=(local);" +  
"Initial Catalog=MyDatabaseName;" +  
"Integrated Security=SSPI";  
conn.Open();
```

VB.NET

```
Imports System.Data.SqlClient  
...  
Dim conn As SqlConnection = New SqlConnection()  
conn.ConnectionString = "Data Source=(local);" & _  
"Initial Catalog=MyDatabaseName;" & _  
"Integrated Security=SSPI"  
oSQLConn.Open()
```

Dies gilt nun für den Microsoft SQL Server, der lokal installiert ist. Für einen SQL Server der auf einem anderen Rechner installiert ist, muss lediglich der Connectionstring ausgetauscht und angepasst werden. Die entsprechenden Connectionstrings sind im oben angeführten Link zu finden.

Noch als Zusatzinfo: Die Beispiele funktionieren sowohl bei einem SQL Server 2000, als auch beim SQL Server 2005.

2.2.37. [Strings unter .NET](#)

Dieser Eintrag soll aufführen, wie Strings unter .NET intern behandelt werden. Strings sind unter .NET immutable. Das bedeutet, dass sie nicht veränderbar sind.

Man nehme das folgende Beispiel:

```
string muh = "muh";  
muh += "kuh";
```

Hier sieht das ganze so aus, dass zuerst ein String-Objekt mit dem Value "muh" gebildet wird. Möchten wir an "muh" den String "kuh" anhängen passiert folgendes:

Es wird ein neues String-Objekt am Heap erzeugt. Die Größe (Länge) des String-Objektes beträgt `muh.Length + "kuh".Length`. Nun wird das Ergebnis aus "muh" + "kuh" in das neue String-Objekt kopiert. Das alte String-Objekt muh wird nun für die Garbage Collection freigegeben.

Was sagt uns das jetzt? Es sollte wenn möglich der `StringBuilder` verwendet werden. Dieser zahlt sich aber erst nach einigen String-Operationen aus, da die Instanzierung des `StringBuilders` natürlich auch Kosten verursacht. Ich nehme hier als Richtwert 5 bis 7 String-Operationen.

2.2.38. [Einführung Garbage Collector](#)

Garbage Collection unter C# sieht im Prinzip folgendermaßen aus:

Der GC wird meist dann angetriggert, wenn der Heap-Speicher aufgebraucht sprich voll ist, manuell gestartet wird, oder wenn ein anderes Programm mehr Speicher benötigt, als aktuell noch frei ist.

Der GC iteriert durch den Heap und markiert zuerst jedes Objekt als Garbage, also setzt es auf "zu verwerfen". Danach wird nochmals jedes Objekt rekursiv überprüft, ob Referenzen darauf zeigen - es also noch benutzt wird. Jedes Mal, wenn ein Objekt vom Garbage Collector besucht wird, wird es als erreichbar markiert. Nach diesem Durchlauf steht nun fest, welche Objekte erreichbar sind und welche nicht. Zyklische Referenzen (also Objekt A referenziert Objekt B, welches Objekt C referenziert, welches wiederum Objekt A referenziert) werden ebenfalls entsprechend behandelt.

Im nächsten Schritt wird der Managed Heap durchlaufen und dieser kompaktiert. Das bedeutet, dass Objekte, die verworfen werden können durch Objekte, die im Heap weiter oben gelagert sind überschrieben werden. Dadurch verkleinert sich der Heap und benötigte Objekte wandern weiter nach unten. Es entstehen also keine "Löcher" Heap (Fragmentierung). Dadurch müssen natürlich auch alle Referenzen vom GC geändert werden. Dies ist natürlich ein aufwändiges Unterfangen und aus diesem Grund sollte der GC manuell nur aufgerufen werden, wenn dies unbedingt notwendig ist.

Wann werden Objekte nicht mehr benötigt? Beispielsweise werden in einer Methode unterschiedliche Objekte instanziiert. Wurde diese Methode durchlaufen, werden diese Objekte natürlich nicht mehr benötigt und werden beim nächsten Durchlauf des GC entsprechend behandelt. Sie müssen also nicht explizit auf null gestellt werden. Anders verhält es sich bei Objekten, die "expensive resources" (Dateien, Socket-Verbindungen, Ports, Daten Strukturen etc.) enthalten. Hier bietet .NET die `object finalization` an. Werden also Verbindungen etc. benutzt sollte eine Methode `Finalize()` vorhanden sein. Diese wird bei der Freigabe des Objektes aufgerufen und somit können verwendete Ressourcen auch entsprechend behandelt bzw. tatsächlich freigegeben werden. Hier ist zu beachten, dass die

`Finalize()`-Methode der Basisklasse auch aufgerufen wird. Weiters immer einen `try-catch`-Block darum ziehen.

Eine weitere Möglichkeit besteht durch das `Dispose()`-Pattern. Hier ist eine Methode `Dispose()` zu implementieren. Durch den Aufruf von `Dispose` werden alle teuren Ressourcen entsprechend freigegeben - wie auch in der `Finalize()`-Methode muss hier der Code für die Freigabe der Ressourcen manuell eingetragen werden. Die Ressourcen werden durch diesen Aufruf sofort freigegeben - ausser man wartet auf den Garbage Collector.

Die schönere Variante besteht darin, das `IDisposable()`-Interface zu implementieren. Dadurch wird die Methode `Dispose()` vorgeschrieben und muss entsprechend implementiert werden.

Natürlich gäbe es an dieser Stelle noch mehr zu sagen, aber das sollte als kurzer Überblick durchaus reichen.

2.2.39. [Wie die benötigten Rechte einer Assembly feststellen?](#)

Mir hat sich die Frage gestellt, wie man möglichst einfach die für die Ausführung einer Assembly notwendigen Rechte herausfinden kann. Unter .NET 2.0 ist dies recht einfach durch das Tool `permcalc` möglich.

Aufruf

```
permcalc -Show norberteder.com.lib.dll
```

Ausgabe

Folgende XML-Datei wird erstellt:

```
<?xml version="1.0"?>
<Assembly>
  <Namespace Name="norberteder">
    <Namespace Name="com">
      <Namespace Name="lib">
        <Namespace Name="globalization">
          <Type Name="Translator">
            <Method Sig="instance string get_DefaultLanguage()" />
            <Method Sig="instance void set_DefaultLanguage(string )" />
            <Method Sig="instance string get_CurrentLanguage()" />
            <Method Sig="instance void set_CurrentLanguage(string )" />
            <Method Sig="instance void RegisterGlobalizationFile(string )" />
            <Method Sig="instance void Initialize()" />
          <Demand>
            <PermissionSet version="1" class="System.Security.PermissionSet">
              <IPermission version="1"
                class="System.Security.Permissions.FileIOPermission, mscorlib,
                Version=2.0.0.0, Culture=neutral,
                PublicKeyToken=b77a5c561934e089" PathDiscovery="*AllFiles*" />
            </PermissionSet>
          </Demand>
          <Sandbox>
            <PermissionSet version="1" class="System.Security.PermissionSet">
              <IPermission version="1"
```



```
class="System.Security.Permissions.FileIOPermission, mscorlib,
Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" PathDiscovery="*AllFiles*" />
</PermissionSet>
</Sandbox>
</Method>
<Method Sig="instance string Translate(struct TranslatorSelector
, string )" />
<Method Sig="class Translator get_GetTranslator()" />
<Method Sig="void .ctor()" />
</Type>
<Type Name="TranslatorException">
<Method Sig="instance void .ctor(string )" />
<Method Sig="instance void .ctor(string , class Exception )" />
</Type>
</Namespace>
</Namespace>
</Namespace>
</Namespace>
</Namespace>
</Assembly>
```

Aus diesen Informationen kann genau ausgelesen bzw. abgefragt werden, welche Berechtigungen mind. gesetzt werden müssen um alle Funktionen nutzen zu können.

2.2.40. [Drucken unter C#](#)

Obwohl es ansich relativ viele Ressourcen im Internet zum Thema 'Drucken unter C#' gibt, wird immer wieder die Frage danach aufgeworfen.

Daher an dieser Stelle ein Codebeispiel:

```
PrintDocument printDoc = new PrintDocument();
printDoc.PrintPage += new
PrintPageEventHandler(printDoc_PrintPage);
PrintDialog pd = new PrintDialog();
pd.Document = printDoc;
if (pd.ShowDialog() == DialogResult.OK)
{
    printDoc.Print();
}
```

Was genau passiert hier? Zum Ersten wird ein Dokument angelegt. Dieses stellt quasi die Fläche dar, die ausgedruckt werden soll. Zudem wird ein EventHandler auf das Event PrintPage gelegt um feststellen zu können, wann genau gedruckt wird.

Durch den PrintDialog kann ein Druck-Dialog zur Auswahl des Druckers angezeigt werden.

Durch Aufruf der Methode Print() des PrintDocuments wird der Druck gestartet, wodurch das Event PrintPage ausgelöst wird. Darin muss nun der zu druckende Inhalt in das erhaltene Graphics-Objekt geschrieben werden:

```
private void printDoc_PrintPage
(object sender, PrintPageEventArgs e)
{
    String textToPrint = "Test-Ausdruck";
```

```
Font printFont = new Font("Arial", 18, FontStyle.Bold);  
e.Graphics.DrawString(textToPrint, printFont, Brushes.Black, 10,  
25);  
}
```

In diesem Fall wird angegebene Text mit in der Schrift Arial, Schriftgröße 18 und fett auf den Koordinaten $x = 10$ und $y = 25$ gedruckt.

Es lohnt sich auch, einen Blick auf die `PrintPageEventArgs` zu werfen.

2.2.41. [AppDomains und Memory-Überwachung](#)

Folgende Konstellation:

ein Prozess
mehrere AppDomains

Dies könnte beispielsweise ein Plugin-System sein, in welchem Plugins zur Laufzeit wieder entladen werden können.

Nun macht es doch in manchen Fällen Sinn, eine Überwachung des Speichers einzuführen. Schließlich muss man in bestimmten Umgebungen wissen, wieviel Speicher durch welches Modul verbraucht wird. Und zwar im Echtbetrieb und nicht in einer Testumgebung.

Durch AppDomains können unterschiedliche Module/Anwendungen/etc. wunderbar innerhalb eines Prozesses getrennt werden. Eine Kommunikation durch .NET Remoting ist möglich, jedoch ist jede AppDomain für sich abgeschottet. Das macht auch durchaus so Sinn. Ein Punkt jedoch wurde anscheinend von Microsoft nicht bedacht:

Den Speicherverbrauch kann ich mit .NET Boardmitteln nur für einen Prozess bestimmen - was aber nicht immer ausreicht. Wenn schon mehrere Anwendungs-Domänen in einem Prozess laufen können, dann sollte der Speicherverbrauch auch entsprechend weit heruntergebrochen werden können. Funktioniert aber nicht.

Ausweg? Nun, man sieht sich den Heap-Speicher auf CLR-Ebene an (siehe CLR Profiler). Damit weiß man nun, wieviele Objekte geladen sind, wieviele Referenzen auf ein einzelnes Objekt zeigen und wieviele Speicher sie verbrauchen. Nachteil: Die Daten zu den Objekten liegen allerdings im Stack. Wie an diesen Speicherverbrauch herankommen? Weiß ich noch nicht... muss es aber auch fast eine Lösung geben...

2.2.42. [C# - Daten zwischen zwei Formularen austauschen](#)

Immer wieder wird die Frage gestellt, wie denn Daten zwischen zwei Formularen ausgetauscht werden können.

Hierfür habe ich kurz ein kleines Testprogramm geschrieben, welches genau dies zeigen soll.

Im Projekt sind zwei Formulare zu finden und eine Klasse `DataExchange`. Diese wird von beiden Formularen verwendet und enthält die entsprechenden Daten.

Aber seht euch doch einfach das Projekt an. Bei Fragen kann ich immer noch weiterhelfen ;-)

Download Source [DataExchange.zip](#)

Das Projekt wurde unter .NET 1.1 erstellt, sollte aber auch unter 2.0 lauffähig sein.

2.2.43. [C# und Properties](#)

Properties sind öffentliche Eigenschaften von Klassen, die private Member kapseln. Das klingt jetzt kompliziert, ist es aber nicht:

Prinzipiell ist es so, dass einfache Variablen in Klassen immer privat und daher von ausserhalb der Klasse nicht erreichbar sein sollten. In machen Fällen muss aber auf Werte der Klasse (wenn instanziiert, dann Objekt) zugegriffen werden. Dies kann auf mehrere Arten passieren:

Die Variablen werden mittels `public` als öffentlich markiert
Es werden Properties eingeführt

Variante 1 sollte nicht angewendet werden, also bleibt im Endeffekt nur Variante zwei. Wie sieht das anhand eines Sourcecode-Beispiels aus?

```
public class Test {  
  
    private string name = null;  
  
    public string Name {  
        get { return this.name; }  
        set { this.name = value; }  
    }  
  
}
```

Wie an diesem Beispiel zu sehen, wird in der Variable `name` ein Wert gespeichert. Zugegriffen kann auf diesen Wert mittels der Eigenschaft `Name` werden. Hierzu ist zu beachten, dass Properties (unterschiedlich zu Methoden) kein `()` nach dem Methodennamen enthalten, also auch keine Parameter übergeben bekommen können. Weiters gibt es die Bereiche `get` und `set`. Im `get`-Bereich wird der in `name` gespeicherte Wert zurückgegeben. Der `set`-Bereich dient dazu, den übergebenen Wert in die private Variable zu speichern. Das Schlüsselwort `value` beinhaltet hierbei den übergebenen Wert. Dies sieht ausserhalb der Klasse so aus:

```
Test myTest = new Test();  
myTest.Name = "mein Name";  
Console.WriteLine(myTest.Name)  
  
Consolen-Ausgabe:  
mein Name
```

Nun gut, aber welchen Vorteil hat das ganze nun? Ganz einfach. Durch eine Property kann beispielsweise eine Überprüfung der Werte durchgeführt werden. Zum Beispiel könnte im

set-Bereich die Länge des überprüften Wertes abgefragt werden. Überschreitet dieser eine bestimmte Vorgabe, wird der Wert nicht zugewiesen und der alte Wert bleibt erhalten.

Ein Fehler, der oft gemacht wird, ist folgender:

```
private string name = null;

public string Name {
    get { return this.Name; }
    set { this.Name = value; }
}
```

Was genau passiert hier? Sowohl im get-, als auch im set-Bereich wird immer wieder dieselbe Eigenschaft aufgerufen, was in weiterer Folge zu einem Stack-Überlauf und daher zu einer `StackOverflowException` führt. Hier ist wirklich darauf zu achten, auch tatsächlich die private Membervariable anzugeben.

2.2.44. [AppDomains und ShadowCopies](#)

Wer mit Anwendungs-Domänen zu tun hat, stößt irgendwann auch auf das Thema `ShadowCopy`. Hierfür sind allerdings die vorhandenen Informationen rar gestreut. Das MSDN gibt nicht viel her (nur Oberflächliches) und auch sonst ist dafür nicht viel zu finden. Vor allem nicht, wenn es Probleme gibt. Daher hier eine kleine Anleitung, was bei diesem Thema alles zu beachten ist.

Zuerst kurz beine Begriffserklärung:

`AppDomain`

Eine `AppDomain` ist im Endeffekt nichts anderes, als der Kontext, in dem eine Anwendung ausgeführt wird.

`ShadowCopies`

Dies wird zum Beispiel von ASP.NET betrieben. Assemblies, die vom IIS geladen werden sollen, werden in ein `ShadowCopy`-Verzeichnis kopiert und von dort geladen. Die ursprünglichen Assembly-Dateien werden dadurch nicht gelockt und können von neuen Versionen überschrieben werden.

Aber wie kann man nun selbst `ShadowCopies` realisieren?

Unter der Annahme, dass in einer Anwendung verschiedene `AppDomains` geladen werden, sind drei Dinge notwendig (sämtliche Einstellungen können über die Klasse `AppDomainSetup` gemacht werden):

1. `ShadowCopy` aktivieren

`AppDomainSetup.ShadowCopyFiles` muss hier auf `true` gesetzt werden. Zu beachten ist allerdings, dass es sich bei dieser Property um einen String handelt.

2. Wo sollen die `ShadowCopies` erstellt werden?

Hier gibt es zwei Properties, die interessant sind. `AppDomainSetup.CachePath` und

AppDomainSetup.ApplicationName.CachePath stellt das Verzeichnis dar, in welches die Assemblies kopiert werden sollen. ApplicationName ist der Name der Anwendung. Das ganze sieht nun so aus, dass im CachePath ein Unterverzeichnis mit dem unter ApplicationName angegebenen String erstellt wird. Darunter befinden sich dann die ShadowCopies. Wird darauf verzichtet, kommt der eigentlich dafür zuständige Pfad zum Handkuss: %userprofile%\local settings\application data\assembly. ShadowCopies in diesem Pfad werden automatisch wieder gelöscht. Werden beide Properties korrekt gesetzt und damit ein eigener CachePath verwendet, muss man sich auch selbst um das Löschen der Dateien kümmern.

3. Was soll kopiert werden?

Natürlich muss auch angegeben werden, welche Assemblies kopiert werden sollen. Dies wird in der Property AppDomainSetup.ShadowCopyDirectories eingestellt. Mehrere Pfade werden hier durch ein Semikolon (;) getrennt angegeben. Alle darin befindlichen Assemblies werden dann fürs ShadowCopy herangezogen.

Zum Schluss gibt es auch noch ein wenig Sourcecode. Hier (zur Veranschaulichung" mit fixen Pfaden:

```
AppDomainSetup setup = new AppDomainSetup();  
setup.ApplicationName = "Test";  
setup.ApplicationBase = AppDomain.CurrentDomain.BaseDirectory;  
setup.PrivateBinPath = AppDomain.CurrentDomain.BaseDirectory;  
setup.CachePath = @"C:\temp\cache";  
setup.ShadowCopyFiles = "true";  
setup.ShadowCopyDirectories = Path.Combine  
    (AppDomain.CurrentDomain.BaseDirectory,  
    Path.DirectorySeparatorChar.ToString());
```

Die Kopien würden also im Verzeichnis C:\temp\cache\Test erstellt werden.

2.2.45. [Connection-Probleme zu SQL Server 2005 Express?](#)

Eventuell die nette Meldung provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server?

Die Lösung ist einfach: Im ConnectionString wurde einfach der Instanzname bei der Data Source vergessen. Beispielsweise 127.0.0.1\Instanzname angeben und schon funktioniert alles wie gewollt.

2.2.46. [Wieso gibt es keine Assembly.Unload\(\) Methode?](#)

Jeder, der mit Reflection, AppDomains und Assemblies arbeitet, stellt sich früher oder später die Frage, warum die Assembly-Klasse keine `Unload()`-Methode bietet.

Eine Antwort findet sich in [Jason Zander's WebLog](#). Hier werden die damit zusammenhängenden Probleme gut und ausführlich beschrieben.

2.2.47. [Unable to debug: The binding handle is invalid.](#)

Welch nette Fehlermeldung beim Startversuch einer Anwendung unter dem Visual Studio 2005.

Nach einiger Recherche kam ich dann auf des Rätsels Lösung - und ich war richtig verwundert.

Gestern ging ich meine Dienste am Rechner durch und deaktivierte alles, was ich eigentlich nicht brauch. Darunter auch die Terminal Services. Das war übrigens ein Fehler. Denn Visual Studio 2005 benötigt die Terminal Services um Debuggen zu können. Aber bitte, nicht dass jemand auf die Idee kommt, mich zu fragen, warum das denn so ist.

Merke: Programmiert man mit Visual Studio 2005, niemals die Terminal Services deaktivieren. Einfach auf Manuell lassen und schon freut sich das Visual Studio wieder.

2.2.48. [Dateien vergleichen](#)

Möchte man zwei Dateien vergleichen, kann man leider auf keine fertige Methode im Framework zurückgreifen. Im MSDN findet man allerdings eine [Anleitung](#) inkl. Quellcode um so eine Methode selbst zu schreiben. Eigentlich muss man den Quellcode einfach nur kopieren und in das Projekt einfügen.

Nachfolgend der Quellcode:

```
// This method accepts two strings the represent two files to
// compare. A return value of 0 indicates that the contents of
the files
// are the same. A return value of any other value indicates that
the
// files are not the same.
private bool FileCompare(string file1, string file2)
{
    int file1byte;
    int file2byte;
    FileStream fs1;
    FileStream fs2;

    // Determine if the same file was referenced two times.
    if (file1 == file2)
    {
        // Return true to indicate that the files are the same.
```

```
        return true;
    }

    // Open the two files.
    fs1 = new FileStream(file1, FileMode.Open);
    fs2 = new FileStream(file2, FileMode.Open);

    // Check the file sizes. If they are not the same, the files
    // are not the same.
    if (fs1.Length != fs2.Length)
    {
        // Close the file
        fs1.Close();
        fs2.Close();

        // Return false to indicate files are different
        return false;
    }

    // Read and compare a byte from each file until either a
    // non-matching set of bytes is found or until the end of
    // file1 is reached.
    do
    {
        // Read one byte from each file.
        file1byte = fs1.ReadByte();
        file2byte = fs2.ReadByte();
    }
    while ((file1byte == file2byte) && (file1byte != -1));

    // Close the files.
    fs1.Close();
    fs2.Close();

    // Return the success of the comparison. "file1byte" is
    // equal to "file2byte" at this point only if the files are
    // the same.
    return ((file1byte - file2byte) == 0);
}
```

Den Artikel dazu findet man unter <http://support.microsoft.com/default.aspx?scid=kb;EN-US;320348>

2.2.49. [Ressourcen schonen - Datenbanken besser öffnen und schließen](#)

Viele Programme und insbesondere Asp.NET Seiten verwenden Datenbanken, um Daten zu speichern und zu verwalten. Grundsätzlich sollte jeder Entwickler sorgsam mit Verbindungen zu solchen Ressourcen umgehen, d.h. geöffnete Verbindungen müssen immer geschlossen werden sobald diese nicht mehr benötigt werden. Allerdings ist oft genau das Gegenteil der Fall. Dieser Artikel soll kurz aufzeigen, welche Fehler am häufigsten gemacht werden, und wie man mit einem kleinen Trick diese umgehen kann.

Verbindungen zu solchen Ressourcen wie Datenbanken herzustellen benötigt Zeit. Deshalb arbeitet das .NET Framework mit dem sog. Connection-Pooling. Der Connection-Pool hält eine gewisse Anzahl von möglichen Verbindungen vor und teilt diese den anfragenden Teilen des Programms zu. Dabei wird natürlich darauf geachtet, daß

Verbindungen nicht mehr geöffnet sind, bevor sie neu zugeteilt werden. Sobald alle Verbindungen aufgebraucht sind wird eine entsprechende Ausnahme ausgegeben.

Unter normalen Umständen reicht die Anzahl der möglichen Verbindungen aus. Sollte es doch mal zu einem Engpaß kommen ist oft die Unachtsamkeit des Entwicklers schuld, der eine Verbindung nicht geschlossen hat. Aber selbst wenn dieser alles richtig gemacht hat, können offene Verbindungen zurückbleiben.

Häufig werden Datenbanken nach folgendem Schema geöffnet und geschlossen:

```
string sql = "SELECT * FROM Tabelle1";
OdbcConnection sqlConn = new OdbcConnection();
OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

sqlConn.Open();
OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
while (sqlReader.Read())
{
    Console.WriteLine(sqlReader["ID"]);
}

sqlReader.Close();
sqlConn.Close();
```

Dieser Code erfüllt seinen Zweck: Verbindung zur Datenbank wird geöffnet, Daten werden gelesen, Verbindung zur Datenbank wird geschlossen. Was aber passiert mit der Verbindung, wenn während des Lesens der Daten eine Exception auftritt? Sie bleibt geöffnet, zumindest solange bis der GarbageCollector die Verbindung irgendwann schließt. Tritt dieser Fehler nun öfters auf, verringert sich die Anzahl der möglichen Verbindungen im Connection-Pool, bis eine Exception ausgegeben wird.

Häufig wird jetzt ein Try-Catch-Finally Konstrukt in die Methode eingebaut:

```
string sql = "SELECT * FROM Tabelle1";

OdbcConnection sqlConn = new OdbcConnection();
OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

try
{
    sqlConn.Open();
    OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
    while (sqlReader.Read())
    {
        int id = Convert.ToInt32(sqlReader["Name"]);
    }

    sqlReader.Close();
}
finally
{
    sqlConn.Close();
}
```

Der Code wirkt jedoch unübersichtlich und oft wird im finally Abschnitt trotzdem das Schließen der Verbindung vergessen.

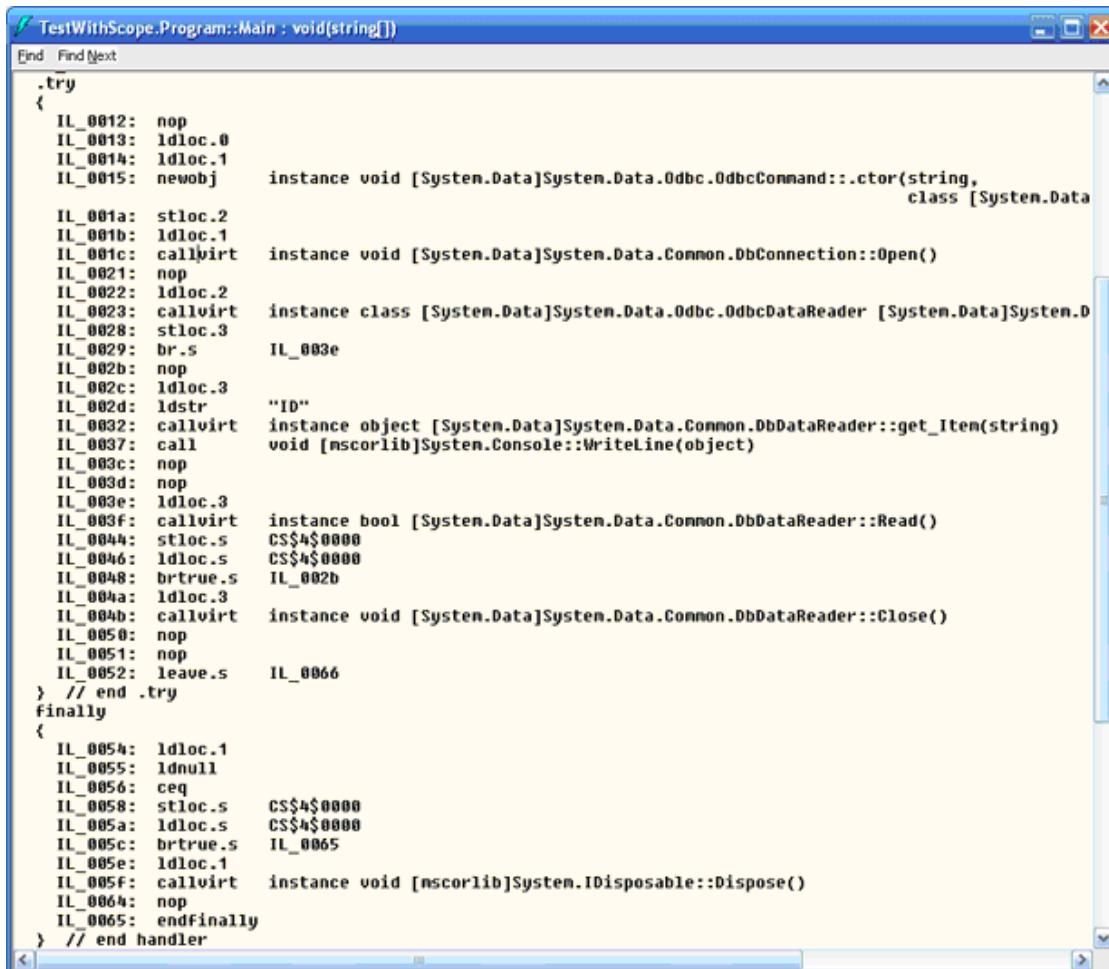
Übersichtlicher Code, der trotzdem im Falle eines Fehlers die Verbindung schließt, erhält man mit sog. „*scopes*“. Ein *scope* ist nichts anderes als ein in geschweifte Klammern eingeschlossener Codeabschnitt. Das Objekt, welches die Verbindung zur Datenbank aufbaut, ist also nur innerhalb dieses *scopes* gültig. Im Code würde das folgendermaßen aussehen:

```
using (OdbcConnection sqlConn = new OdbcConnection())
{
    OdbcCommand sqlCmd = new OdbcCommand(sql, sqlConn);

    sqlConn.Open();
    OdbcDataReader sqlReader = sqlCmd.ExecuteReader();
    while (sqlReader.Read())
    {
        Console.WriteLine(sqlReader["ID"]);
    }
    sqlReader.Close();
    sqlConn.Close();
}
```

Diese Vorgehensweise hat mehrere Vorteile. Zum einen wird der Zugriff auf eine bereits geschlossene Verbindung verhindert, da das entsprechende Objekt außerhalb des *scopes* nicht vorhanden ist. Zum anderen wird beim Auftreten eines Fehlers in jedem Fall die Verbindung zur Datenbank geschlossen, da der Compiler diesen Code intern in einen *try-catch-finally* Block übersetzt. Somit ist es auch möglich das manuelle Schließen der Datenbank ganz wegzulassen. Der Übersicht halber wird es in diesem Beispiel aber trotzdem verwendet.

Mit Hilfe des Tool `ildasm.exe` kann man dies anhand der vom Compiler erzeugten Metadaten überprüfen:



```
TestWithScope.Program::Main : void(string[])
Find Find Next
.try
{
  IL_0012: nop
  IL_0013: ldloc.0
  IL_0014: ldloc.1
  IL_0015: newobj      instance void [System.Data]System.Data.Odbc.OdbcCommand::.ctor(string,
                                                    class [System.Data

  IL_001a: stloc.2
  IL_001b: ldloc.1
  IL_001c: callvirt    instance void [System.Data]System.Data.Common.DbConnection::Open()
  IL_0021: nop
  IL_0022: ldloc.2
  IL_0023: callvirt    instance class [System.Data]System.Data.Odbc.OdbcDataReader [System.Data]System.D
  IL_0028: stloc.3
  IL_0029: br.s     IL_003e
  IL_002b: nop
  IL_002c: ldloc.3
  IL_002d: ldstr     "ID"
  IL_0032: callvirt    instance object [System.Data]System.Data.Common.DbDataReader::get_Item(string)
  IL_0037: call     void [mscorlib]System.Console::WriteLine(object)
  IL_003d: nop
  IL_003e: nop
  IL_003f: ldloc.3
  IL_0040: callvirt    instance bool [System.Data]System.Data.Common.DbDataReader::Read()
  IL_0044: stloc.s   CS$4$0000
  IL_0046: ldloc.s   CS$4$0000
  IL_0048: brtrue.s  IL_002b
  IL_004a: ldloc.3
  IL_004b: callvirt    instance void [System.Data]System.Data.Common.DbDataReader::Close()
  IL_0050: nop
  IL_0051: nop
  IL_0052: leave.s   IL_0066
} // end .try
finally
{
  IL_0054: ldloc.1
  IL_0055: ldnull
  IL_0056: ceq
  IL_0058: stloc.s   CS$4$0000
  IL_005a: ldloc.s   CS$4$0000
  IL_005c: brtrue.s  IL_0065
  IL_005e: ldloc.1
  IL_005f: callvirt    instance void [mscorlib]System.IDisposable::Dispose()
  IL_0064: nop
  IL_0065: endfinally
} // end handler
```

Abbildung 4: Ressourcen schonen - IL-Code

Wichtig ist die Zeile:

```
IL_005f: callvirt instance
void [mscorlib]System.IDisposable::Dispose()
```

Dieser Befehl gibt die Anweisung, daß alle offenen Ressourcen geschlossen werden sollen. Mit Hilfe dieser kleinen Änderung am Code kann man nun davon ausgehen, daß die Verbindung zur Datenbank immer geschlossen wird. Außerdem fängt man weitere mögliche Fehler schon beim kompilieren ab.

[PDF Version](#)

2.2.50. [Download einer Datei via HTTP](#)

Die Klasse WebClient ermöglicht es einen einfachen Download einer Datei via HTTP durchzuführen. Ein Anwendungsfall wäre z.B. der Download einer Bilddatei, die sich auf einem Webserver befindet.

Der Quellcode für den Aufruf beschränkt sich auf einen Zweizeiler, verpackt in einer Methode:

```
private static void DownloadFile(string url)
{ WebClient webClient = new WebClient();
webClient.DownloadFile(url,
Path.GetFileName(url)); }
```

In diesem Beispiel wird die Datei heruntergeladen und lokal unter demselben Namen gespeichert.

2.2.51. [Dateidownload ohne Filesystem](#)

Ab und an möchte der asp.net Entwickler dem User eine zuvor generierte Datei zum Download anbieten. Die normale Vorgehensweise wäre nun die Datei zu erstellen, diese im Filesystem abzuspeichern und dann zum Client zu streamen. Muss man diese Datei archivieren, ist dieses Vorgehen völlig OK. Anders sieht es aus, wenn die Datei nur temporär für den Download erstellt wird. In diesem Fall gibt es eine wesentlich bessere Lösung: Datei erstellen und zum Client streamen.

Fürs Erste benötigen wir eine Methode, die unseren Dateiinhalte in einem String speichert. In diesem Beispiel ist das ein einfacher Dummytext, der 10.000 Mal an den String angehängt wird.

```
private byte[] GenerateFile()
{
    StringBuilder sb = new StringBuilder();

    for (int i = 0; i < 10000; i++)
    {
        sb.Append("Test;Test;Test;Test;Test;Test;Test;Test\r\n");
    }

    string fileContent = sb.ToString();

    return ConvertStringToByteArray(fileContent);
}
```

Dazu braucht man natürlich noch eine Methode, die den Stream zum Client schickt

```
private void DownloadFile(byte[] file, string filename)
{
    Response.AddHeader("Content-disposition", "attachment;
filename=" + filename);
    Response.AddHeader("Content-Length",
file.Length.ToString());
    Response.ContentType = "application/octet-stream";
    Response.BinaryWrite(file);
    Response.End();
}
```

Interessant ist jetzt natürlich die Methode `ConvertStringToByteArray()`. Diese konvertiert, wie der Name schon sagt, einen String in ein ByteArray mit Hilfe der Klasse `UnicodeEncoding`.

```
private byte[] ConvertStringToByteArray(string stringToConvert)
{
    return (new UnicodeEncoding()).GetBytes(stringToConvert);
}
```

Mehr muss man im ersten Moment auch nicht machen. Mit diesen paar Zeilen spart man sich nun den Umweg über das Filesystem und schickt den Stream der Datei direkt zum Client. Das Beispielprojekt kann man [hier](#) herunterladen. .

2.2.52. [Fehlender Namespace](#)

Wahrscheinlich kennt die Shortcuts schon jeder, aber ich bin grade erst durch Zufall drauf gestoßen.

Wenn man Klassen verwendet deren Namespace noch nicht eingebunden ist, springt man entweder mit den Cursor an den Anfang der Datei und fügt diesen per Hand hinzu, oder man verwendet den Shortcut "Shift + Alt + F10".

```
private void GenerateFile()
{
    StringBuilder
}
protected void ... sender, Ev
{
    using System.Text;
    System.Text.StringBuilder
```

Abbildung 5: Fehlender Namespace 1

Sehr nützlich ist auch "Ctrl + Tab". Dieser zeigt eine Übersicht aller geöffneten Dateien und Tool Fenster.

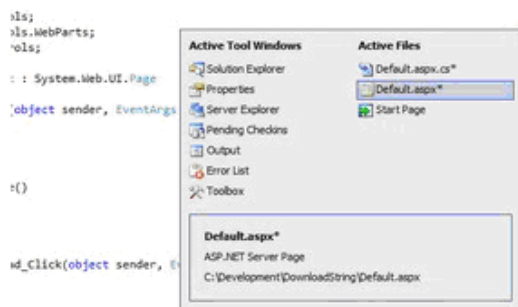


Abbildung 6: Fehlender Namespace 2

2.2.53. [Daten aus dem Clipboard in einer Konsolenanwendung verwenden](#)

Möchte man Daten aus dem Clipboard in einer Konsolenanwendung verwenden, muss diese als Single-Threaded laufen. Möglich macht dies das Attribut `STAThread`, welches vor der `Main()`-Methode plaziert werden muss. In Winforms-Anwendungen wird dieses Attribut standardmäßig eingesetzt.

```
using System;
using System.Windows.Forms;

public class ReadFromClipboard
{
    [STAThread]
    public static void Main()
    {
        IDataObject iData = Clipboard.GetDataObject();
        if (iData.GetDataPresent(DataFormats.Text))
        {
            string str = (String)iData.GetData(DataFormats.Text);
        }
    }
}
```

2.2.54. [Url per WebRequest auslesen](#)

Immer wieder taucht in div. Foren und Newsgroups die Frage auf wie man eine URL einlesen und das Ergebnis in einem string speichern kann. Vor längerer Zeit habe ich mir dafür eine Methode geschrieben, die genau diese Sache erledigt:

```
public static string GetUrlResponse(string url, string
username, string password)
{
    string content = null;

    WebRequest webRequest = WebRequest.Create(url);

    if(username == null || password == null)
    {
        NetworkCredential networkCredential = new
NetworkCredential(username, password);
        webRequest.PreAuthenticate = true;
        webRequest.Credentials = networkCredential;
    }

    WebResponse webResponse = webRequest.GetResponse();

    StreamReader sr = new
StreamReader(webResponse.GetResponseStream(), Encoding.ASCII);
    StringBuilder contentBuilder = new StringBuilder();
    while (-1 != sr.Peek())
    {
```

```
        contentBuilder.Append(sr.ReadLine());  
        contentBuilder.Append("\r\n");  
    }  
    content = contentBuilder.ToString();  
  
    return content.ToString();  
}
```

Der Aufruf sieht wie folgt aus:

```
string responseFromUrl = GetUrlResponse("http://www.google.de",  
null, null)
```

2.2.55. [ApplicationPath in C# Konsolenanwendungen](#)

Ab und zu benötigt man mal den aktuellen Pfad, indem die Konsolenanwendung ausgeführt wird. In Winforms Anwendungen bekommt man dieses über `Application.StartupPath` zurück geliefert. In der Konsolenanwendung fehlt diese Klasse allerdings. Damit ich trotzdem den aktuellen Pfad auslesen kann, habe ich folgende Zeilen geschrieben:

```
static string ApplicationPath  
{  
    get  
    {  
        return  
        Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);  
    }  
}
```

2.2.56. [Read From Clipboard](#)

Vielleicht ganz nützlich. Folgende Funktion liest den Inhalt der Zwischenablage aus und gibt ihn als String zurück:

```
private string ReadFromClipboard() {  
    //Gibt den Inhalt aus dem Clipboard zurück  
  
    string strTextFromClipboard = "";  
    IDataObject iData = Clipboard.GetDataObject();  
    if(iData.GetDataPresent(DataFormats.UnicodeText)) {  
        string strChar =  
        (String)iData.GetData(DataFormats.UnicodeText);  
        strTextFromClipboard = strChar;  
    }  
    return strTextFromClipboard;  
}
```

2.2.57. [Parsing Dates and Times in .NET](#)

Wie Datums- und Uhrzeitwerte geparkt werden können, zeigt der nachfolgende Artikel:
<http://www.stevex.org/dottext/articles/916.aspx>

2.2.58. [String formatting in C#](#)

Hab grade einen sehr interessanten Eintrag in dem Blog von Steve Tibbett gefunden:
<http://www.stevex.org/dottext/articles/158.aspx>

2.3. Windows Forms

2.3.1. [ComboBox als DropDownList kann kein Text gesetzt werden](#)

Wer eine ComboBox verwendet und die Eigenschaft `DropDownStyle` auf `DropDownList` gesetzt hat, kann keinen Text mehr setzen. Dadurch entfällt auch die Möglichkeit, einen Default-Text zu setzen, wenn kein Item ausgewählt ist/wurde. Dem kann durch eine kurze und schnell Ableitung leicht Abhilfe geschafft werden.

```
public partial class ComboBoxEx : ComboBox
{
    private Label _statusLabel = new Label();
    private string _statusText = null;

    public string StatusText
    {
        get { return this._statusText; }
        set { this._statusText = value; }
    }

    public ComboBoxEx()
    {
        InitializeComponent();

        Init();

        this.Controls.Add(this._statusLabel);

        this._statusLabel.Click += new
EventHandler(_statusLabel_Click);
        this.SizeChanged += new
EventHandler(ComboBoxEx_SizeChanged);
        this.SelectedIndexChanged += new
EventHandler(ComboBoxEx_SelectedIndexChanged);
    }

    void _statusLabel_Click(object sender, EventArgs e)
    {
        this.DroppedDown = true;
    }
}
```

```
void ComboBoxEx_SelectedIndexChanged(object sender, EventArgs
e)
{
    if (this.SelectedItem == null)
        Init();
    else
        this._statusLabel.Visible = false;
}

void ComboBoxEx_SizeChanged(object sender, EventArgs e)
{
    Init();
}

public void Init()
{
    if (this.DropDownStyle == ComboBoxStyle.DropDownList)
    {
        this._statusLabel.Visible = true;
        this._statusLabel.Location = new Point(1, 1);
        this._statusLabel.Size = new Size(this.Width - 20,
this.Height - 2);

        if (this._statusText != null)
        {
            this._statusLabel.Text = this._statusText;
            this._statusLabel.Font = this.Font;
        }
        else
        {
            this._statusLabel.Text = "[Nothing selected]";
        }
        this._statusLabel.BringToFront();
    }
    else
    {
        this._statusLabel.Visible = false;
    }
}
}
```

Es wird direkt von der `ComboBox` abgeleitet. Die neue Klasse erhält die Eigenschaft `StatusText` mit dessen Hilfe ein entsprechender Text gesetzt werden kann, der angezeigt wird, wenn kein Item ausgewählt ist.

2.3.2. [Meine GUI friert während der Ausführung ein, was tun?](#)

Ebenfalls eine oft gesehene Frage. Eine Aufgabe wird gerade ausgeführt und dabei friert die Oberfläche der Anwendung ein. Weder ein Fortschrittsbalken wird aktualisiert, noch ein Label, das den Fortschritt anzeigt. Ja selbst beim Verschieben der Form wird diese weiß und zeigt keine Informationen mehr an.

Der Hintergrund liegt einfach daran, dass durch eine ausgeführte Arbeit von dieser Form keine System-Message mehr angenommen wird bzw. diese nicht verarbeitet werden kann.

Um dies zu vermeiden (und damit der Benutzer nicht irrtümlich annimmt, die Anwendung sei abgestürzt), bieten sich mehrere Lösungen an:

DoEvents

Durch ein `Application.DoEvents()` erhält die Anwendung die Möglichkeit, andere Events abzuhandeln und die Form neu zu zeichnen.

Threading

Aufwändige Aufgaben sollten in einem eigenen Thread ausgeführt werden. Dies kann über die Klassen des Namespaces `System.Threading` geschehen oder über einen `BackgroundWorker`. Hier ist jedoch zu beachten, dass von Threads nicht direkt auf die GUI-Elemente zugegriffen werden kann. Hierzu muss mit `Invoke` gearbeitet werden.

Wichtig: Locking

In manchen Fällen ist es wichtig, bestimmte Code-Teile mit einem `lock` (oder anderen entsprechenden Blockierungs-Maßnahmen) zu versehen. Dadurch kann der gleiche Code nur von einem einzigen Thread aufgerufen und ausgeführt werden. Dies kann mit einem `DoEvents` nicht ausreichend gelöst werden. In solchen Fällen sollte auf jeden Fall Threading verwendet werden.

2.3.3. [Menüs dynamisch mit Hilfe einer XML Datei erstellen](#)

In manchen Fällen macht es Sinn, bestimmte Teile einer Anwendung dynamisch erstellen zu lassen. Ob es sich dabei um Menüs handelt, um Eingabefelder, oder auch anderen Dingen. Immer wieder stößt man auf entsprechende Fragen, oft mit keinem wirklich guten Beispiel, vor allem für .NET Newcomer.

Hier nun ein Beispiel für das dynamische Erstellen eines Menüs aus einem XML File heraus.

Wie funktioniert das?

Die XML-Datei beschreibt die Hauptmenü-Einträge als auch die Untermenüs. Informationen wie Name, Text, welche Methode welcher Klasse bei einem Klick aufgerufen werden soll, sowie ob der Menüeintrag beim Startup der Anwendung verfügbar ist, werden darüber gesteuert.

Für jeden Hauptmenü-Eintrag existiert eine eigene Klasse, welche die entsprechenden Methoden zur Verfügung stellt. Für diese werden über den `DynamicMenuHandler` automatisiert Delegates erstellt, die das `Click`-Ereignis an diese Methoden weiterleiten und den dahinterliegenden Code ausführen.

Die gleiche Vorgehensweise kann auch für andere Zwecke verwendet (missbraucht) werden.

[Download Dynamic Menu Creator Beispiel](#)

Sollten zu diesem Beispiel Fragen auftauchen, dann können diese natürlich über die Kommentar-Funktion gestellt werden. Der Download liegt als Visual Studio 2005 Solution vor.

2.3.4. [UserControls eines Namespaces finden](#)

Auf die Frage hin, wie man denn alle UserControls eines bestimmten Namespaces herausfinden kann, um diese dann dynamisch in ein Container-Control zu verfrachten, schrieb ich eine kleine Testanwendung die zeigt, wie man alle Klassen aus einem bestimmten Namespace bekommt. Zusätzlich wird die Information ausgegeben, ob es sich dabei um ein UserControl handelt oder nicht.

[Download Beispiel-Projekt](#) (VS 2005 Solution, 40KB)

Bei dieser Lösung ist jedoch anzumerken, dass sich ein Namespace auch über mehrere Assemblies hinweg erstrecken kann. Dies wurde nicht berücksichtigt. Das Beispiel sollte auch eher einen Denkanstoss geben, als eine fix fertige Solution liefern.

2.3.5. [.NET 2.0: ComboBox und AutoComplete](#)

Unter .NET 2.0 gibt es eine sehr einfache Möglichkeit, der ComboBox eine AutoComplete-Funktion bzw. eine Vorschlags-Funktion zu verpassen.

Dazu einfach folgende Schritte ausführen:

1. `AutoCompleteCustomSource` setzen (zusätzlich zur normalen `DataSource`)
2. `AutoCompleteMode` auf `Suggest` stellen
3. `AutoCompleteSource` auf `ListItems` stellen

Fertig ist die Hexerei und der User freut sich über die verbesserte Usability. Statt `Suggest` (Punkt 2) gibt es auch noch weitere Möglichkeiten um eventuell eigene Einträge anzuhängen etc. Einfach ein wenig ausprobieren.

2.3.6. [C# Beginner: UserControl DoubleTrackBar Beispiel](#)

Deisem Eintrag liegt ein Beispiel bei, welches zeigt, wie ein einfaches `DoubleTrackBar`-Control erstellt werden kann.

Mit Hilfe dieses Controls können mit zwei Schieberegler ein Minimum-Wert und ein Maximum-Wert eingestellt werden. Eigentlich ein recht simples Problem, jedoch nicht für C# Programmierer, die in Themen wie GDI+, UserControls wenig Erfahrung haben.

Der folgende Screenshot zeigt das Aussehen des UserControls in einer kleinen Testanwendung:

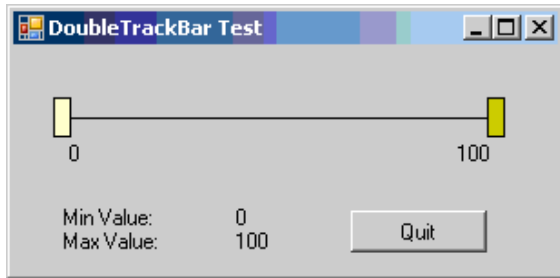


Abbildung 7: Double Trackbar

Zusätzliche Funktionalitäten sollten sich recht einfach einfügen lassen.

Anforderungen: .NET 2.0 für die Solution. Sourcecode auch unter .NET 1.1 nutzbar.

[Download](#)

2.3.7. [C# Beginner: Beispiel für den Aufbau eines Strategie-Spieles](#)

In diversen Foren wird oft nachgefragt, wie denn man denn ein Strategiespiel angehen könnte. Im Vordergrund stehen dabei keine Probleme á la Pathfinding, sondern bereits das Aufbauen des Spielfeldes bereitet oft Probleme.

Daher habe ich aus Lust und Laune eine kleine Demo erstellt, aus der man diverse Ansätze herauslesen kann.

Das Aussehen der Demo hatte hierbei keinen Vorrang und kommt daher mit folgender Oberfläche daher:



Abbildung 8: Matrix Test Game

Nun zur Erklärung einige Punkte:

Darstellungsfläche stellt in diesem Beispiel ein ganz normales Panel dar, welches lediglich zum besseren Erkennen eingefärbt wurde. Wird nun der Button `Generate Matrix` betätigt, wird quasi eine Matrix auf die Spielfläche gelegt. Diese wird automatisch berechnet und kann durch den Parameter `Length` gesteuert werden. Dieser gibt die Seitenlänge des Quadrates an.

Der Button `Add Building` dient dazu, ein `Building` anzulegen. Es ist so vorzugehen, dass der Button zu klicken ist, danach ist auf eine beliebige Stelle des Panels zu klicken. Ist der Platz noch nicht belegt, wird das Gebäude auf diesem Feld "gebaut", andernfalls erscheint die entsprechende Meldung.

Dies ist im Endeffekt auch schon die gesamte Funktionalität dieser Demo. Trotz der Kürze der Entwicklungszeit habe ich dennoch versucht, dies möglichst einfach und erweiterbar zu gestalten, so dass aufgrund dieser "Vorlage" recht schnell unterschiedliche Bauten etc. eingefügt werden können.

Vorstellbar wären sicherlich noch Dinge wie Wiesen, Personen und auch Gebäude, die nicht auf ein Feld beschränkt sind. Ebenfalls müsste noch ein Pathfinding-Algorithmus implementiert werden und danach wäre eine erste spielbare Demo fast fertig.

Sollte ich Zeit finden werden noch die einen oder anderen Erweiterungen in diese Demo fließen, aber grundlegend sollte eine Basis für simple Strategie-Spiele vorhanden sein.

Hier noch das Demoprojekt inklusive Sourcecode: [MatrixTestGame](#)

2.4. ASP.NET

2.4.1. [ViewState und TextBox - Control](#)

Das Attribut `EnableViewState` gibt an, ob der Zustand des zugehörigen Controls im ViewState gespeichert werden soll. Dies ist natürlich auch bei dem `TextBox` Control der Fall. Allerdings ist nach einem `PostBack` der Zustand noch immer vorhanden, obwohl das Attribut auf `false` gesetzt wurde. Der Grund hierfür ist das `PostBack` selbst. Wird ein Formular abgeschickt, werden die Daten per `POST` oder `GET` an die Zielseite geschickt. Da die Zielseite gleich der Quellseite ist, stehen somit die Daten auch über `PostBacks` hinweg zur Verfügung.

Möchte man den Zustand des Controls explizit zurücksetzen, muß dies manuell erledigt werden.

2.4.2. [Bilder im GridView anzeigen](#)

Bilder im GridView anzuzeigen erscheint den meisten erfahrenen ASP.NET Usern als ziemlich einfach. Dennoch wird diese Frage häufig in Foren gestellt. Zusammenfassend kann man sagen dass es wirklich mehr als einfach ist, wenn man nur die richtigen Schritte kennt.

Dieses Beispiel geht davon aus, dass die Bilder im Dateisystem vorliegen und der Name in einem Feld der Datenbank gespeichert ist.

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
Name	varchar(50)	<input checked="" type="checkbox"/>
Surname	varchar(50)	<input checked="" type="checkbox"/>
ImageName	varchar(50)	<input type="checkbox"/>

Abbildung 9: Bilder im GridView 1

Nachdem eine Verbindung zur Datenbank per `SqlDataSource` Control hergestellt wurde, können die angezeigten Felder des GridView über das Kontextmenü und den Punkt *Edit Columns* konfiguriert werden. In diesem Dialog muß ein `ImageField` Control hinzugefügt werden.

Über die Eigenschaften `DataImageUrlField` und `DataImageUrlFormatString` kann nun eingestellt werden, welches Datenbankfeld verwendet werden soll und wo die Bilder im Web vorliegen. Der Platzhalter `{0}` im Wert der Eigenschaft `DataImageUrlFormatString` wird vom GridView Control automatisch mit dem Inhalt des Feldes `DataImageUrlField` ersetzt, so daß ein gültiger Verweis auf ein Bild ausgegeben wird.

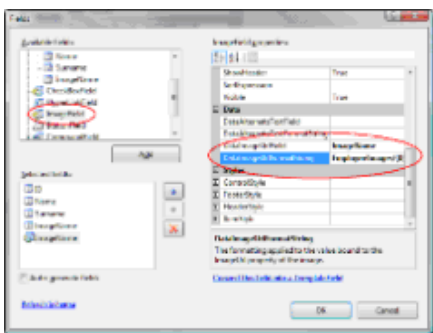


Abbildung 10: Bilder im GridView 2

Das Ergebnis ist ein zusätzliches Feld in Form einer Grafik. Den Download des Beispielprojekts findet man [hier](#).

2.4.3. [Mehrere Spalten im DataTextField der DropDownList](#)

Leider ist es nicht möglich in der Eigenschaft `DataTextField` des `DropDownList` Controls mehr als eine Spalte anzugeben. Allerdings kann man mit Hilfe eines kleinen SQL-Tricks diese Gegebenheit umgehen.

```
SELECT *, (Name + ' ' + Surname) AS FullName FROM [Persons]
```

Diese Syntax fasst zwei vorhandene Spalten zusammen und gibt diese unter dem Namen FullName im Ergebnis aus. Eingebaut in das SqlDataSource Control, kann diese neue Spalte auch schon verwendet werden.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
    SelectCommand="SELECT *, (Name + ' ' + Surname) AS FullName
FROM [Persons]">
</asp:SqlDataSource>

<asp:DropDownList ID="DropDownList1" runat="server"
DataSourceID="SqlDataSource1"
    DataTextField="FullName" DataValueField="ID">
</asp:DropDownList>
```

Das Beispielprojekt kann [hier](#) heruntergeladen werden. .

2.4.4. [Länge eines Strings per Validation Control überprüfen](#)

Leider ist es mit dem ASP.NET RangeValidator nicht ohne weiteres möglich die Länge eines Strings zu überprüfen. Mit Hilfe eines kleinen regulären Ausdrucks und dem RegularExpressionValidator kann man diesen Umstand allerdings umgehen.

```
^\w{1,20}$
```

Dieser Ausdruck lässt nur Eingaben mit der maximalen Länge von 20 Zeichen zu. Eingebaut in den RegularExpressionValidator, ist das Problem auch schon gelöst.

```
<asp:RegularExpressionValidator ID="regexCheck"
    runat="server"
    ErrorMessage="Error!"
    ValidationExpression="^\w{1,20}$" />
```

Möchte man die Eingabe von vornherein begrenzen, kann man das Attribut MaxLength des TextBox Controls benutzen.

```
<asp:TextBox runat="server" MaxLength="20" />
```

2.4.5. [Per Shift-Taste mehrere CheckBox Controls markieren](#)

Die Aufgabe ist es, ein wenig Windows Look & Feel in eine WebForm zu portieren. Genauer gesagt geht es darum mit Hilfe der Maus und der Shift Taste mehrere CheckBox Controls zu markieren. In diesem Beispiel ist eine Liste mit mehreren Produkten gegeben, die jeweils anhand einer CheckBox zum Löschen markiert werden können. Klickt man nun die erste CheckBox der Liste an, hält die Shift-Taste gedrückt und klick anschließend die letzte CheckBox werden automatisch alle zwischenliegenden Controls markiert.

ProductID	ProductName	ProductPrice	
1	Item 1	12,99	<input type="checkbox"/>
2	Item 2	15,99	<input type="checkbox"/>
3	Item 3	16,99	<input type="checkbox"/>
4	Item 4	17,99	<input type="checkbox"/>

Abbildung 11: Mehrere CheckBox Controls markieren

Diese Funktionalität soll per JavaScript umgesetzt werden, um einPostBack und somit einen erneuten Aufbau der Liste zu vermeiden.

Zunächst ist es nötig, eine Liste mit allen vorhandenen CheckBox-Controls IDs aufzubauen. Hierfür bietet sich ein Array an, welches später mit Hilfe einer for-Schleife durchlaufen werden kann.

```
protected void GridView1_DataBound(object sender, EventArgs e)
{
    string js = "chkCollection = new Array(";
    int index = 0;

    foreach (GridViewRow row in GridView1.Rows)
    {
        CheckBox chkBox =
        (CheckBox) row.Cells[3].FindControl("chkBox");
        js += "'" + chkBox.ClientID + "',";

        chkBox.Attributes.Add("onclick", "ShiftClick(" + index +
        ");");
        index++;
    }

    js = js.Substring(0, js.Length - 1) + ");";

    ClientScript.RegisterClientScriptBlock(this.GetType(),
    "ClickScript", js, true);
}
```

Bei dieser Gelegenheit wird jedem CheckBox-Control die Methode ShiftClick hinzugefügt, welche beim Event onclick angerufen werden soll und als Parameter, die Position innerhalb der Arrays entgegen nimmt.

Um die gedrückte Shift-Taste zu erkennen ist es nötig, dem WebForm entsprechende Methoden für die Events onkeydown und onkeyup hinzuzufügen. Wird eine Taste gedrückt, liefern diese Events einen KeyCode zurück der jeweils einer Taste zugeordnet ist. Die Shift Taste gibt den Code 16 zurück. Aufgabe dieser Methoden ist es nun, den KeyCode in der globalen Variable globalKeyCode zu speichern oder bei loslassen der Taste wieder zu löschen.

```
function Document_KeyDown(event)
{
    document.onkeydown = function SetGlobalKeyCode(event)
    {
        globalKeyCode = GetKeyCode(event);
    }
}
```

```
}  
  
function Document_KeyUp(event)  
{  
    document.onkeyup = function ClearKeyCode(event)  
    {  
        globalKeyCode = 0;  
    }  
}  
function GetKeyCode(event)  
{  
    event = event || window.event;  
    return event.keyCode;  
}
```

Wie oben angedeutet, wird die Methode `ShiftClick` aufgerufen sobald ein `CheckBox` Control geklickt wird. Diese Methode stellt zunächst mit Hilfe weiterer globaler Variablen (`firstClick` und `lastClick`) fest, ob bereits eine `CheckBox` vorher markiert wurde und die `Shift`-Taste gedrückt ist. Ist dies der Fall, wird anschließend der Start und das Ende der Schleife ermittelt um die zwischenliegenden Controls zu markieren oder die Markierung zu entfernen.

```
function ShiftClick(clickedPos)  
{  
    lastClick = clickedPos;  
  
    if(globalKeyCode == 16)  
    {  
        if(firstClick > -1 && lastClick > -1)  
        {  
            var i;  
            var start = firstClick < lastClick ? firstClick :  
lastClick;  
            var end = firstClick > lastClick ? firstClick :  
lastClick;  
  
            for(i = start; i <= end; i++)  
            {  
                document.getElementById(chkCollection[i]).checked  
= document.getElementById(chkCollection[lastClick]).checked;  
            }  
        }  
    }  
  
    firstClick = clickedPos;  
}
```

Mit Hilfe dieser paar Zeilen wird dem User ein wenig mehr Komfort bei der Markierung der `CheckBox`-Controls geboten. Gerade bei langen Listen erleichtert dies, die Arbeit erheblich. Zusätzlich zu dem gezeigten Feature sollte noch eine weitere `CheckBox`, die grundsätzlich alle Controls markiert, implementiert werden.

Den Download des Beispielprojekts findet man [hier](#).

2.4.6. [„Wirklich löschen?“ im DetailsView](#)

In einem [anderen Beitrag](#) hatte ich bereits gezeigt, wie leicht es ist eine "Wirklich löschen?" Abfrage im GridView zu implementieren. Nun bekam ich mehrere Anfragen per E-Mail, ob es auch möglich ist diese Abfrage im DetailsView-Control einzufügen.

Natürlich ist dies möglich und ebenfalls sehr leicht zu realisieren. Schlüssel ist hier auch der Befehl *Convert this field into a TemplateField*, welcher im Dialog *Field* zur Verfügung steht.

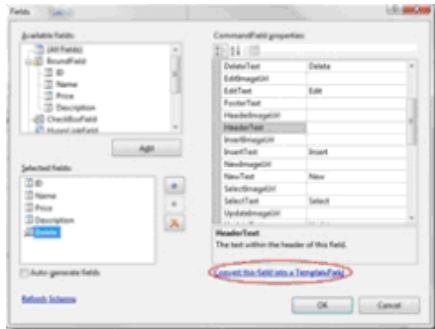


Abbildung 12: "Wirklich löschen" im DetailsView 1

Nach der Konvertierung steht ein vollwertiges LinkButton Control zur Verfügung. Das Attribut `OnClick` des Controls kann nun dazu benutzt werden um eine JavaScript Anweisung auszuführen, sobald der Button geklickt wird.

```
<asp:TemplateField ShowHeader="False">
  <ItemTemplate>
    <asp:LinkButton ID="LinkButton1" runat="server"
      CausesValidation="False" CommandName="Delete"
      Text="Delete" OnClientClick="return confirm('Wirklich
      löschen');"></asp:LinkButton>
  </ItemTemplate>
</asp:TemplateField>
```

Wird nun der Link-Button angeklickt, öffnet sich ein Fenster und fordert den User nochmals zur Bestätigung der Anweisung auf.

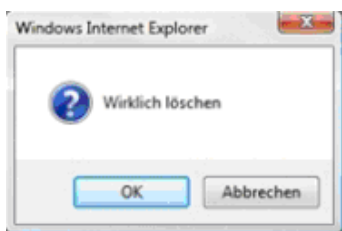


Abbildung 13: "Wirklich löschen" im DetailsView 2

Der Rückgabewert des Dialogs wird per *return* weitergereicht und beendet ggf. die weitere Verarbeitung.

Das Beispielprojekt kann [hier](#) heruntergeladen werden. .

- .
- .
- .

2.4.7. [Login Control ohne returnUrl](#)

Verwendet man Forms Authentication in Verbindung mit den Login-Controls von ASP.NET gibt es ein kleines Feature, dessen Funktion unter bestimmten Umständen unterbunden werden soll. Fordert der User eine Seite an, ohne sich authentifiziert zu haben, wird nach erfolgreichem Login auf diese Seite weitergeleitet. Die angeforderte Seite wird dabei per GET mit dem Key `ReturnUrl` übergeben.

Beispiel:

<http://localhost/Login/default.aspx?ReturnUrl=%2fLogin%2fMember%2fMember.aspx>

Allerdings gibt es viele Anwendungszwecke, bei denen dieses Feature eher hinderlich oder nicht erwünscht ist. Um diese Funktion zu deaktivieren, sind ein paar kleine Schritte nötig. Nach dem erfolgreichen Login, wird von dem Login-Control zunächst ein Event `LoggedIn` aufgerufen.

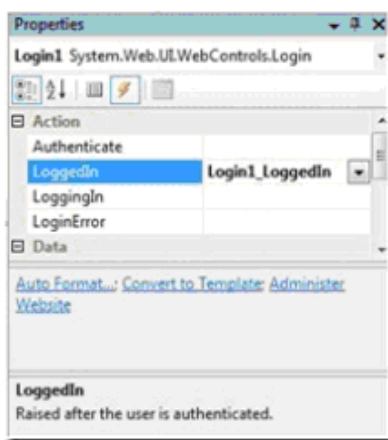


Abbildung 14: Login Control ohne ReturnUrl

Dieser Event muss nun mit einer entsprechenden Methode belegt werden deren Aufgabe es ist, die Weiterleitung per `Response.Redirect()` zu steuern.

```
protected void Login1_LoggedIn(object sender, EventArgs e)
{
    Response.Redirect("Default2.aspx");
}
```

Anschließend wird, unabhängig der `ReturnUrl`, immer auf die gewünschte Seite weitergeleitet.

Den Download des Beispielprojekts findet man [hier](#).

2.4.8. [Literal content is not allowed within a 'skin file'](#).

Zumindest auf den ersten Blick ist diese Fehlermeldung ein wenig verwirrend, deshalb eine kurze Erklärung dazu:

Die mit ASP.NET 2.0 neu hingekommenen skin Files ermöglichen die Definition von Control Styles an einer zentralen Stelle. Fügt man dieser Datei ein Control hinzu, wird die Definition für alle verwendeten Controls der Applikation verwendet. Allerdings darf man an dieser

Stelle tatsächlich auch nur Server-Controls hinzufügen, welche man an dem Attribut `runat="Server"` erkennt. Hat man dieses Attribut vergessen, zeigt Visual Studio o.g. Fehlermeldung an.

Mehr Informationen zum Thema skin Files und ASP.NET Themes findet man unter:

<http://quickstarts.asp.net/QuickStartv20/aspnet/doc/themes/default.aspx>
<http://www.microsoft.com/germany/MSDN/webcasts/library.aspx?id=118767541>
[http://msdn2.microsoft.com/de-de/library/wcyt4fxb\(VS.80\).aspx](http://msdn2.microsoft.com/de-de/library/wcyt4fxb(VS.80).aspx)
<http://www.15seconds.com/issue/040105.htm>

2.4.9. [Login.aspx: Cannot convert type](#)

Grade wieder drüber gestolpert:

Benennt man eine Datei mit dem Namen `Login.aspx`, wird dazu in der Code-Beside Datei die Klasse `Login` angelegt. Was im Development Server von Visual Studio noch funktioniert, wird im IIS mit folgender Fehlermeldung quittiert:

Compiler Error Message:

```
CS0030: Cannot convert type 'ASP.login_aspx' to  
'System.Web.UI.WebControls.Login'
```

Source Error:

```
Line 112: public login_aspx() {  
Line 113: string[] dependencies;  
Line 114: ((Login)(this)).AppRelativeVirtualPath = "~/login.aspx"  
Line 115: if ((global::ASP.login_aspx.@"__initialized" == false)) {  
Line 116: dependencies = new string[1];
```

Anscheinend besteht hier ein Namenskonflikt zwischen der Klasse `Login` im Namespace `System.Web.UI.WebControls` und der Klasse `Login` der eigentlichen Page.

Um dieses Problem zu lösen reicht es einfach die Klasse in der Code-Beside Datei umzubenennen. In einfachsten Fall ergänzt man den Namen um einen Unterstrich.

```
public partial class _Login
```

In der aspx-Datei muß der Name ebenfalls angepasst werden

```
<%@ Page Language="C#" AutoEventWireup="true"  
CodeFile="Login.aspx.cs" Inherits="_Login" %>
```

2.4.10. [Von welcher Methode aufgerufen?](#)

Für eine Logging Routine, welche bestimmte Ereignisse in einer Datenbank speichert, benötigte ich unter anderem den Namespace, die Klasse und den zugehörigen Methodennamen, von wo aus die Logging Routine aufgerufen wurde.

Die Informationen kann man mit Hilfe der Klasse `StackTrace` im `System.Diagnostics` Namespace auslesen werden.

```
System.Diagnostics.StackTrace stackTrace = new
System.Diagnostics.StackTrace();
System.Diagnostics.StackFrame stackFrame =
stackTrace.GetFrame(1);
System.Reflection.MethodBase methodBase = stackFrame.GetMethod();
```

Zunächst wird eine Instanz der Klasse `StackTrace` erstellt und mit Hilfe der Methode `GetFrame(1)` (beginnend bei 0) die Aufrufliste des aktuellen Threads ausgelesen. Nun kann man mit `GetMethod()` eine Verbindung zu der Methode, welche die Logging Routine aufgerufen hat, herstellen.

```
string methodFullName = String.Format("{0}.{1}.{2}",
methodBase.DeclaringType.Namespace,
methodBase.DeclaringType.Name, methodBase.Name);
```

Über die Properties `Namespace` und `Name` des `DeclaringType` kann man, wie der Name schon sagt, den Namespace und den Klassennamen auslesen, während das Property `Name` des `methodBase` Objekts, den wirklichen Namen der Methode ausgibt.

.

2.4.11. [User and Role Management in ASP.NET 2.0](#)

Verwendet man die ASP.NET Membership und Role Provider sucht man evtl. eine Möglichkeit auf einfache Weise die User und Roles zu verwalten, ohne dieses direkt zu programmieren. Normalerweise kann dies über die ASP.NET Web Site Administration Tool erledigt werden, welches allerdings nach dem deployen der Website nicht mehr zur Verfügung steht.

[Brennan Stehling](#) hat sich die Mühe gemacht und Control programmiert, die genau für diese Aufgabe gedacht sind. Eine kurze Beschreibung und das Beispielprojekt findet man unter [dieser](#) URL.

Komischerweise funktioniert der Download des Zip-Archives nur mit FireFox, ansonsten erhält man eine defekte Datei.

Eine Anleitung um das Web Seite Administration Tool im Web zu hosten gibt es im [Blog von Alex](#).

.

2.4.12. [Custom Controls per Web.config registrieren](#)

Wie ich bereits in einem anderen [Beitrag](#) geschrieben hatte, ist es möglich User Controls per Web.Config zu registrieren, und diese somit auf allen Seiten zu nutzen.

Natürlich ist dies auch mit Custom Controls möglich. Auch hier muß im Node *Controls* mit Hilfe des *add* Tags das Control hinzugefügt und somit referenziert werden. Vorher sollte man dieses allerdings per *Add Reference* zum Web hinzugefügt haben.

```
<pages>
  <controls>
    <add assembly="MyCustomControl"
          namespace="MyCustomControl"
          tagPrefix="CC" />
  </controls>
</pages>
```

Natürlich steht jetzt auch wieder die Visual Studio IntelliSense für dieses Control zur Verfügung.

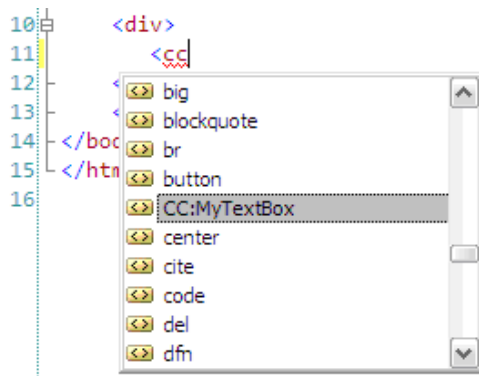


Abbildung 15: Custom Controls und IntelliSense

Den Download des Beispielprojekts findet man [hier](#) .

-
.

2.4.13. [Login-Cookie des Community Server verwenden](#)

Betreibt man neben dem Community Server eine weitere Anwendung, welche auch die ASP.NET Membership Controls nutzt, macht es Sinn das Login-Cookie in beiden Anwendungen zu nutzen. Der User muß sich somit nur einmal einloggend um in beiden Anwendungen arbeiten zu können.

Die Anwendungen selbst müssen innerhalb der gleichen Domain ausgeführt werden, z.B. www.domain.de u. www.domain.de/forum

Weiterhin ist es nötig in beiden Web.Config folgende Einstellungen hinzuzufügen oder anzupassen, falls diese bereits vorhanden sind.

```
<machineKey
  validationKey="xxxx"
  decryptionKey="xxxx"
  validation="SHA1" />
```

Die Werte für die einzelnen Attribute können entweder direkt aus der Web.Config des Community Servers übernommen, oder selbst erstellt werden. Des Weiteren muss der Eintrag Authentication gleich sein

```
<authentication mode="Forms">  
  <forms name=".CommunityServer"  
    protection="All"  
    timeout="60000"  
    loginUrl="/default.aspx"  
    slidingExpiration="true" />  
</authentication>
```

Löscht man nun die Cookies im Browser und meldet sich erneut kann, wird das Cookie ebenfalls in der anderen Anwendung benutzt. .

2.4.14. ["Wirklich löschen?" im GridView](#)

Das GridView in Verbindung mit den DataSource Controls bietet von Haus aus die Möglichkeit Datensätze zu löschen. Allerdings werden die Datensätze ohne eine weitere Bestätigung des Users gelöscht. Für diesen Fall bietet sich ein kurzer JavaScript Dialog an.

Dafür ist es zunächst nötig die Spalten über den Befehl *Edit Columns* zu editieren

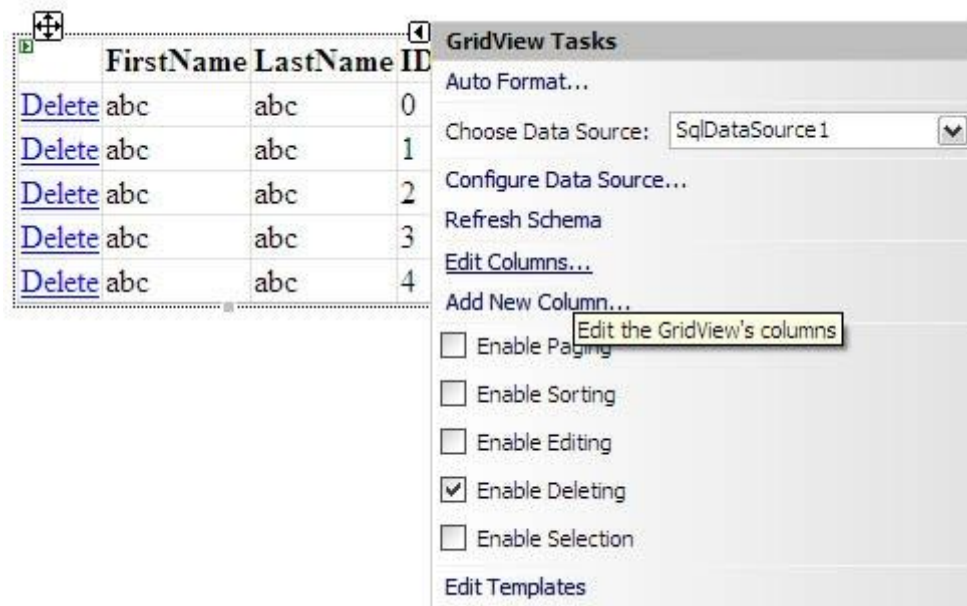


Abbildung 16: "Wirklich löschen" im GridView 1

Das Feld *Delete* muß anschließend in ein *TemplateField* konvertiert werden, welches mit einem Klick auf den Link *Convert this field into a TemplateField* erledigt ist.

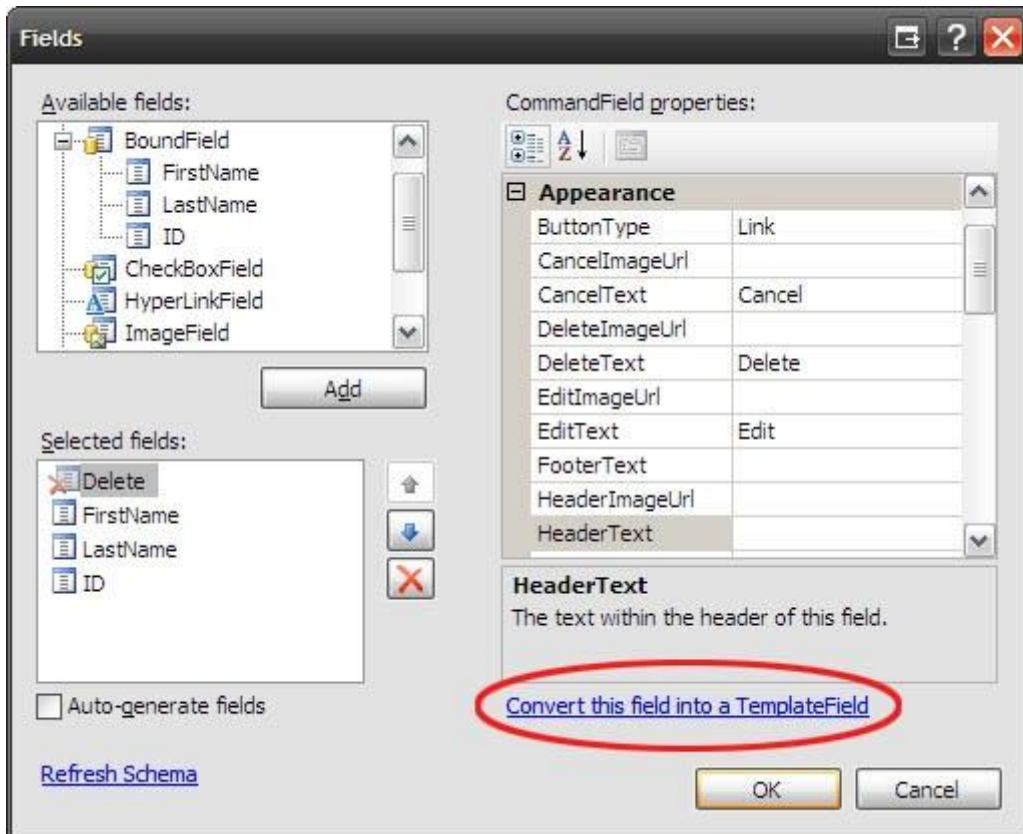


Abbildung 17: "Wirklich löschen" im GridView 2

Nach der Konvertierung steht ein vollwertiges LinkButton Control zur Verfügung. Das Attribut `OnClickClientClick` des Controls kann nun dazu benutzt werden um eine JavaScript Anweisung auszuführen, sobald der Button geklickt wird.

```
<asp:TemplateField ShowHeader="False">
  <ItemTemplate>
    <asp:LinkButton ID="LinkButton1" runat="server"
      CausesValidation="False"
      CommandName="Delete" OnClientClick="return
      confirm('Wirklich löschen');"
      Text="Delete">
    </asp:LinkButton>
  </ItemTemplate>
</asp:TemplateField>
```

Die JS Methode `Confirm` öffnet eine Hinweis-Fenster und bietet die Buttons `OK` und `Abbrechen` an. Entsprechend des Klicks, liefert die Methode anschließend ein `true` oder `false` zurück.



Abbildung 18: "Wirklich löschen" im GridView 3

Der Rückgabewert wird nun per *return* Anweisung weitergereicht und somit die Verarbeitung des JavaScript zum Löschen des Datensatzes abgebrochen oder weitergeführt.

Den Download des Beispielprojekts gibt es [hier](#).

.

2.4.15. [Pop-Up per Response.Redirect\(\)](#)

Immer mal wieder taucht in div.Foren die Frage auf: "Wie kann ich ein Pop-Up per `Response.Redirect()` öffnen". Um die Frage hinreichend zu beantworten, bedarf es einer weiteren Erklärung. Grundsätzlich wird in der Webforms-Entwicklung zwischen zweierarten Code unterschieden. Code, der auf dem Client abgearbeitet wird, und Code, der auf dem Server abgearbeitet wird. Um dies zu verdeutlichen habe ich eine kleine Grafik angefertigt.

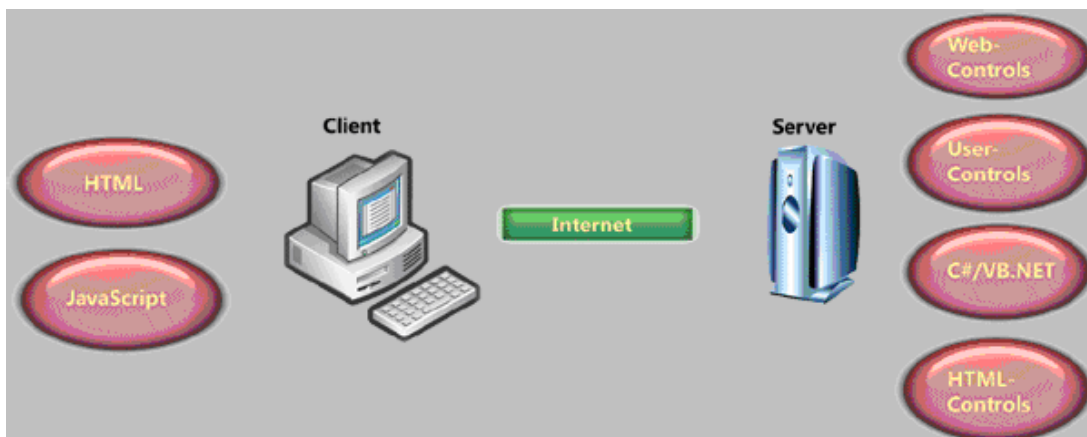


Abbildung 19: PopUp per Response.Redirect

Wie die Grafik zeigt, verarbeitet der Server div. Controls und C# bzw. VB.NET Code. Ist dieser Code verarbeitet, wird nichts anderes als reines HTML an den Client übertragen. Zu diesem Zeitpunkt ist die Arbeit des Servers für diese Anfrage beendet. Dem Client obliegt nun die Interpretation dieses Codes. Da das Öffnen des Pop-Ups eine Aufgabe des Clients ist, beantwortet sich die oben gestellte Frage von selbst. Es ist nicht möglich ein Pop-Up per `Response.Redirect()` zu öffnen, da das `Redirect()` ohne Wissen des Clients serverseitig ausgeführt wird.

Allerdings ist es möglich per C# JavaScript-Code einzufügen, der dann wiederum vom Client beim Aufruf der Seite interpretiert wird. Hierfür bietet sich die Methode `RegisterStartupScript()` an.

```
Page.ClientScript.RegisterStartupScript(this.GetType(),  
"StartPopUp", "window.open('http://www.asp.net');", true);
```

Eine genaue Beschreibung der Methode findet man in der [MSDN](#).

2.4.16. [Read-Only Datensätze im GridView, die Zweite](#)

In [diesem](#) Eintrag beschrieb ich, wie man einzelne Zeilen im GridView auf Read-Only setzt, so daß diese nicht mehr bearbeitet oder gelöscht werden können. Anhand der Spalte *locked* wird entschieden, ob die Command Buttons funktionsfähig sind. Per E-Mail wurde ich nun mehrfach gefragt, ob es auch möglich ist Spalten auf Read-Only zu setzen und die Spalte *locked* nicht anzuzeigen. Hier nun die, zugegeben schnelle, Lösung.

```
protected void GridView1_RowDataBound(object sender,
GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        CheckBox chk =
        (CheckBox)e.Row.Cells[2].FindControl("CheckBox1");

        if(chk.Checked)
            e.Row.Cells[3].Enabled = false;
    }

    e.Row.Cells[2].Visible = false;
}
```

Die Spalte *locked* wird mit Hilfe des Properties *Visible* einfach ausgeblendet. Wichtig ist, daß diese Anweisung außerhalb der if-Anweisung ausgeführt wird, da ansonsten die Überschrift der Spalte weiterhin angezeigt wird. .

.
.
.
.

2.4.17. [Read-Only Datensätze im GridView](#)

Die Aufgabe ist es, bestimmt Zeilen im GridView auf Read-Only zu setzen und somit die Möglichkeit diese zu löschen oder zu bearbeiten zu deaktivieren. In diesem Beispiel wird folgendes Datenbank-Layout verwendet.

Data			
	Column Name	Data Type	Allow Nulls
🔑	ID	int	<input type="checkbox"/>
	Name	varchar(50)	<input type="checkbox"/>
	City	varchar(50)	<input type="checkbox"/>
	locked	bit	<input type="checkbox"/>
			<input type="checkbox"/>

Abbildung 20: Readonly-Datensätze im GridView 1

Die Spalte *locked* zeigt an, ob die Spalte bearbeitet oder gelöscht werden darf. Zunächst muß das SqlDataSource Control konfiguriert werden. Neben dem Connection-String und dem Select Command , werden auch Update u. Delete Command benötigt.

Anschließend kann das GridView konfiguriert und an das SqlDataSource Controls gebunden werden. Bis auf das Feld ID, welches lediglich zur Identifizierung der Datensätze dient, werden alle anderen per Bound- bzw. TemplateField referenziert. Außerdem muß das Feld

locked in Form einer CheckBox angezeigt werden, damit Datensätze manuell gesperrt werden können. In diesem Beispiel ist es nicht möglich einen Datensatz zu bearbeiten, sobald er gesperrt ist. Demnach ist es auch nicht möglich einen Datensatz wieder zu entsperren.

```
<asp:GridView ID="GridView1" runat="server" AllowSorting="True"
AutoGenerateColumns="False"
    DataSourceID="SqlDataSource1" Width="423px"
    DataKeyNames="ID"
    OnRowDataBound="GridView1_RowDataBound">
    <Columns>
        <asp:BoundField DataField="Name" HeaderText="Name"
SortExpression="Name" />
        <asp:BoundField DataField="City" HeaderText="City"
SortExpression="City" />

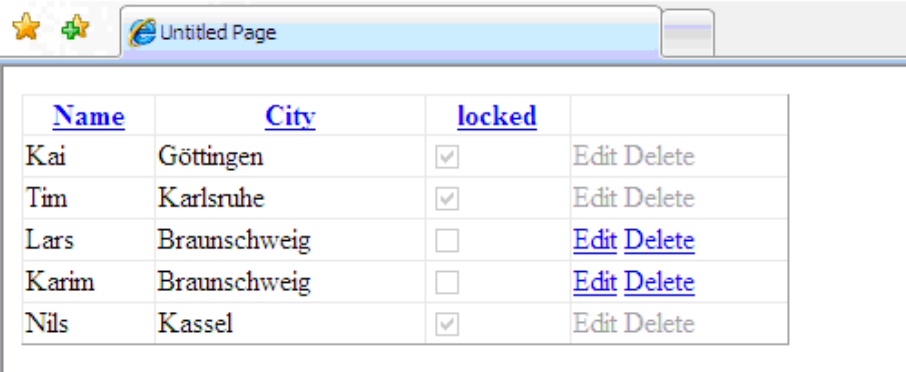
        <asp:TemplateField HeaderText="locked"
SortExpression="locked">
            <EditItemTemplate>
                <asp:CheckBox ID="CheckBox1" runat="server"
                    Checked='<%# Bind("locked") %>' />
            </EditItemTemplate>
            <ItemTemplate>
                <asp:CheckBox ID="CheckBox1" runat="server"
                    Checked='<%# Bind("locked") %>' Enabled="false"
                />
            </ItemTemplate>
        </asp:TemplateField>
        <asp:CommandField ShowDeleteButton="True"
ShowEditButton="True"/>
    </Columns>
</asp:GridView>
```

Wichtig ist nun das Attribut OnRowDataBound. Dieses gibt an welche Methode aufgerufen wird, sobald das Event OnRowDataBound vom GridView gestartet wird. Wie der Name schon sagt, wird dieser Event direkt nachdem ein Datensatz an das GridView gebunden wird aufgerufen. Aufgabe dieser Methode ist es nun zu überprüfen, ob der Datensatz gesperrt ist und ggf. die Command-Buttons zum Editieren oder Löschen zu deaktivieren.

```
protected void GridView1_RowDataBound(object sender,
GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        CheckBox chk =
        (CheckBox)e.Row.Cells[2].FindControl("CheckBox1");

        if(chk.Checked)
            e.Row.Cells[3].Enabled = false;
    }
}
```

Zunächst wird überprüft, ob es sich bei der aktuellen Zeile um eine Datensatz-Zeile handelt. Diese ist nötig, da beim Hinzufügen von Überschriften und Fußzeilen ebenfalls dieser Event aufgerufen wird. Anschließend sucht die Methode FindControl() die CheckBox anhand der definierten ID. Ist diese nun angehakt, wird die dritte Zelle der aktuellen Zeile des GridView deaktiviert. Das Ergebnis sind gesperrte Command-Buttons im GridView:



<u>Name</u>	<u>City</u>	<u>locked</u>	
Kai	Göttingen	<input checked="" type="checkbox"/>	Edit Delete
Tim	Karlsruhe	<input checked="" type="checkbox"/>	Edit Delete
Lars	Braunschweig	<input type="checkbox"/>	Edit Delete
Karim	Braunschweig	<input type="checkbox"/>	Edit Delete
Nils	Kassel	<input checked="" type="checkbox"/>	Edit Delete

Abbildung 21: Readonly-Datensätze im GridView 2

Das Beispielprojekt kann [hier](#) heruntergeladen werden. .

2.4.18. [Dynamischer Meta-Refresh in MasterPage](#)

Heute wurde in einem Forum die Frage gestellt, wie man eine Html-Seite (bzw. Web Form) nach einer fest definierten Zeit aktualisieren kann. Hier bietet sich ein Meta-Refresh an, welcher im Head-Tag der Seite hinzugefügt werden muß.

```
<meta http-equiv="refresh" content="5;URL=http://www.asp.net">
```

Verwendet man allerdings eine MasterPage muß dieser Html-Tag dynamisch hinzugefügt werden, ansonsten wäre er auf allen Seiten aktiv. Hierfür habe ich eine kleine Methode geschrieben.

```
private void AddMetaReferesh(int time, string url)
{
    HtmlGenericControl metaRefresh = new
    HtmlGenericControl("meta");
    metaRefresh.Attributes.Add("http-equiv", "refresh");
    metaRefresh.Attributes.Add("content", time + ";URL=" + url);
    Master.Page.Header.Controls.Add(metaRefresh);
}
```

Nachdem eine Instanz der `HtmlGenericControl` Klasse erstellt und die verschiedenen Attribute zugewiesen wurden, wird das Control dem Header der MasterPage hinzugefügt. Das Ergebnis ist ein dynamisch hinzugefügter Meta-Refresh.

Den Download des Beispielprojekts gibt es [hier](#)

2.4.19. [Server.MapPath in Session_End \(\)](#)

Möchte man die Methode `Server.MapPath` im `Session_End` Event der `global.asax` Datei verwenden, erhält man eine *"Object reference not set to an instance of an object"* Exception. Was zunächst ein wenig verwirrend ist, hat einen ganz einfachen

Hintergrund. Die Klasse `HttpContext`, welche eine Instanz der Klasse `HttpServerUtility` und somit die Methode `MapPath` zurück gibt, ist null. Der Grund hierfür ist, dass der `HttpContext` nur Request-bezogen zur Verfügung gestellt wird. Zum Zeitpunkt des Aufrufs von `Session_End`, ist die Session des Users (wie der Name schon sagt) bereits beendet und somit findet auch kein Request mehr statt.

Um die Funktionalität von `Server.MapPath` trotzdem abbilden zu können, kann man das Property `AppDomainAppPath` der `HttpRuntime` Klasse zur Hilfe nehmen.

```
private string MyMapPath(string file)
{
    return Path.Combine(HttpRuntime.AppDomainAppPath, file);
}
```

In Verbindung mit der `Path.Combine`, liefert diese Methode nun den kompletten Pfad zurück.

2.4.20. [Dynamische Bilder per Generic Handler \(*.ashx\) anzeigen](#)

Um dynamische Bilder in ASP.NET Seiten anzuzeigen verwendet man am besten einen Generic Handler (*.ashx), anstatt eines Web Forms und die damit verbundenen [Probleme](#). Diesen Tipp gab mir [Thomas](#) per Kommentar, beziehend auf das Posting [Formulare gegen SPAM schützen](#) und [Using themed css files requires a header control on the page](#)

Grundsätzlich ist ein Web Form von der Klasse `System.Web.UI.Page` abgeleitet und ist somit dafür gedacht HTML, Webcontrols und Ähnliches anzuzeigen. Da die Anforderung aber einfach nur die Ausgabe eines Bildes per Stream ist, sind diese Dinge völlig unnötig. In diesem Moment kommt der Generic Handler ins Spiel.

Ein Generic Handler implementiert das `IHttpHandler` Interface und erlaubt es somit einen HTTP Handler zu schreiben, ohne ihn vorher kompilieren zu müssen. HTTP Handler werden normalerweise benutzt und auf eine vorher definierte Dateiendung reagieren zu können. Definiert man z.B. die Dateiendung `rss` in der `Web.Config`, leitet man die Verarbeitung an eine entsprechend Klasse, die das `IHttpHandler` Interface implementiert weiter. Diese Klasse übernimmt nun die Ausgabe des XML, welches man hinter der Endung `rss` erwarten würde.

In diesem Beispiel ist es aber lediglich gewollt ein Bild per Stream auszugeben. Fügt man eine `ashx` Datei dem Projekt hinzu, findet man eine Klasse mit der Methode `ProcessRequest` und das Property `IsReusable` vor.

```
using System;
using System.Web;

public class CImage : IHttpHandler {

    public void ProcessRequest (HttpContext context) {
        context.Response.ContentType = "text/plain";
        context.Response.Write("Hello World");
    }
}
```

```
public bool IsReusable {  
    get {  
        return false;  
    }  
}
```

Um den Zugriff auf den `SessionState` und somit den gespeicherten Text in der Session zu ermöglichen, muss die `Generic Handler Klasse` zunächst um das Interface `IRequiresSessionState` erweitert werden.

```
public class CIIImage : IHttpHandler, IRequiresSessionState
```

Anschließend muß der Code zum Erstellen des Bildes in die Methode `ProcessRequest` kopiert und der `ContentType` entsprechend der Ausgabe verändert werden.

```
public void ProcessRequest (HttpContext context)  
{  
    context.Response.ContentType = "image/jpg";  
    CaptchaImage.CaptchaImage ci = new  
    CaptchaImage.CaptchaImage(context.Session["CaptchaImageText"].ToS  
    tring(), 200, 50, "Century Schoolbook");  
    context.Response.Clear();  
    context.Response.ContentType = "image/jpeg";  
    ci.Image.Save(context.Response.OutputStream,  
    System.Drawing.Imaging.ImageFormat.Jpeg);  
    ci.Dispose();  
}
```

Die Variable `context` vom Typ `HttpContext` ermöglicht die Ausgabe.

Ein Aufruf der Seite `CIIImage.ashx` zeigt nun das Bild an, und birgt keine unerwarteten [Probleme](#) in sich. Den Download des erweiterten Beispielprojekts findet man [hier](#).

2.4.21. [Using themed css files requires a header control on the page](#)

Mit dieser Fehlermeldung wird die Verarbeitung der ASP.NET Seite quitiert, sobald man das von [mir beschriebene Captcha Control](#) in eine Seite implementiert, die Themes verwendet.

ASP.NET erwartet den Head-Tag in Form eines Server-Controls, damit die entsprechen Theme Informationen eingebunden werden können:

```
<head runat="server">
```

Allerdings gibt es ASPX-Seiten in denen dieser Tag nicht vorhanden ist. In oben genanntem Beispiel übernimmt die Seite `CIIImage.aspx` das streaming des Captcha Images und somit würde dieser Tag zu Fehlern führen.

Um dieses Problem zu lösen hatte ich zunächst das Attribut `EnableTheming` im Page Header auf `false` gesetzt. Diese Einstellung bewirkt, dass die Seite während der Verarbeitung der Themes nicht berücksichtigt wird. Allerdings stört sich der ASP.NET

Page Parser noch immer am fehlenden Tag und beendet den Aufruf mit derselben Fehlermeldung. Es führt also kein Weg am Tag vorbei.

Mit einem kleinen Trick ist es aber dennoch möglich den Page Parser zufrieden zustellen und eine korrekte Ausgabe zu gewährleisten:

```
<head runat="server" visible="false"/>
```

Das Attribut `visible` sorgt dafür das die Ausgabe ohne Tag erfolgt, während der Page Parser dieses vorfindet und somit die Verarbeitung nicht unterbricht.

.

2.4.22. [Formulare gegen SPAM schützen](#)

Um Formulare gegen SPAM-Einträge zu schützen bietet sich ein sog. Captcha an. Die Abkürzung Captcha steht für "*completely automated public Turing test to tell computers and humans apart*". Hinter diesem Wort verbirgt sich eine Technik, die mit Hilfe eines Bildes alphanumerische Zeichen darstellt.

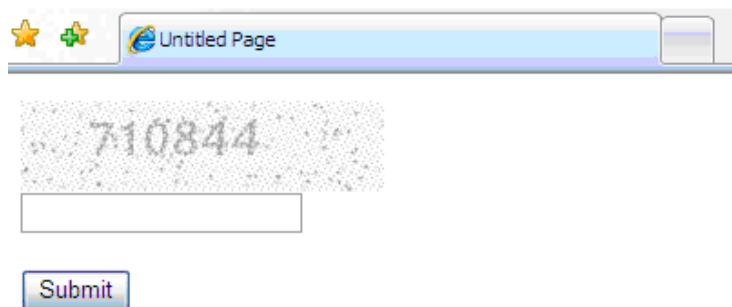


Abbildung 22: Captcha erstellen

Robots, die nun versuchen das Formular per Automatismus auszufüllen scheitern daran, die Zeichen abzulesen und in ein dafür vorgesehenes Formular-Feld einzutragen.

Neben div. kostenpflichtigen Controls findet man auf [CodeProject.com](#) eine fertige Klasse inkl. Beschreibung zur freien Verwendung. Die Implementierung ist nicht aufwendig und folgt einem einfachen Schema:

*Generierte Zeichen in Session speichern
Captcha Image erstellen und anzeigen
Eingegebene Zeichen mit dem in der Session gespeicherten Wert vergleichen.*

Zunächst muß die Klasse in den `App_Code` Ordner des Webs kopiert und anschließend ein WebForm mit dem Namen `CIImage.aspx` hinzugefügt werden. Diese WebForm übernimmt das Streaming des Bildes, weshalb auch der gesamte Html-Code entfernt werden muß. In der `Page_Load()` Methode wird das Bild erstellt und an den `OutputStream` des `Response`-Objektes gesendet.

```
protected void Page_Load(object sender, EventArgs e)  
{
```

```
CaptchaImage ci = new  
CaptchaImage(Session["CaptchaImageText"].ToString(), 200, 50,  
"Century Schoolbook");  
Response.Clear();  
Response.ContentType = "image/jpeg";  
ci.Image.Save(this.Response.OutputStream, ImageFormat.Jpeg);  
ci.Dispose();  
}
```

Die Seite Default.aspx übernimmt die Anzeige des Captcha-Images und auch die Generierung der angezeigten Zeichen. Ein einfacher HtmlImage-Tag wird verwendet um das Bild anzuzeigen. Die Überprüfung übernimmt ein CustomValidator, der die eingegebenen Zeichen mit denen der Session vergleicht.

```
<br />  
<asp:TextBox ID="txtCaptchaText" runat="Server" /><br />  
<asp:CustomValidator ID="valCaptcha" runat="server"  
ErrorMessage="Bitte geben Sie den korrekten Code ein."  
Display="dynamic" OnServerValidate="CheckCaptcha"/><br />  
<asp:Button ID="cmdGo" runat="server" OnClick="cmdGo_Click"  
Text="Submit" CausesValidation="true"/>
```

Die Zeichen werden mit Hilfe der Random-Klasse erstellt und anschließend in der Session gespeichert:

```
protected void Page_Load(object sender, EventArgs e)  
{  
    if(!IsPostBack)  
        Session["CaptchaImageText"] = GenerateRandomCode();  
}  
  
private string GenerateRandomCode()  
{  
    Random random = new Random();  
  
    string captchaCode = "";  
    for (int i = 0; i < 6; i++)  
        captchaCode = String.Concat(captchaCode,  
random.Next(10).ToString());  
    return captchaCode;  
}
```

Bei Formularen deren Daten sofort nach dem Submit sichtbar sind, wie z.B. für Gästebücher oder Kommentare für Blog-Einträge, macht diese Absicherung durchaus sinn. Auch die hier gezeigte, einfache Implementierung spricht dafür. Spätestens jedoch nachdem ein Robot mehrere Einträge verfasst hat, wird sie sogar absolut erforderlich.

Das Beispielprojekt findet man [hier](#). Das Sourceforge-Beispiel inkl. Captcha-Control Klasse [hier](#).

Wird dieses Beispiel in einer Anwendung mit Themes verwendet muss [dies](#) beachtet werden.

Wer oft mit User Controls arbeitet hat sich vielleicht schon darüber beschwert, die User Controls auf jeder Seite registrieren zu müssen. ASP.NET 2.0 bietet allerdings die Möglichkeit dies zentral in der `Web.config` zu erledigen.

Dazu muss die `Web.Config` im Node `system.web` angepasst und folgender Code hinzugefügt werden:

```
<pages>
  <controls>
    <add tagPrefix="UC" tagName="World2"
src="~/UC/MyUserControl1.ascx"/>
  </controls>
</pages>
```

Dieser sorgt dafür, daß das User Controls `MyUserControl1` nun für das gesamte Web registriert ist - Natürlich mit voller IntelliSense Unterstützung im Visual Studio.

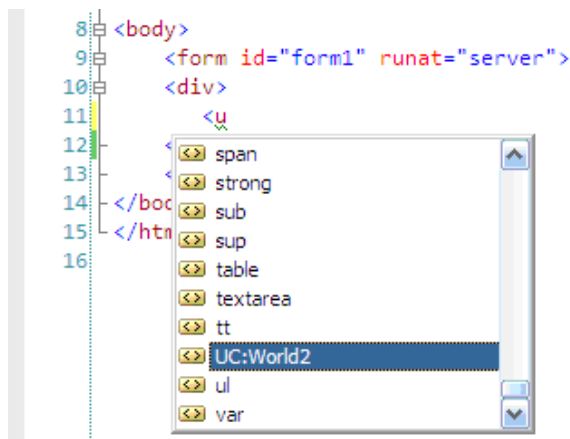


Abbildung 23: Caching von Bildern verhindern

Ein kleines Beispielprojekt gibt es [hier](#).

-
.

2.4.24. [Caching von Bildern verhindern](#)

In div. Foren stellen User häufiger die Frage, wie man das Caching von Bildern verhindern kann. Ein einfacher aber wirkungsvoller Trick ist es, an den Bildnamen eine Zufallszahl zu hängen. Meist reicht die Verwendung des Datums und der Uhrzeit:

```
image.ImageUrl = "Image/image.gif?" +  
System.DateTime.Now.ToString("yyyMMddHHmmss");
```

Der Browser findet nun bei jedem Aufruf ein neuer Bildnamen vor und cached somit das Bild nicht.

-

2.4.25. [XmlDataSource, GridView u. DataFormatString](#)

Die Darstellung einer XML - Datei ist mit Hilfe der `XmlDataSource` und des `GridView` kein Problem. Zumindest solange man nicht die Ausgabe der einzelnen Felder formatieren

möchte. Hierbei stieß ich auf ein Problem, welches ich nur mit einem kleinen Workaround lösen konnte.

Zunächst der Aufbau der XML-Datei:

```
<Data> <Item> <Number>0001</Number> <Name>Test
1</Name> <Price>20.25</Price> <Date>12/11/1999</Date>
</Item> <Item> <Number>0002</Number> <Name>Test
2</Name> <Price>1.75</Price> <Date>12/11/1999</Date>
</Item> </Data>
```

Mit Hilfe des XmlDataSourceControls werden die Daten an das GridView gebunden.

```
<asp:GridView ID="grid" runat="server"
DataSourceID="XmlDataSource1"

    AutoGenerateColumns="False"> <Columns> <asp:BoundField
DataField="Number" HeaderText="Number"

    SortExpression="Number" HtmlEncode="false"/>
<asp:BoundField DataField="Name" HeaderText="Name"

    SortExpression="Name" HtmlEncode="false"/> <asp:BoundField
DataField="Price" HeaderText="Price"

    SortExpression="Price" HtmlEncode="false"
DataFormatString="{0:C2}" /> <asp:BoundField DataField="Date"
HeaderText="Date"

    SortExpression="Date" HtmlEncode="false"
DataFormatString="{0:dd.MM.yyyy}" /> </Columns> </asp:GridView>
<asp:XmlDataSource ID="XmlDataSource1" runat="server"
DataFile="~/App_Data/Data.xml"
TransformFile="~/App_Data/Data.xsl"
EnableCaching="False"></asp:XmlDataSource>
```

Wie man sehen kann wird bei den Feldern Price u. Date ein DataFormatString angewendet. Außerdem wird das Attribut HtmlEncode auf false gesetzt, um den bekannten Bug des GridView zu umgehen.

Allerdings zeigt der DataFormatString dennoch keine Wirkung, so daß die Ausgabe weiterhin unformatiert im Browser erscheint.

Number	Name	Price	Date
0001	Test 1	20.25	12/11/1999
0002	Test 2	1.75	12/11/1999
0003	Test 3	10.5	12/11/1999
0004	Test 4	15.5	12/11/1999
0005	Test 5	25	12/11/1999

Abbildung 24: GridView und XmlDataSource

Ein möglicher Workaround für diese Gegebenheit wäre die Verwendung des `TemplateField` und das manuelle Casten und Formatieren der Daten.

```
<asp:TemplateField> <ItemTemplate> <%#  
    Convert.ToDateTime(Eval("Date")).ToShortDateString() %>  
</ItemTemplate> </asp:TemplateField>
```

Die Ausgabe erscheint nun formatiert, allerdings ohne die Möglichkeit die Vorzüge der `BoundFields` zu nutzen. .

2.4.26. [Controls dynamisch hinzufügen](#)

Immer mal wieder taucht in Foren die Frage auf, wie man Controls (wie z.B. die `TextBox`) dynamisch zu einer Seite hinzufügen kann. Mit Hilfe des [PlaceHolder](#) Controls ist dies ganz einfach möglich.

In diesem Beispiel soll der User die Anzahl der `TextBox` bestimmen können, die zur Seite hinzugefügt werden.

Zunächst muß das `PlaceHolder` Control, eine `TextBox` und ein Button auf der Seite platziert werden. In die `TextBox` trägt der User die Anzahl der `TextBox` ein, die hinzugefügt werden sollen. Der Event `OnClick` ruft die Methode `cmdAddControls_Click()`, liest die Anzahl aus und übergibt diese anschließend an die Methode `AddControls()`.

```
protected void cmdAddControls_Click(object sender,  
    EventArgs e) { int numToAdd = Convert.ToInt32(txtAddNum.Text);  
    AddControls(numToAdd); }
```

Die Methode `AddControls()` ist einfach aufgebaut. Innerhalb einer `For`-Schleife wird eine Instanz der `TextBox`-Klasse angelegt und diese dann der `Control-Collection` des `PlaceHolders` hinzugefügt.

```
private void AddControls(int number)  
    { for (int i = 0; i < number;  
        i++) { TextBox textbox = new TextBox(); textbox.Text =  
        String.Format("Textbox  
        {0}", i); plcControls.Controls.Add(textbox);  
        plcControls.Controls.Add(new LiteralControl("<br
```

```
    /><br />")); } }
```

Das Ergebnis dieser paar Zeilen sind dynamisch hinzugefügte TextBox-Controls. Allerdings müssen diese Controls nach jedemPostBack erneut hinzugefügt werden. Der Inhalt jeder TextBox wird allerdings vom ViewState gespeichert und bleibt somit erhalten.

Ein Tip am Rand: Das Control [DynamicControlsPlaceholder](#) von [Denis Bauer](#) speichert die hinzugefügten Controls auch über PostBacks hinweg.

Das Beispielprojekt gibt es [hier](#).

2.4.27. [DropDownList - Änderung der Auswahl durch JavaScript-Dialog bestätigen](#)

Heute tauchte in einem Forum die Frage auf, ob es möglich ist die Änderung der Auswahl einer DropDownList nochmals durch den User per Dialog bestätigen zu lassen oder einen Hinweis auszugeben. Natürlich ist das möglich.

Sobald sich die Auswahl und somit auch der Index der DropDownList ändert, wird der clientseitige Event onChange ausgeführt. Die Lösung besteht also darin diesen Event mit einer entsprechenden JavaScript-Methode zu belegen. Hierzu habe ich mir folgenden Code geschrieben:

```
private void RegisterDDLScript()
{
    string script =
        "if(!confirm('Sicher?')){document.getElementById('" +
        ddlList.ClientID+ "') .selectedIndex = currentIndex;return
        false;}";
    ddlList.Attributes.Add("onchange", script);
}
```

Da der Event ausgeführt wird nachdem sich die Auswahl bereits geändert hat, muß man den vorherigen Index speichern. Das entsprechende Script hierfür registriert, in meinem Fall die Methode RegisterStartupScript():

```
private void RegisterStartupScript()
{
    string script = "var currentIndex =
    document.getElementById('" + ddlList.ClientID +
    "') .selectedIndex;";
    ClientScript.RegisterStartupScript(this.GetType(),
    "ddlSelectedIndex",
    script, true);
}
```

Das entsprechende Beispielprojekt findet man [hier](#).

EDIT: Auf vielfachen Wunsch habe ich auch eine .NET Framework 1.1 Version des Beispielprojekts erstellt. [Download](#).

2.4.28. [Cache löschen](#)

Folgende kleine Methode habe ich heute geschrieben. Ihre Aufgabe ist es alle vorhandenen Cache-Items zu löschen.

```
private void DeleteAllCacheItems()
{
    HttpContext ctx = HttpContext.Current;

    IDictionaryEnumerator d = ctx.Cache.GetEnumerator();

    while(d.MoveNext())
    {
        ctx.Cache.Remove(d.Key.ToString());
    }
}
```

Brav wie ich bin, habe ich diese natürlich auch auf snippetcenter.org veröffentlicht.

2.4.29. [Kopieren einer ASP.NET 1.1 Anwendung und VS 2003](#)

Visual Studio .NET hat ermittelt, dass auf dem angegebenen Webserver nicht die ASP.NET, Version 1.1, ausgeführt wird. ASP.NET -Webanwendungen und -dienste können daher nicht ausgeführt werden.

Diese Fehlermeldung bekam ich heute angezeigt, nachdem eine Anwendung in einen anderen Ordner kopiert wurde. Diese verwirrende Fehlermeldung hat eine ganz einfache Ursache. Nachdem die Anwendung in einen anderen Ordner bzw. eine andere URL kopiert wurde, muss die Datei <Projektname>.csproj.webinfo angepasst werden.

Diese Datei wird geschrieben, sobald eine Anwendung erstellt wird:

```
<VisualStudioUNCWeb>
  <Web URLPath = "http://www.url.de/Anwendung.csproj" />
</VisualStudioUNCWeb>
```

Nachdem der Pfad angepasst wurde, kann man die Anwendung ohne Probleme öffnen. Verwendet man die Visual Studio Funktion "Projekt kopieren", muss man sich keine Gedanken um die Anpassung dieser Datei machen.

2.4.30. [Controls anhand der ID rekursiv suchen](#)

Schon vor längerer Zeit habe ich mir eine kleine Methode geschrieben, um Controls auf einer ASP.NET Seite per ID zu finden. Vielleicht hat der eine oder andere ja Verwendung dafür.

```
public static Control FindControlRecursive(Control root, string
id) {
    if (root.ID == id) {
        return root;
    }

    foreach (Control c in root.Controls) {
        Control t = FindControlRecursive(c, id);
        if (t != null) {
            return t;
        }
    }
}
```

```
    }  
    }  
    return null;  
}
```

2.4.31. [AnkhSVN, TortoiseSVN und ASP.NET](#)

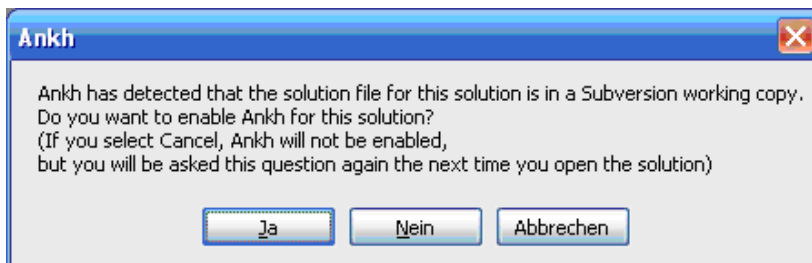
Verwendet man die [TortoiseSVN ASP.NET Version](#), wird statt dem Standard Admin Ordner “.svn” der Ordner “_svn” erstellt. Visual Studio 2003 hat ein Problem mit Ordnern deren Namen mit einem Punkt beginnen, weshalb dieser Workaround gewählt wurde.

Nun kann es vorkommen, dass [AnkhSVN](#) bei Starten eines Projekts den Ordner “.svn” sucht, nicht findet und somit die Unterstützung für das Projekt deaktiviert. Damit der nach dem korrekten Ordner gesucht wird, reicht eine einfache Einstellung in der Configdatei von AnkhSVN. Die Datei findet man im Ordner:

```
C:\Dokumente und  
Einstellungen\<<Benutzer>\Anwendungsdaten\AnkhSVN\ankhsvn.xml
```

In der Datei ersetzt man den Inhalt des Nodes AdminDirectoryName mit dem Wert “_svn”. Per default war bei mir der Node noch auskommentiert. Nachdem die Änderungen durchgeführt wurden, reicht es wenn die Datei gespeichert und das Projekt geladen wird.

Hat alles geklappt, meldet sich Visual Studio bzw. AnkhSVN beim Starten mit folgendem Fenster:



2.4.32. [Reservierte ASP.NET Projektnamen](#)

Gibt man ein ASP.NET 1.1 Projekt z.B. den Namen "ValidationSummary" wird der Aufruf mit einer Fehlermeldung quittiert:

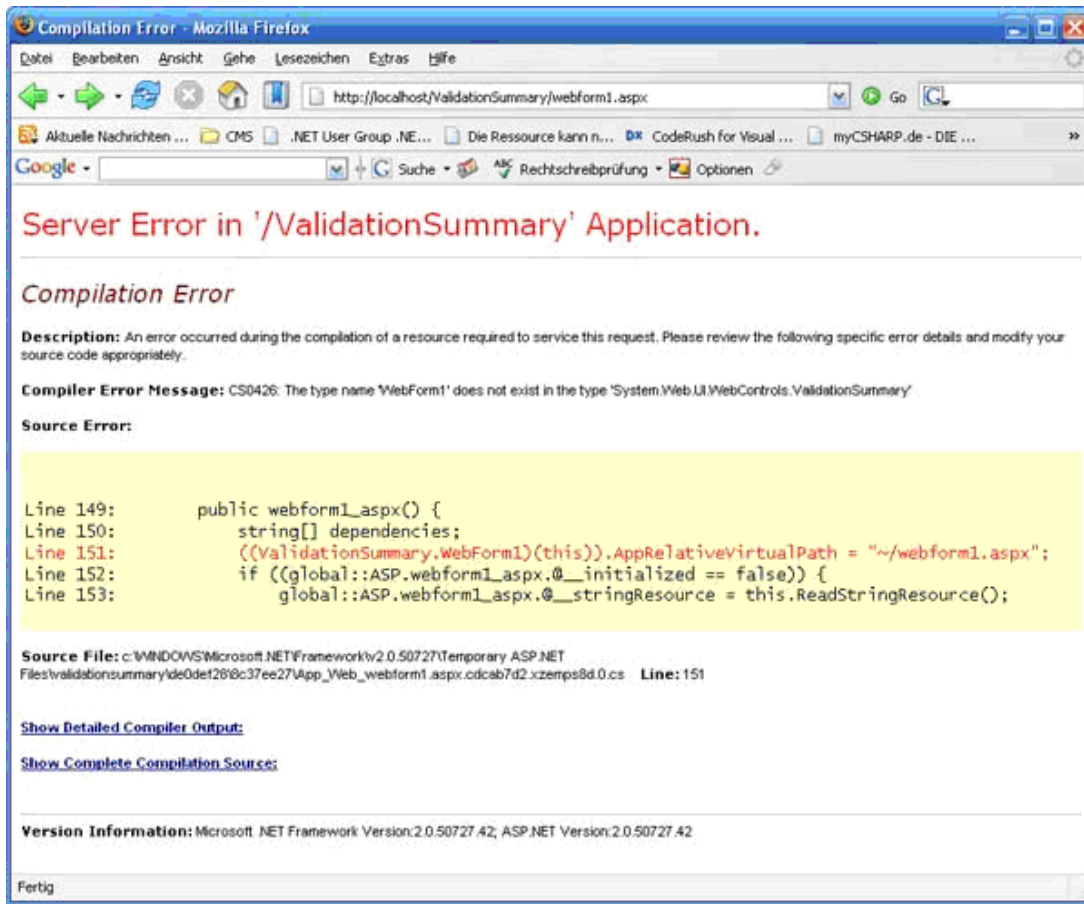


Abbildung 25: Reservierte ASP.NET Projektnamen

Der Grund für die Fehlermeldung ist klar, nur hätte Visual Studio die Namen der Projekte bzw. dlls vielleicht schon beim Anlegen auf reservierte Wörter überprüfen sollen. Ich denke dabei an unbedarfte User, die Ihr erstes Projekt vielleicht "TextBox" nennen, grade weil Sie mit diesem Control beginnen.

-
.
.

2.4.33. [SiteMap menu with icons](#)

Möchte man Icos in der Sitemap verwenden, sollte man sich folgende Lösung zu Gemüte führen:

<http://weblogs.asp.net/bleroy/archive/2006/01/24/436350.aspx>

.

2.4.34. [Debug and Release Builds in ASP.NET 2.0](#)

Wer sind nach ersten Experimenten mit ASP.NET 2.0 gefragt hat, wohin das /bin Folder und die Project dll verschwunden sind, sollte einen Blick auf den Artikel "Debug and Release Builds in ASP.NET 2.0" von K. Scott Allen unter folgender Adresse werfen:

<http://odetocode.com/Blogs/scott/archive/2005/11/15/2464.aspx>

.

2.4.35. [Maximum request length exceeded](#)

Die Meldung wird ausgegeben, sobald man versucht eine Datei, die größer als 4096 KB ist per `HtmlInputFile` Control hoch zu laden. Um dieses Limit zu erhöhen überschreibt (bzw. ergänzt) man folgenden Wert in der `Web.Config` oder `Maschine.Config`:

```
<httpRuntime maxRequestLength="8192" />
```

`maxRequestLength` steht natürlich für den Maximalwert in KB.

Die Knowledge Base Artikel findet man dazu unter:

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;295626>.

.

2.4.36. [Session lost, nachdem ein Verzeichnis umbenannt wurde](#)

Anscheinend verliert ASP.Net nachdem ein Verzeichnis umbenannt wurde die komplette Session. Diesen Fehler kann ich jederzeit reproduzieren und wurde mir auch von Olaf Lüder in `microsoft.public.de.german.entwickler.dotnet.asp` bestätigt. Dieser Fehler tritt auch auf, wenn leere Verzeichnisse umbenannt werden.

Folgender Workaround:

1. Neues Verzeichnis anlegen
2. Dateien kopieren
3. Altes Verzeichnis löschen.

Hierzu habe ich die Funktion `CopyFiles()` geschrieben:

```
public static void CopyFiles(string strSourceFolder, string  
strDestinationFolder, string strRestrictions) {  
    //Kopiert alles Dateien aus einem Verzeichnis in das andere.  
    //Es können auch Einschränkungen (z.B. *.aspx) übergeben  
    werden.  
  
    if(  
strDestinationFolder.Substring(strDestinationFolder.Length-1,1)  
!= "\\") strDestinationFolder += "\\";  
  
    DirectoryInfo dir = new DirectoryInfo(strSourceFolder);  
    FileInfo[] files = dir.GetFiles(strRestrictions);  
    foreach(FileInfo file in files) {  
        file.CopyTo(strDestinationFolder + file.Name,true);  
    }  
}
```

.

2.4.37. [ASP.NET: HttpWebRequest und Timeout-Problem](#)

Bis dato habe ich ja noch nicht viel mit ASP.NET gemacht und muss daher auf die eine oder andere Tücke erst kommen. Da ich mich beruflich nun verstärkt damit beschäftigen muss, wird wohl meine ASP.NET Kategorie ein wenig aufgefüllt werden.

Nun jedoch zum eigentlichen Thema: Wer via ASP.NET WebRequests aufbaut kann eventuell auf das Problem eines Timeouts stoßen. Meist beim zweiten Aufruf erfolgt ein Timeout. Des Rätsels Lösung liegt darin, die dazugehörigen Response-Objekte zu schließen. Vergisst man ein `Response.Close()` kommt es eben zum genannten Timeout-Problem. Also immer brav darauf Acht geben :)

2.4.38. [Probleme mit WebUIValidation.js](#)

Wer nach dem Deployment einer ASP.NET Anwendung oder bei der Entwicklung ein Problem mit der Datei `WebUIValidation.js` hat (es erscheint eine wunderschöne Fehlermeldung, dass diese Datei nicht vorhanden ist oder aber nur nicht verwendet werden kann), dem kann einfach geholfen werden.

Dazu einfach die Visual Studio 2005 Eingabeaufforderung öffnen und den Befehl

```
aspnet_regiis -c
```

eingeben. Ab sofort sollte wieder alles ganz normal funktionieren.

2.5. Services

2.5.1. [Windows Dienste mit C# und .NET 2.0 kontrollieren](#)

Hier eine kleine Demoklasse die den Umgang mit Windows-Diensten zeigt. Die Klasse selbst bietet nur die Möglichkeit den Status eines Dienstes abzufragen und diesen zu Starten bzw. zu Stoppen. Weitere Möglichkeiten kann den Klassen `ServiceController` entnommen werden.

```
public class ProcessHandler
{
    private string processName = null;

    public string ProcessName
    {
        get { return this.processName; }
        set { this.processName = value; }
    }

    public ServiceControllerStatus GetProcessState()
    {
```



```
        ServiceController sc = new
ServiceController(processName);
        if (sc != null)
        {
            return sc.Status;
        }
        return ServiceControllerStatus.Stopped;
    }

    public void StartProcess()
    {
        ServiceController sc = new
ServiceController(processName);
        if (sc != null)
            sc.Start();
    }

    public void StopProcess()
    {
        ServiceController sc = new
ServiceController(processName);
        if (sc != null && sc.CanStop)
            sc.Stop();
    }
}
```

2.6. Windows Presentation Foundation

2.6.1. [Windows Presentation Foundation - Teil 2: XAML und Layouts](#)

Im zweiten Teil der Tutorials-Reihe über die Windows Presentation Foundation werden die Themen XAML und Layouts behandelt. So wird beschrieben, was XAML genau ist und wofür es da ist. Weiters wird gezeigt, welche grundlegenden Layout-Elemente zur Verfügung stehen. Natürlich ist auch dieser Teil wieder voll von Beispiel-Code und Screenshots zur Veranschaulichung.

[Windows Presentation Foundation - Teil 2: XAML und Layouts \(PDF\)](#)

2.6.2. [WPF: Rotation und Skalierung einfach gemacht](#)

Beschäftigt man sich näher mit der Materie Windows Presentation Foundation, sieht man schon an sehr einfachen Beispielen, dass die Möglichkeiten schon sehr mächtig sind. Beispielhaft zeige ich an dieser Stelle, wie einfach ein Button skaliert und gedreht werden kann.

Dazu wird im Beispiel ein simpler Button erstellt und plaziert. Über Schieberegeler ist es möglich, die Values für die Rotation bzw. des Zoomfaktors zu setzen. Dies sieht dann folgendermaßen aus:

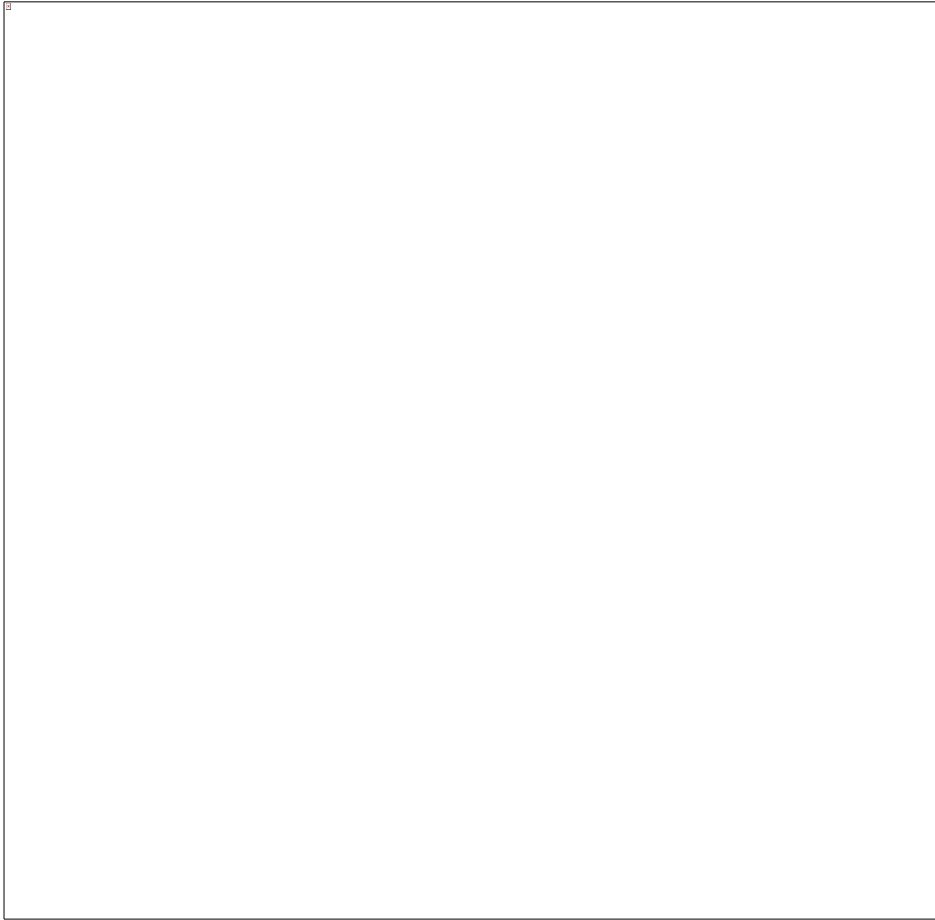


Abbildung 26: WPF Rotation 1

Um nun die Funktionalität zu implementieren ist nichts weiter zu machen, als die entsprechenden EventHandlerer zu setzen:

Zoomfaktor

```
public void sliderZoom_ValueChanged(object sender,
RoutedEventArgs e)
{
    ScaleTransform st = new ScaleTransform(sliderZoom.Value,
sliderZoom.Value);
    this.btnTest.LayoutTransform = st;
}
```

Rotationsfaktor

```
public void sliderRotate_ValueChanged(object sender,
RoutedEventArgs e)
{
    RotateTransform rt = new RotateTransform(sliderRotate.Value);
    this.btnTest.RenderTransform = rt;
}
```

Das Ergebnis sieht dann wie folgt aus:



Abbildung 27: WPF Rotation 2

Dies kann jetzt nicht nur auf einen Button angewandt werden, sondern auf die gesamte Oberfläche, oder auch nur einzelne Bereiche. Der dahinterliegende XAML-Code etc. ist im gesamten Projekt enthalten, welches zum Download bereit steht.

[Download WPF Rotation Test Beispiel](#)

2.6.3. [Windows Presentation Foundation - Teil 1: Einführung](#)

Der erste Teil einer Tutorials-Reihe gibt eine kleine Einführung in das Thema Windows Presentation Foundation (ehemals Avalon) und beschreibt die wichtigsten Punkte. Zudem wird anhand einer sehr simplen Testanwendung gezeigt, wie eine solche entwickelt werden kann.

Der nächste Teil dieser Serie wird weiter in die Tiefe gehen und Hintergründe näher erläutern.

[Windows Presentation Foundation - Teil 1: Einführung \(PDF\)](#)

2.7. Sonstiges

2.7.1. [Im Batch den Ausführungserfolg einer aufgerufenen \(selbst erstellten\) Anwendung prüfen?](#)

Folgender Sachverhalt:

Eigene Anwendungen (vorzugsweise Consolen-Anwendungen) werden im Batch ausgeführt (beispielsweise via NAnt, MSBuild etc.). Im Batch soll auf Abbruchbedingungen geprüft und entsprechend reagiert werden.

Wie kann dies einfach mit .NET 2.0 gelöst werden? Eigentlich ganz einfach. Wieder einmal hilft der `System.Environment`-Namespace. Mit Hilfe der Eigenschaft `ExitCode` besteht die Möglichkeit einen solchen zu setzen. 0 (Null) ist standardmäßig gesetzt und bedeutet, dass die Anwendung erfolgreich ausgeführt wurde. Nun könnte es aber sein, dass es Rückmeldungen nicht nur für eine erfolgreiche Durchführung geben soll, sondern auch wenn Fehler oder Warnungen gemeldet werden sollen. Hierzu ist lediglich ein `Integer`-Wert zu definieren und `ExitCode` entsprechend zu setzen. Damit steht der Behandlung von Abbruchbedingungen im Batch nichts mehr im Weg.

2.7.2. [Abhängigkeiten \(Referenzen\) führen zu Kompilations-Problemen](#)

Wer kennt das nicht bei größeren Aufträgen bzw. Produkten: Mehrere Projekte tummeln sich in einer Solution und referenzieren einander. Oder noch schlimmer, die einzelnen Projekte sind in unterschiedlichen Solutions eingebunden. Und eventuell sind diese auch gleichzeitig geöffnet. Dann kann es zu folgender Fehlermeldung kommen:

```
Error: The dependency [name], Version=2.0.0000.0,
Culture=neutral,
PublicKeyToken=[etc]' in project [name] cannot be copied to
the run directory because it would conflict with dependency
[name],
Version=0.0.0000.0, Culture=neutral, PublicKeyToken=[etc]'.

```

In diesem Fall einfach das Visual Studio beenden und alles aus dem bin-Ordner des Projektes löschen. Danach das Projekt im VS laden und neu kompilieren. Nun sollte ansich wieder alles in Ordnung sein.

PS: Natürlich auch die einzelnen Referenzen überprüfen, ob es hier nicht notwendige Einträge in diversen Projekten der Solution gibt.

3. VISUAL STUDIO

3.1. [Auflistung Visual Studio Shortcuts](#)

[Kai](#) hat sich die Mühe gemacht, die wichtigsten Visual Studio Shortcuts in einem [PDF](#) zusammen zu fassen. Diese Liste ist sicherlich nicht vollständig, wird aber sehr wahrscheinlich um neue Shortcuts erweitert werden. Wer also zukünftig vermehrt auf seine Maus verzichten möchte, der sollte einen Blick darauf werfen.

3.2. [Visual Studio: Bookmarks aller geöffneten Dokumente ohne Rückfrage entfernen](#)

Wer viel mit Bookmarks arbeitet der kommt auch des öfteren in die Lage, alle gesetzten Bookmarks entfernen zu wollen. Hierfür gibt es das Tastenkürzel CTRL + B + C. Allerdings erscheint hier die Nachfrage, ob denn wohl wirklich alle entfernt werden sollen. Wer dies nicht möchte, kann sich folgendes Makro einbauen und dann einem Button/Shortcut zuweisen - und das ohne diese Nachfrage. Dies funktioniert für alle geöffneten Dokumente:

```
Public Sub RemoveAllBookmarks()  
    Dim i As Integer  
  
    For i = 1 To DTE.Documents.Count  
        Dim doc As EnvDTE.TextDocument  
        doc = DTE.Documents.Item(i).Object  
        doc.ClearBookmarks()  
    Next  
End Sub
```

3.3. [Interessante VS 2005 Tastenkombinationen](#)

Durch Zufall heute auf zwei interessante Tastenkombinationen des Visual Studios 2005 gekommen.

[CTRL] + [']: Korrespondierende/s Klammer/Tag anspringen
[CTRL] + [']: Fokus in die Find-ComboBox setzen

Diese Tastenkombinationen funktionieren übrigens auch unter dem Visual Studio 2003.

3.4. [Visual Studio 2005: Default Browser setzen](#)

Ich hab es schon irgendwann einmal irgendwo gepostet, aber jetzt auf die Schnelle nicht gefunden, daher an dieser Stelle, damit ich es selbst nicht schon wieder vergesse.

Visual Studio 2005 öffnet gerne den Default-Browser für's Debuggen und das muss natürlich nicht immer der Internet Explorer sein (oder man möchte explizit einen anderen Browser wählen). Dazu sind folgende Schritte notwendig.

Debugging stoppen, eine ASPX-Datei des Web-Projektes öffnen. Danach auf File/Browse With ... dadurch erscheint der folgende Dialog:

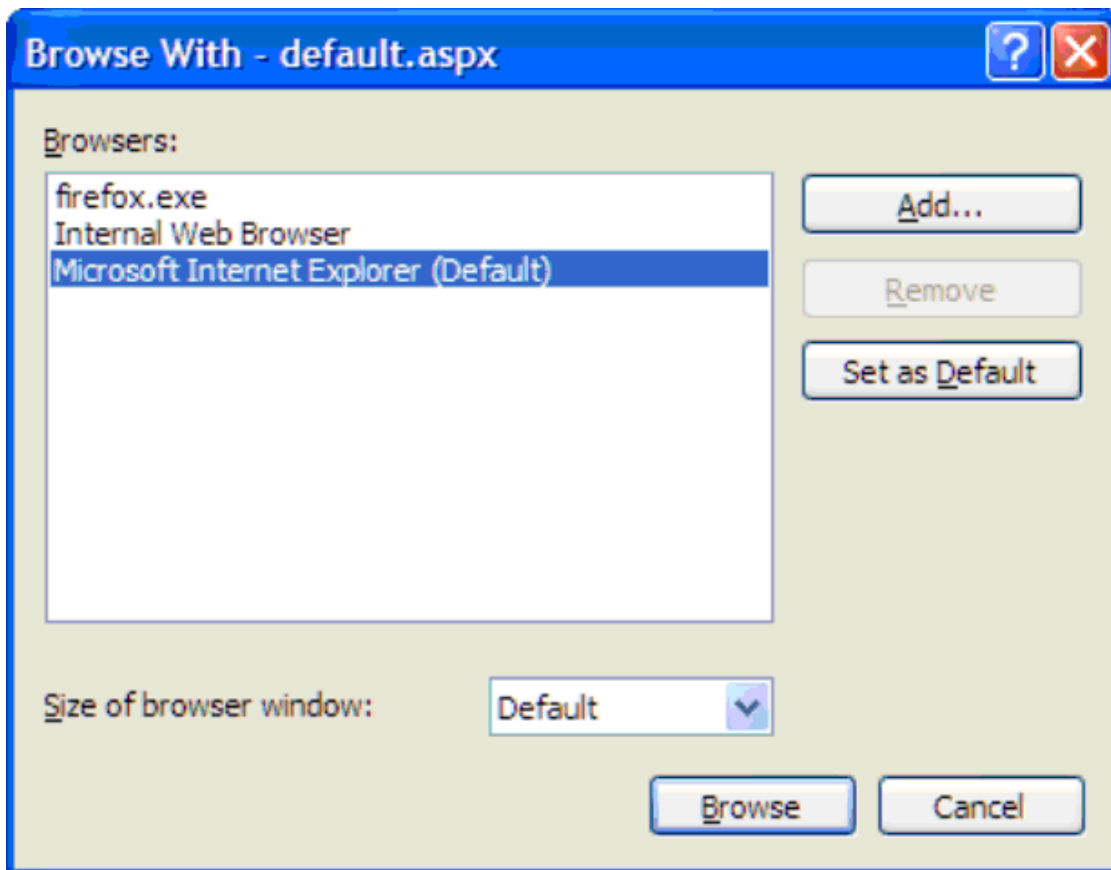


Abbildung 28: Visual Studio Default-Browser

Hier nun den gewünschten Browser auswählen, als Default setzen und auf Browse klicken. Ab sofort wird dieser Browser für die Debugging-Sessions verwendet.

3.5. [Visual Studio: Anpassung Class Template](#)

Ich persönlich unterteile meine Klassen gerne in Regionen nach privaten Feldern, Eigenschaften etc. um hier eine bestmögliche Übersicht zu erhalten. Natürlich ist es anstrengend, dies bei jeder neuen Klasse zu erstellen.

Um ein entsprechendes Ergebnis zu erhalten, kann das Class-Template angepasst werden. Aber gleich hier: Achtung!! Bei Fehlern kann ich nicht garantieren, wie sich das Visual Studio verhält - ich übernehme für etwaige Fehlfunktionen etc. keine Haftung.

So geht es

Benötigt wird die Datei *NewCSharpFile.cs*. Diese befindet sich im Programmverzeichnis des Visual Studios unter VC#\VC#\Wizards\CSharpAddClassWiz\Templates\1033.

Diese ist wie folgt anzupassen:

```
using System;

namespace [!output SAFE_NAMESPACE_NAME]
{
    /// <summary>
    /// Summary description for [!output SAFE_CLASS_NAME].
    /// </summary>
    public class [!output SAFE_CLASS_NAME]
    {
        #region Private Fields

        #endregion

        #region Properties

        #endregion

        #region Private Methods

        #endregion

        #region Public Methods

        #endregion

        #region ctor

        public [!output SAFE_CLASS_NAME] ()
        {
            //
            // TODO: Add constructor logic here
            //
        }

        #endregion
    }
}
```

Abbildung 29: Visual Studio Class Template

Speichern und fertig. Ab sofort wird jede neue Klasse entsprechend dieses Templates erstellt. An dieser Stelle können natürlich auch weitere Informationen (Ersteller, etc.) angegeben werden.

3.6. [Visual Studio 2005: Einfaches Einbinden von Namespaces](#)

Wer sich schon immer darüber geärgert hat, dass Namespaces nicht automatisch eingebunden werden können (á la Eclipse), dem sei die Tastenkombination

<ALT><SHIFT><F10>

nahegelegt. Beispielsweise einfach mal *Hashtable* ins Codefenster schreiben, den Cursor gleich danach positionieren (also keine Leerzeichen eingeben) und die Tastenkombination ausführen. Schon kann man sich den richtigen Namespace aussuchen. Nettes Feature.

3.7. [Visual Studio 2005: Taskbar einblenden](#)

Wem eine Taskbar (ähnlich dem <ALT><TAB>) für Visual Studio 2005 abgeht, dem kann geholfen werden. Einfach mal die nachfolgende Tastenkombination ausprobieren:

<ALT><SHIFT><F7>

3.8. [Visual Studio und Build Events](#)

Oft müssen beim Erstellen eines Releases etc. Daten natráglich in die Ausgabe-Ordner kopiert werden. Dies geht natürlich auch einfacher - automatisiert.

Dazu einfach die Projekteigenschaften öffnen (Menüpunkt Project / {Projectname} Properties).

Im darauf erscheinenden Fenster unter Common Properties / Build Events mit einem der beiden folgenden Einstellungen arbeiten:

Pre-build Event Command Line
Post-build Event Command Line

Wie der Name schon sagt, wird ersteres vor dem Build ausgeführt und zweiteres danach. Darin können nun die gewünschten Aktionen gesetzt werden. Beispielsweise das Kopieren von benötigten Dateien, die nicht ins Projekt eingebunden wurden etc. Einfach einmal auf den Erweiterungsbutton (der mit den 3 Punkten) klicken und im neuen Fenster auf "Macros >>". Darunter finden sich Eigenschaften die des Öfteren benötigt werden. Aufgerufen werden diese Eigenschaften mit \$(Eigenschaftsname).

Im schlimmsten Falle einfach den Help-Button betätigen ;-)

3.9. [Webprojekte mittels Firefox debuggen](#)

Anstelle des Firefox können auch Opera etc. verwendet werden. Um den Browser umzustellen sind folgende Schritte notwendig:

In der geöffneten Web-Solution ein Webfile (beispielsweise eine.aspx) wählen. Danach im Menü folgenden Eintrag wählen:

File / Browse with ...

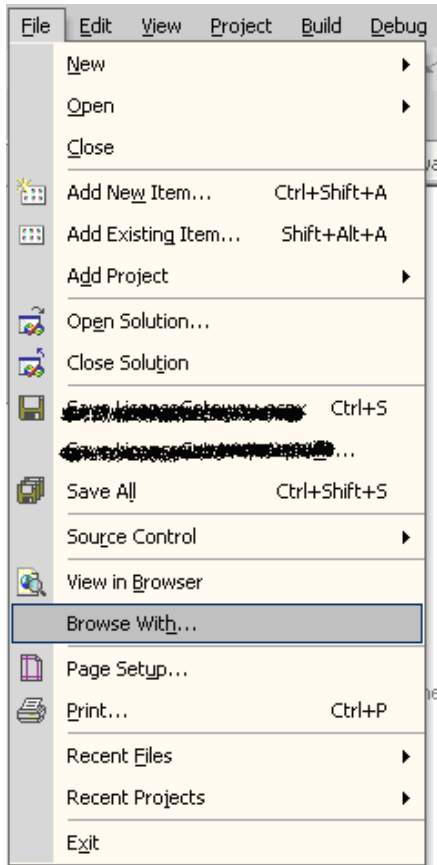


Abbildung 30: Webprojekte und Firefox 1

Im sich öffnenden Dialog einfach den gewünschten Browser auswählen und auf Default stellen.

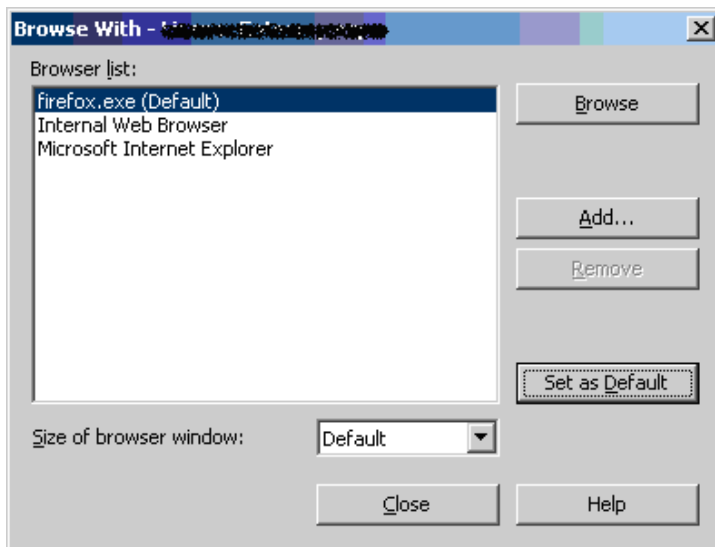


Abbildung 31: Webprojekte und Firefox 2

Nun die Projekteigenschaften öffnen. Im Bereich

Configuration Properties / Debugging

gibt es den Eintrag 'Always Use Internet Explorer'. Dieser muss auf false stehen und schon ist die Umstellung komplett.

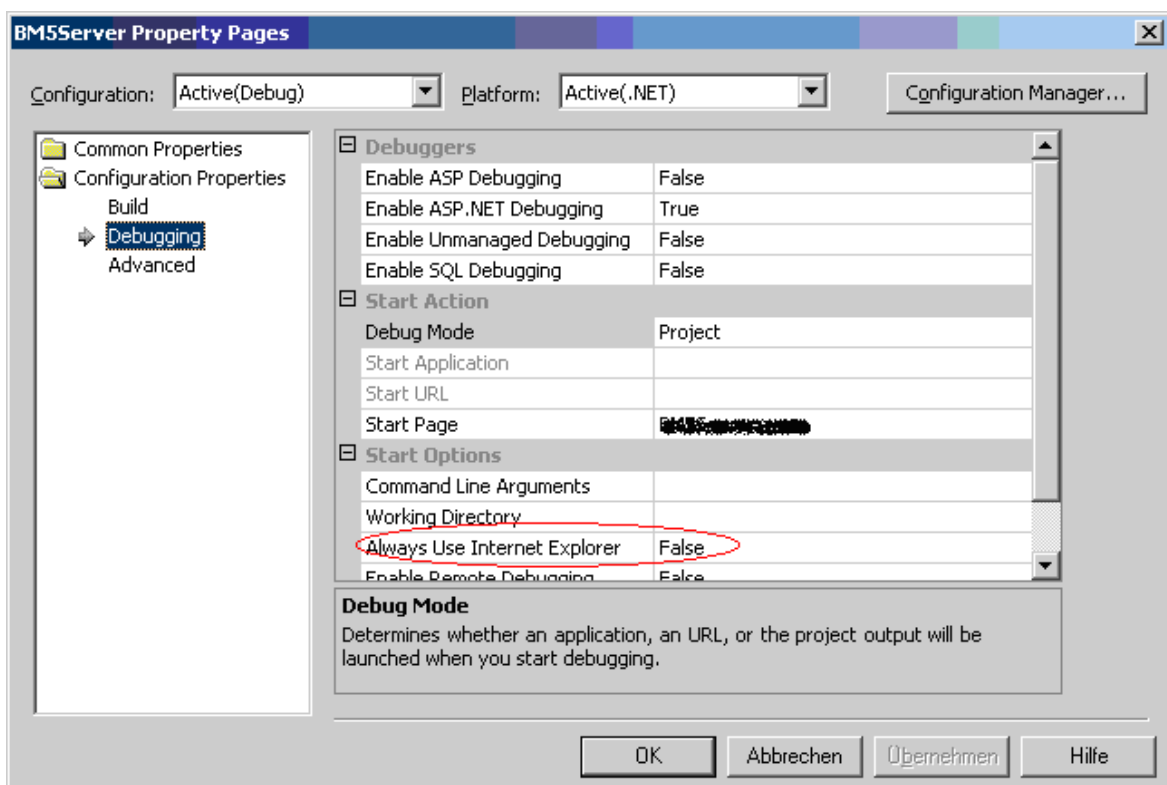


Abbildung 32: Webprojekte und Firefox 3

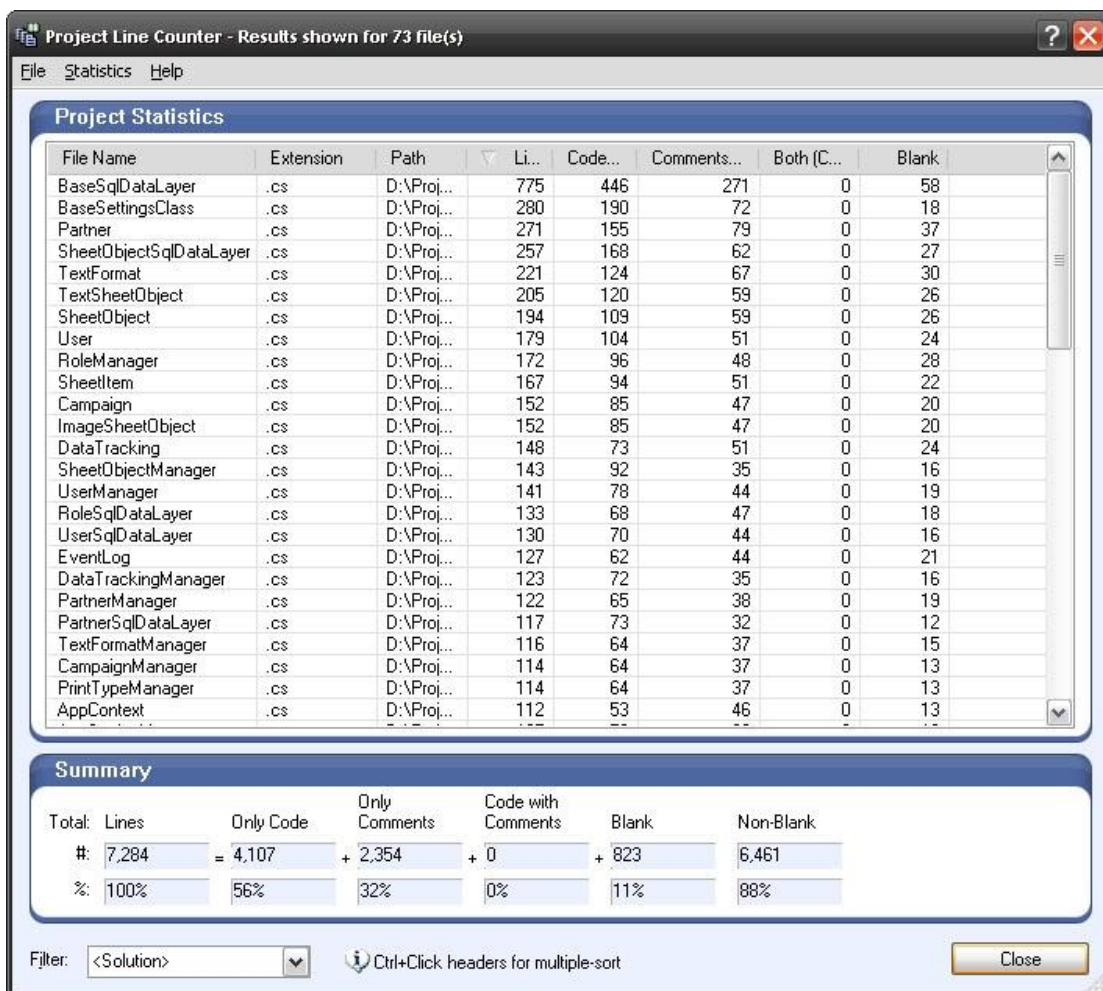
3.10. [Codebehind file in Visual Studio 2005 hinzufügen](#)

Der einfachste Weg um ein Codebehind/Codebeside File einer ASPX Seite hinzuzufügen, ist natürlich der entsprechende Schalter beim anlegen dieser Seite. Vergisst man dieses allerdings mal, oder möchte nachträglich noch ein Codebehind file hinzufügen, gibt es keine Möglichkeit dies mit einem entsprechenden Menüpunkt in Visual Studio zu erledigen.

Im Blog von [Jon Galloway](#) findet man ein kleines Marco, welches genau dieses Job erledigt. [Klick!](#)

3.11. [Project Line Counter für Visual Studio](#)

Für ein Projekt benötigte ich ein paar Statistiken, wie z.B. Anzahl der Code oder Comment Zeilen. Nach kurzer Google Suche stieß ich auf das Tool *Line Counter* von Oz Solomon.



The screenshot shows the 'Project Line Counter' application window. The title bar reads 'Project Line Counter - Results shown for 73 file(s)'. The window contains a table of file statistics and a summary section.

File Name	Extension	Path	Li...	Code...	Comments...	Both (C...	Blank
BaseSqlDataLayer	.cs	D:\Proj...	775	446	271	0	58
BaseSettingsClass	.cs	D:\Proj...	280	190	72	0	18
Partner	.cs	D:\Proj...	271	155	79	0	37
SheetObjectSqlDataLayer	.cs	D:\Proj...	257	168	62	0	27
TextFormat	.cs	D:\Proj...	221	124	67	0	30
TextSheetObject	.cs	D:\Proj...	205	120	59	0	26
SheetObject	.cs	D:\Proj...	194	109	59	0	26
User	.cs	D:\Proj...	179	104	51	0	24
RoleManager	.cs	D:\Proj...	172	96	48	0	28
SheetItem	.cs	D:\Proj...	167	94	51	0	22
Campaign	.cs	D:\Proj...	152	85	47	0	20
ImageSheetObject	.cs	D:\Proj...	152	85	47	0	20
DataTracking	.cs	D:\Proj...	148	73	51	0	24
SheetObjectManager	.cs	D:\Proj...	143	92	35	0	16
UserManager	.cs	D:\Proj...	141	78	44	0	19
RoleSqlDataLayer	.cs	D:\Proj...	133	68	47	0	18
UserSqlDataLayer	.cs	D:\Proj...	130	70	44	0	16
EventLog	.cs	D:\Proj...	127	62	44	0	21
DataTrackingManager	.cs	D:\Proj...	123	72	35	0	16
PartnerManager	.cs	D:\Proj...	122	65	38	0	19
PartnerSqlDataLayer	.cs	D:\Proj...	117	73	32	0	12
TextFormatManager	.cs	D:\Proj...	116	64	37	0	15
CampaignManager	.cs	D:\Proj...	114	64	37	0	13
PrintTypeManager	.cs	D:\Proj...	114	64	37	0	13
AppContext	.cs	D:\Proj...	112	53	46	0	13

Total:	Lines	Only Code	Only Comments	Code with Comments	Blank	Non-Blank
#:	7,284	= 4,107	+ 2,354	+ 0	+ 823	6,461
%:	100%	56%	32%	0%	11%	88%

Filter: <Solution> [v] Ctrl+Click headers for multiple-sort [i] Close [b]

Abbildung 33: Project Line Counter

Dieses Add-In für VS2003/VS2005 u. VC++ 6.0 kann die Informationen entweder für die gesamte Solution oder jedes einzelne Projekt anzeigen. Zusätzlich ist es möglich die Ergebnisse als CSV oder XML Datei zu exportieren.

Zumindest für einen kurzen Überblick reicht das Tool völlig aus, zudem ist es kostenlos.

<http://www.wndtabs.com/download/func,fileinfo/id,17/>

3.12. [Zur korrespondierenden Klammer springen](#)

Durch Zufall bin ich heute auf einem weiteren sehr nützlichen Shortcut im Visual Studio gestoßen. Mit Hilfe der Tastenkombination *Ctrl + `* (siehe Abbildung) ist es möglich direkt zur korrespondierenden Klammer zu springen. Befindet sich der Cursor z.B. bei der schließenden Klammer einer Methode, springt er direkt zur öffnenden und somit zum Anfang der Methode.

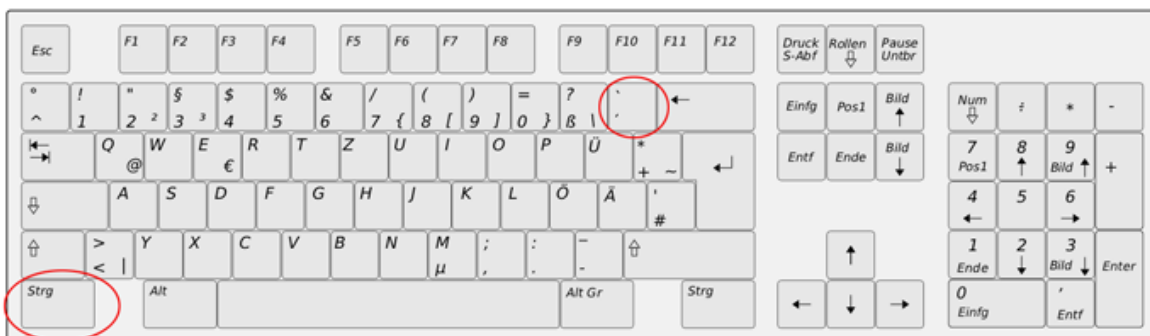


Abbildung 34: Korrespondierende Klammer

Weitere nützliche Shortcuts findet man unter folgenden URLs:

<http://www.codinghorror.com/blog/archives/000315.html>

<http://www.codeproject.com/tips/VSnetIDETipsAndTricks.asp>

<http://safari.oreilly.com/0596003609/mastvsnet-APP-C>

<http://blogs.dotnet-braunschweig.de/Karim/PermaLink,guid,2b2b0774-feba-4c88-9421-8cd44c4d7ed5.aspx>

4. ENTWICKLUNG

4.1. Allgemein

4.1.1. [Continuous Integration - kenn ich nich ..](#)

Nein? Dann dürfte ein Artikel von Martin Fowler [1] zu diesem Thema [2] wirklich weiterhelfen.

Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly. This article is a quick overview of Continuous Integration summarizing the technique and its current usage.

[1] [Martin Fowler](#)

[2] [Continuous Integration](#)

4.1.2. [Überladen vs. überschreiben](#)

Da ich heute wieder auf eine Verwechslung dieser beiden Begriffe (engl. Override, Overload) gestoßen bin, möchte ich diese kurz hier erklären.

Überladung

Beim Überladen wird die Signatur einer Methode verändert. Ert Methodename ändert sich jedoch nicht. Was sich hierbei ändert sind die Parameter bzw. die Anzahl der Parameter. Der Rückgabetyt kann sich ändern, jedoch müssen die Parameter unterschiedliche Typen aufweisen, oder die Anzahl der Parameter muss unterschiedlich sein.

Was ist die Signatur?

Unter der Signatur versteht man die Definition einer Methode, sprich mehr oder weniger der Methoden-Rumpf.

Beispiel:

```
public string CalcValue(int value1, int value2) {}
```

Was darf hier nun geändert werden? Wie gesagt, der Methoden-Name muss ident bleiben. Eine Änderung des obigen Beispiels zu

```
public string CalcValue(int value1, double value2) {}
```

wäre erlaubt. Dies bedeutet nun, dass sich auch die Typen der Parameter ändern können. Besteht allerdings nur ein Parameter vom Typ `int` namens `value1`, dann darf der es nicht sein, dass der Typ derselbe bleibt, sich jedoch der Name ändert. Diese Möglichkeit besteht nicht.

Überschreibung

Beim Überschreiben einer Methode sieht dies anders aus. In diesem Fall bleibt die Signatur dieselbe. Das heißt, weder der Rückgabewert, noch der Methodename, noch die Parameter (auch nicht deren Typen, Parameternamen, Anzahl der Parameter) ändern sich. Hier ändert sich der Innenteil der Methode, sprich die Funktionalität.

Beispielsweise wird

```
public virtual int Calc(int value1, int value2)
{
    return value1 + value2;
}
```

zu

```
public override int Calc(int value1, int value2)
{
    return value1 * value2;
}
```

Einsatzgebiete

Überladung wird beispielsweise verwendet, wenn eine Methode mehrere optionale Parameter besitzen soll bzw. unterschiedliche Eingangswerte haben soll, um Grunde aber entsprechend der Parameter eine entsprechende Aufgabe löst.

Überschreibung kommt bei der Vererbung zu tragen. So kann eine Basisklasse die Implementierung einer Methode vorschreiben, die jedoch bei jeder Ableitung eine unterschiedliche Aufgabe besitzt, aufgrund der weiteren Verwendung jedoch immer die gleiche Signatur besitzen muss.

4.2. Analyse

4.2.1. [Eigenen Code analysieren](#)

Wo gibt es Verbesserungsmöglichkeiten? Sind für die Performance relevante Codepassagen in meiner Anwendung? Oder befinden sich sogar potentielle Sicherheitslöcher im Programm?

Wer bereits mit FxCop gearbeitet hat, kennt auch die entsprechende Visual Studio 2005

Funktionalität. FxCop wurde integriert und steht somit allen Nutzern zur Verfügung. Dazu gibt es den Tab "Code Analysis" in den Projekteigenschaften. Dort können ebenfalls auch die einzusetzenden Regeln ausgewählt werden. Auch die Aktivierung der Code Analyse kann in diesem Fenster eingestellt werden. Die Ergebnisse einer Analyse werden im Error Output angezeigt.

Generell empfiehlt es sich, Code Analysis zu verwenden, um potentielle Sicherheitsprobleme und weitere "kritische" Stellen zu finden. Allerdings sollten die Warnings an das jeweilige Anwendungsdesign "angepasst" werden, da nicht alle angewandt werden können.

4.3. Software Testing

4.3.1. [Das Übel Software-Testing](#)

Gestern hatte ich ein längeres Gespräch mit einem befreundeten Entwickler. Basis der Diskussion war das Thema Software-Testing in allen Formen, Varianten und warum die Akzeptanz so gering ausfällt.

Fakt ist (zumindest laut meiner persönlichen Erfahrung), dass nur sehr wenige Firmen Software wirklich testen. Damit meine ich nun weniger, das rein zufällige Finden von offensichtlichen Bugs durch unkontrolliertes - an Zuckungen erinnerndes - Herumklicken auf der GUI, sondern vielmehr der Einsatz von einschlägigen Hilfsmitteln á Unit-Tests und Co.

Wenn ich nun kommuniziere, dass es am zusätzlichen Aufwand liegt, dann wiederhole ich mich, denn dieser dürfte es anscheinend nicht sein. Vielmehr daran, dass viele Entwickler nur dann Spass am Programmieren haben, wenn Neues geschaffen werden kann. Beim Schreiben von Tests muss jedoch bereits entwickeltes geprüft werden (in vielen Fällen), was einer Wiederholung gleich kommt. Um es auf den Punkt zu bringen: es ist einfach fad. Daher wird dieser wichtige Schritt einfach nicht durchgeführt.

Das Resultat? Nun, wer würde sich ein Auto kaufen, welches jedes Mal auf der Autobahn abstirbt, weil der Motor aufgrund der andauernden Belastung überhitzt, also quasi einen Überlauf verursacht. Oder man stelle sich vor, dass das Aufblendlicht nicht funktioniert, weil der Kontakt zur Birne fehlt, was vergleichbar mit einer Null-Reference-Exception wäre. Viele Beispiele gäbe es an dieser Stelle. Grundsätzlich möchte jeder ein voll funktionstüchtiges Auto. Genauso verhält es sich mit Software. Wenn andere Branchen so fehlerbestückte Produkte ausliefern würden, wie es Software-Branche tut, könnten wir vermutlich mit keinem Gerät bzw. keinem Produkt tatsächlich etwas anfangen. Das stimmt sehr nachdenklich.

In diesem Sinne wünsche ich einen guten Start ins Wochenende und auf dass es in Zukunft nicht auch notwendig ist, die Software zum Service zu bringen oder gar jährlich ein Pickerl (TÜV) machen zu lassen.

4.3.2. [Unit-Tests und Aufwand](#)

Danke für die vielen Rückmeldungen, die ich vor allem in Form von Emails erhalten habe. Dabei stand fast immer die Frage nach dem Aufwand im Vordergrund.

Zu diesem Thema kann ich folgendes sagen/behaupten:

Natürlich verursacht das Erstellen von Unit-Tests einen entsprechenden Aufwand, da jede Methode in allen erdenklichen Varianten getestet werden soll. Sinnvollerweise wird der Unit-Tests parallel zur zu testenden Klasse entwickelt. Dadurch erhöht sich zwar der Aufwand für die Entwicklung von Klassen, man bedenke aber das große ABER:

Durch die ständigen Test-Durchläufe kristallisieren sich sehr schnell Fehler heraus bzw. wo es nach einer durchgeführten Änderung ein wenig zwickt. Sollte dann doch einmal ein Fehler durchrutschen, kann mit Hilfe der Unit-Tests dieser recht schnell nachvollzogen bzw. überhaupt gefunden werden. Das erspart sehr viel Zeit. Vermutlich mehr, als man sich durch das Nichtschreiben von Unit-Tests sparen würde.

Jeder Leser kann sich nun selbst ein Bild davon machen. Einfach die durchschnittliche Anzahl der Bugs heranziehen und kurz darüber nachdenken, wie schwer so manche davon zu finden ist. Viele davon wären mit Unit-Tests erst gar nicht zustande gekommen.

Ich hoffe dieser Beitrag regt zum Nachdenken und Nachrechnen an und liefert so für jeden von Euch ein Ergebnis bezgl. der Sinnhaftigkeit von Unit-Tests.

4.3.3. [Grundlagen: Testgetriebene Entwicklung oder auch test-driven Development](#)

Die testgetriebene Entwicklung kommt aus dem Bereich der agilen Softwareentwicklung und legt fest, dass Software-Tests vor der Entwicklung der zu testenden Komponenten erstellt werden. Zur Verwendung kommen sogenannte Grey-Box-Tests. Diese vereinen die Vorteile der White-Box-Tests [1] und der Black-Box-Tests [2] in sich:

1. Die Software-Tests werden vom Entwickler der Komponente selbst geschrieben (White-Box-Test)
2. Der Test wird ohne Kenntnis der Komponente (da diese ja noch nicht entwickelt ist) erstellt (Black-Box-Test)

Der Vorteil dieser Variante liegt darin, dass nicht um Fehler "herumgetestet" wird. D.h. die Tests werden nicht der Komponente angepasst, sondern die Komponente muss so funktionieren, wie dies im Test festgelegt wurde.

Bei dieser Art der Tests empfiehlt es sich jedoch eine hohe Disziplin an den Tag zu legen und zusätzliche Arbeitsweisen aus der agilen Softwareentwicklung zu nutzen. Zusätzlich sollte nicht auf Black-Box-Tests verzichtet werden.

[1] [White Box Tests](#)

[2] [Black Box Tests](#)

4.3.4. [Grundlagen: White Box Tests](#)

Bei den White Box Tests wird im Gegensatz zu den Black Box Tests die Implementierung einer Komponente getestet. Dies bedeutet, dass vor dem Entwickeln des Tests ein Blick auf den Sourcecode durchaus erlaubt ist.

Ziel der White Box Tests ist es, dass bezüglich des Sourcecodes bestimmte Hinlänglichkeitskriterien erfüllt werden. Eine komplette Fehlerfreiheit kann durch diese Tests nicht sichergestellt werden. Eine Kombination mit anderen Test-Methoden ist anzuraten.

Weitere Methoden der Softwaretests:

[Black Box Tests](#)

[Grey Box Tests](#)

4.3.5. [Grundlagen: Black Box Tests](#)

Die Black-Box-Tests gehören den allgemeinen Softwaretests an und stellen eine Methode dar, Tests für Komponenten zu entwickeln, die noch nicht erstellt wurden. Für die einzelnen Tests wird die Spezifikation der Methoden, Klassen etc. herangezogen, jedoch nicht die dahinterliegende Implementierung (funktionsorientiertes Testen).

Das Ziel besteht darin, die Software hinsichtlich der Spezifikationen zu prüfen. Diese geben Schnittstellen, Definitionen und Ergebnisse vor, welche von den einzelnen Komponenten einzuhalten sind und entsprechend geprüft werden müssen.

Ein Black-Box-Test kann jedoch sehr aufwändig sein und birgt auch ein weiteres Problem in sich:

In der Softwareentwicklung werden Komponenten ständig weiter entwickelt. Daraus ergeben sich neue Funktionalitäten, die ebenfalls getestet werden müssen. Diese Tests sind jedoch in den Black-Box-Tests nicht enthalten, da diese bereits vor der Komponente entwickelt werden. Erfolgreiche Testläufe bedeuten daher jedoch nicht zwangsweise, dass die gesamte Funktionalität der Komponente zu einem späteren Zeitpunkt getestet wurde, sondern lediglich die ursprüngliche Spezifikation.

Zusätzlich gibt es noch weitere Verfahren:

[Grey Box Test](#)

[White Box Test](#)

4.3.6. [\[Tutorial\] Unit-Tests mit Visual Studio .NET](#)

Einführung und Ausblick

Bei fast allen Fragestellungen in diversen Foren gibt es Fragen zu Fehlern, die auf dreierlei Arten auftreten:

1. Mangelndes Studieren der zur Verfügung stehenden Quellen. Dies inkludiert nicht nur diverse Artikel, die es im Internet zu finden gibt (ein Beispiel hier wäre das MSDN), sondern auch in Form von Büchern oder den zu den verwendeten Werkzeugen mitgelieferten Ressourcen.

2. Fehlendes Verständnis und Wissen rund um den Debugger. Immer wieder muss ich feststellen, dass der Debugger wenig bis gar keine Beachtung findet, obwohl sich durch ihn das wohl beste "Werkzeug" überhaupt offenbart. Auf schnelle Art und Weise können nicht nur Syntax-Fehler gefunden werden, nein, es können auch Variableninhalte abgefragt werden, Objektinformationen bezogen und sogar Abfragekommandos an den Compiler übergeben werden.

3. Ignoranz von Unit-Tests. Die Verwendung von Unit-Tests wird in vielen Fällen nur Profis zugetraut und aus diesem Grunde wird ihnen kaum Beachtung geschenkt - oder aber, einfach das fehlende Wissen um diese Möglichkeit.

Dieser Artikel soll zeigen, dass selbst Programmier-Anfänger sehr einfach Unit-Tests verwenden können.

Begriffserklärung

Bevor wir aber tiefer in die Materie einsteigen, muss der Begriff Unit-Test erklärt werden.

Durch Unit-Tests ist es möglich, Testklassen zu schaffen, die automatisiert andere, vorhandene Klassen testen.

So werden pro Testklasse folgende Dinge der zu testenden Klasse angegeben und getestet:

- alle Methoden
- alle Überladungen
- in allen erdenklichen Übergabeparametern (auch sinnlosen Varianten)

Vor allem die "sinnlosen Varianten" sind für die Tests sehr wertvoll: Dadurch werden Überläufe, und vor allem auch Fehleingaben durch User (diese passieren immer und überall) mit in den Test einbezogen und im Fehlerfalle kann dies bevor das Produkt den User erreicht nachgebessert werden.

Der Sinn hinter solchen Tests ist der, dass wie oben beschrieben, problematische Parameterübergaben im Fehlerfalle behandelt werden (da auf das Testergebnis entsprechend zu reagieren ist). Da eine Software laufend weiterentwickelt wird, werden auch ständig an allen erdenklichen Stellen Änderungen vorgenommen. Dadurch ist es in manchen Fällen schwer zu sagen, ob eine andere Stelle noch korrekt funktioniert. Durch diese Unit-Tests kann dies einfach festgestellt werden. Dazu sind nach der durchgeführten Änderung die Tests auszuführen und wenige Sekunden bis Minuten später (Abhängig von der Projektgröße) hat der Programmierer Gewissheit.

Notwendige Tools

Zu Beginn muss natürlich geklärt werden, welche Tools notwendig sind, um Unit-Tests unter dem Visual Studio verwenden zu können.

Aus meiner Erfahrung haben sich folgende Produkte als äußerst hilfreich erwiesen:

TestDriven.NET [1]
NUnit [2]

Bei TestDriven.NET handelt es sich um Unit-Testing Add-In für Visual Studio, welches mit unterschiedlichsten Unit-Testing-Tools zusammenarbeiten kann. Zu erwähnen wären hier NUnit, MbUnit und csUnit.

Bei NUnit handelt es sich um ein Test-Unit Framework, mit dessen Hilfe Unit-Tests durchgeführt werden können.

Installation

An diesem Punkt möchte ich nur an die entsprechenden Installations-Hinweise der Hersteller verweisen:

NUnit: <http://www.nunit.org/index.php?p=installation&r=2.2.6>

Bei TestDriven.NET ist lediglich die Installationsdatei zu starten. Daraufhin wird das Add-In im Visual Studio registriert und steht fortan zur Verfügung.

Vorarbeiten

Um nun für ein neues oder bereits bestehendes Projekt Unit-Tests anzuwenden, sind kleine Vorarbeiten notwendig.

Idalerweise empfiehlt es sich, Unit-Tests in ein eigenes Projekt auszulagern. Dieses Projekt muss natürlich Teil der Visual Studio Solution sein.

Nach dem Anlegen dieses Projektes, ist eine Referenz auf die nunit.framework.dll zu setzen. Ist dies geschehen, kann es mit einem konkreten Beispiel weitergehen.

Ein konkretes Beispiel

Gehen wir davon aus, dass unser Projekt einen Logger besitzt. Dieser hat die Aufgabe, allfällige Fehler in eine Logdatei zu schreiben. Natürlich muss dieser getestet werden, ob er auch den an ihn gestellten Anforderungen gerecht wird.

Dazu erstellen wir eine Testklasse LoggerTest. Die neue Klasse muss zusätzlich mit dem Attribut [TestFixture] markiert werden. Danach erstellen wir die einzelnen Methoden, welche die einzelnen Tests darstellen. Beispielsweise könnte dies folgendermaßen aussehen:

```
public void LoggerLog()
{
    try
    {
        Logger log = new Logger();
        log.LogPath = @"C:temptemp.log";
        log.Log("test");
        Assert.IsTrue(true);
    }
    catch (Exception ex)
    {
        Assert.Fail(ex.Message);
    }
}
```

Wie an diesem Beispiel zu sehen ist, müssen die zu testenden Methoden mit dem Attribut `[Test]` versehen werden.

Mit einem Rechtsklick auf die entsprechende Datei kann nun mittels "Run Test(s)" der Test gestartet werden. Wird dieser Befehl auf das Projekt angewandt, werden alle darin enthaltenen Tests gestartet.

Nach dem Durchlauf der Tests erscheint im Ausgabe-Fenster die Angabe, welche Tests durchgelaufen (also erfolgreich waren) und welche nicht.

Wann ein Test als erfolgreich und wann als nicht erfolgreich gilt, muss vom Entwickler selbst angegeben werden. Im Falle einer Exception darf der Test natürlich nicht als erfolgreich angeführt sein. Wurden alle Punkte durchlaufen und stimmt das Ergebnis, kann der Test als erfolgreich markiert werden.

Dafür zuständig ist die Klasse `Assert`. Diese hat einige Methoden, die für diese Aufgabe sehr hilfreich sind:

`Assert.Fail`: Hier ist ein String zu übergeben, der angibt, um welchen Fehler es sich handelt. Dies kann die `Message`-Eigenschaft einer Exception sein, oder auch ein selbst definierter String, um mitzuteilen wo bei was ein Fehler aufgetreten ist.

`Assert.IsTrue`: Hier werden normalerweise Vergleiche angegeben. Beispielsweise das Ergebnis der Methode und das Ergebnis, welches die Methode liefern sollte. Sind beide Ergebnisse ident, ist der Test durchgelaufen, andernfalls nicht.

`Assert.IsFalse`: Dies funktioniert wie `Assert.IsTrue`, nur in die entgegengesetzte Richtung.

Konklusio

Dieses kleine Tutorial lieferte einen kurzen Einblick in das Thema Unit-Tests unter Visual Studio und zeigt durchaus, dass sich dahinter keine komplizierten Abläufe verbergen. Stöbert man ein wenig in den angegebenen Internet-Ressourcen herum, können sehr schnell gute und hilfreiche Ergebnisse erreicht werden.

Sollten dennoch Fragen zu dem Thema entstehen - welche durch die angegebenen Ressourcen nicht abgedeckt werden können, kann der werte Leser mich unter csharp@gmx.at erreichen.

Referenzen

[1] <http://www.testdriven.net>

[2] <http://www.nunit.org>

4.3.7. [Nicht ausgeführte UnitTests mit TestDriven.NET](#)

Grade wieder drüber gestolpert:

Mit [TestDriven.NET](#) ist es möglich [UnitTests](#) bequem aus Visual Studio zu starten und auszuwerten. Über das ContextMenu der Projektmappe ist es außerdem möglich, alle vorhandenen Tests des Projekts zu starten. Allerdings sollte man darauf achten, dass die Test-Klassen auch tunlichst als `public` gekennzeichnet sind. Ansonsten werden diese Tests nicht ausgeführt.

Wer noch nie etwas von [UnitTests](#) oder [TestDriven.NET](#) gehört hat, sollte einen Blick in die aktuelle Ausgabe der Zeitschrift [Visual Studio one](#) werfen und sich den Artikel [Visual Studio goes Unit Testing](#) von [Norbert Eder](#) zu Gemüte führen. .

4.3.8. [Externes Configuration File benutzen](#)

Heute musste ich in einer Konsolenanwendung ein Configuration File benutzen, dass nicht direkt zur Anwendung gemapped ist. Noch in blasser Erinnerung an das 1.1 Framework hatte mich mich schon mit dem Gedanken abgefunden, mir einen Xml-Parser zu schreiben, der die entsprechend Einträge sucht. Tatsächlich aber bietet das Framework 2.0 dafür entsprechende Klassen und Methoden. Somit ist das Problem mit ein paar Zeilen schnell gelöst:

```
ExeConfigurationFileMap configFile = new
ExeConfigurationFileMap();
configFile.ExeConfigFilename = "test.config";

Configuration config =
ConfigurationManager.OpenMappedExeConfiguration(configFile,
ConfigurationUserLevel.None);
AppSettingsSection section =
(AppSettingsSection)config.GetSection("appSettings");

string configValue = section.Settings["Folder"].Value;
```

4.3.9. [ToolTips einzelner Items einer CheckBoxList setzen](#)

In einem Forum tauchte die Frage auf, wie man die Eigenschaft `ToolTip` einzelner `CheckBoxen` einer `CheckBoxList` setzt. Das `CheckBoxList` Control bietet selbst eine Eigenschaft `ToolTip`, allerdings setzt man dann gleich den `ToolTip` für alle `CheckBoxen`.

Die Eigenschaft `ToolTip` selbst bewirkt, daß das `Html`-Attribut `Title` gesetzt wird. Man muß also nur durch alle `CheckBoxen` iterieren und dieses Attribut manuell setzen. Der Code hierfür ist recht einfach:

```
for (int i = 0; i < CheckBoxList1.Items.Count; i++)  
{  
    ListItem item = (ListItem)CheckBoxList1.Items[i];  
    item.Attributes.Add("title", "TestTooltip " + i);  
}
```

Um den `ToolTip` einzelner Items der Liste direkt setzen, ohne durch die gesamte Liste zu iterieren, habe ich mir eine kleine Methode geschrieben:

```
private void SetCheckBoxToolTipByValue(string value, string  
tooltip)  
{  
    ListItem item =  
(ListItem)CheckBoxList1.Items.FindByValue(value);  
  
    if (item != null)  
        item.Attributes.Add("title", tooltip);  
}
```

Die Methode sucht anhand eines `Values` den entsprechenden Eintrag und setzt den `Title`.

Das komplette Beispielprojekt gibt es [hier](#).

4.4. Deployment

4.4.1. [Deploying unter .NET 3](#)

Alles zum Thema Deployment unter .NET 3 findet sich wohl unter [1]. Wer also schon früh informiert sein will, der sollte sich diesen Link genauer ansehen.

[1] [Deploying Microsoft .NET Framework Version 3](#)

4.5. Design Patterns

4.5.1. [Was sind Design Patterns?](#)

Bevor unterschiedlichste Design-Patterns erklärt werden, muss definiert werden, worum es sich bei Design-Patterns handelt.

Hier eine kurze Definition:

"Jedes Muster beschreibt in unserer Umwelt ein beständig wiederkehrendes Problem und erläutert den Kern der Lösung für dieses Problem, so dass diese Lösung beliebig oft angewendet werden kann, ohne sie jemals ein zweites Mal gleich auszuführen"

A Pattern Language von Christopher Alexander, Oxford University Press 1977.

Was genau bedeutet dies? Jedes Pattern wird/wurde für eine bestimmte Aufgabe entwickelt und kann darin generisch eingesetzt werden. Sie sollen also häufig gestellte Aufgaben vereinfachen. Weiters handelt es sich um eine Beschreibung der zusammen arbeitenden Objekte und Klassen.

Zur weiterführenden Recherche möchte ich erwähnen, dass es unterschiedliche Typen von Design-Patterns gibt:

Erzeugungs-Pattern
Struktur-Pattern
Verhaltens-Pattern

Sollten nähere Erklärungen von Nöten sein, werde ich dies an den entsprechenden Stellen nachholen, oder zu den unterschiedlichen Typen noch entsprechende Einträge schreiben.

4.5.2. [Design Patterns: Die Geschichte](#)

Unter [Was sind Design Patterns](#) wurde der Begriff Design Pattern bereits definiert und auch die unterschiedlichen Arten wurden angeführt.

In diesem Beitrag wird auf die Geschichte der Design Patterns eingegangen.

Bereits in den 1970er Jahren wurde die erste Sammlung von Entwurfsmustern erstellt - allerdings von einem Architekten namens Christopher Alexander. Die Idee dahinter hat sich seitdem nicht verändert. Nur fand seine Sammlung wenig Anklang unter anderen Architekten, in der Softwareentwicklung wurde die Idee jedoch bald darauf aufgegriffen und erfreut sich großer Beliebtheit. Ende der 1980er wurde die Sammlung von Christopher Alexander von Kent Beck und Ward Cunningham aufgegriffen und entwickelten auf deren Basis Entwurfsmuster für grafische Benutzerschnittstellen.

Eine neue Ära begann dann mit Erich Gamma. Nach seiner Promotion an der Universität Zürich, 1991, ging er in die USA und verfasste zusammen mit Richard Helm, Ralph Johnson und John Vlissides das Buch *Design Patterns - Elements of Reusable Object-Oriented Software*. In diesem Buch wurden 23 Design Patterns beschrieben. Dies verhalf den Entwurfsmustern zum Durchbruch. Die vier Autoren sind gemeinhin auch unter Gang of Four (GoF) bekannt.

Zur Übersichtlichkeit verwendete die GoF ein einheitliches Schema um die Design Patterns zu beschreiben. Nachfolgend eine kurze Übersicht:

Musternamen und Klassifikation
Zweck
Synonyme

- Motivation
- Anwendbarkeit
- Struktur
- Beteiligte Klassen (Akteure)
- Zusammenspiel der involvierten Klassen
- Vor- und Nachteile
- Implementierung
- Beispielcode
- Praxiseinsatz
- Querverweise

Anhand dieses Schemas konnte ausreichend Information zum entsprechenden Design Pattern geliefert werden (Wann ist es einsetzbar, etc.).

4.5.3. [Patterns: Command Pattern](#)

Das nächste Pattern, welches ich in meiner Rubrik Patterns vorstellen möchte, ist das Command-Pattern.

Defintion

Das Command Pattern ermöglicht die Repräsentation von Aktivitäten in eigenständigen Objekten.

Was bedeutet dies in der Praxis?

Innerhalb der selben Struktur können unterschiedliche Commands ausgeführt werden, die jeweils eine bestimmte Aufgabe besitzen. Unabhängig davon, welche Aufgabe ein Command hat, muss das ausführende Konstrukt nicht verändert werden. Zusätzlich besteht die Möglichkeit, Commands in beispielsweise einer Queue abzulegen um hintereinander abgearbeitet zu werden.

Beispiel-Implementierung

Der Aufbau des Command Patterns wird zwar in manchen Fällen recht aufwändig und kompliziert dargestellt, gestaltet sich in der Praxis jedoch als sehr einfach. Es wird ein Interface ICommand erstellt welches die notwendige Methode Execute() zur Verfügung stellt. Jeder konkrete Command implementiert dieses Interface und die Methode Execute() wird dabei mit der Logik des Commands befüllt. In einigen Fällen macht es auch durchaus Sinn, einen abstrakten Command einzufügen, der bereits Basisfunktionalitäten zur Verfügung stellt.

```
public interface ICommand
{
    string Message { get;set;}
    void Execute();
}
```

Dieses Interface schreibt also die Methode Execute() vor, die von jedem Command implementiert werden muss. Weiters wird eine Eigenschaft Message vorgeschrieben, die eine entsprechende Nachricht speichern soll. Konkreter Anwendungsfall wäre eine Fehlermeldung

wenn der Command auf einem entfernten Host ausgeführt wird (kann als eine Ergänzung zu einem State gute Dienste leisten).

In diesem Beispiel wird nun ein abstrakter Command erstellt, der die Eigenschaft als Basisfunktionalität zur Verfügung stellt:

```
public abstract class Command : ICommand
{
    private string message = null;

    public string Message
    {
        get { return this.message; }
        set { this.message = value; }
    }

    public abstract void Execute();
}
```

Alle Commands, die nun vom abstrakten Command ableiten, müssen die Eigenschaft Message nicht mehr implementieren, da diese bei der Ableitung übernommen wird. Lediglich die Methode Execute ist zu überschreiben. Dies wird anhand des folgenden Sourcecodes gezeigt:

```
public class ConcreteCommand : Command
{
    #region ICommand Members

    public override void Execute()
    {
        this.Message = "This is a concrete command
implementation.";
    }

    #endregion
}
```

Hier passiert nichts anderes, als dass die Eigenschaft Message mit einem String befüllt wird.

Weiters wird von mir noch eine Klasse Executor benutzt, der für die Ausführung der Commands zuständig ist.

```
public class Executor
{
    public void ExecuteCommand(ICommand ic)
    {
        ic.Execute();
    }
}
```

Wie hier zu sehen ist, wird ein ICommand übergeben, der anschließend ausgeführt wird. Dadurch kann jeder beliebige Command ausgeführt werden und eine Erweiterung ist nur dann notwendig, wenn der Executor selbst diverse neue Features bekommt.

Zusammenfassung

Da im Falle des Command Patterns nur mehr die einzelnen Commands implementiert werden

müssen, das ausführende Framework sich jedoch nicht ändert, eignet sich dieses Pattern sehr gut für eine Client-Server Kommunikation und wird daher auch oft eingesetzt, wenn Clients beispielsweise mit WebServices interagieren.

Nachfolgend findet sich ein kleines Testprogramm, welches den gesamten Ablauf dieses Patterns zeigt.

Beispiel-Anwendung: [Download](#)

4.5.4. [Singleton Pattern](#)

Das Singleton-Pattern ist wohl eines der bekanntesten Design-Pattern überhaupt. Ziel ist es, dass es innerhalb einer Anwendung von einer Klasse nur ein Objekt gibt. Daher kann global auf dieses Objekt zugegriffen werden.

Einsatzgebiete für das Singleton-Pattern finden sich einige:

- Logging-System welches Logging-Daten in eine Datei schreibt
- Druckerqueue in der alle Dokumente der Reihe nach abgearbeitet werden sollen
- Implementierung eines globalen Storages
- etc.

Das Singleton-Pattern gehört zu den Erzeugungsmustern (engl. Creational Patterns).

Im nachfolgenden Sourcecode-Beispiel wird ein thread-sicherer Singleton abgebildet. Thread-sicher bedeutet, immer nur ein Thread auf eine Ressource zugreifen kann. Dadurch können ungewollte Effekte vermieden werden.

```
public class Singleton
{
    private static Singleton instance = null;
    private static object lockObject = new object();

    private Singleton()
    {
    }

    public static Singleton GetInstance()
    {
        lock (lockObject)
        {
            if (instance == null)
                instance = new Singleton();
        }
        return instance;
    }
}
```

Zu beachten ist, dass ein Singleton einen privaten Konstruktor hat. Dadurch wird gewährleistet, dass ausserhalb der Klasse kein Objekt der Singleton-Klasse erstellt werden kann. Dafür wird die Methode `GetInstance()` verwendet. Hier findet die Überprüfung statt, ob es bereits eine Instanz gibt. Wenn nicht, wird eine neue erstellt, ansonsten wird die

bestehende Instanz zurückgegeben. Prinzipiell kann die Methode `GetInstance()` auch als Getter-Property abgebildet werden.

Die Thread-Sicherheit bringt uns in diesem Fall das Schlüsselwort `lock`. Diesem kann als Parameter ein Objekt - welches gesperrt werden soll - übergeben werden. Solange das übergebene Objekt durch einen Lock gesperrt wird, kann kein anderer Thread darauf zugreifen.

Weitere Informationen zu diesem Thema sind unter anderem im Weblog von [Dirk Primbs](#) zu finden.

5. TOOLS

5.1. [Tools: DotLucene - Fulltext Search Engine for .NET](#)

Wer für diverse Anwendungsfälle eine kostengünstige und vor allem schnell Volltextsuche benötigt, der sollte sich auf jeden Fall [DotLucene](#) genauer ansehen.

Hinter diesem Open-Source-Projekt verbirgt sich eine vielversprechende Lösung für dieses Thema.

Ein kurzer Auszug aus der Featureliste:

- Gute Performance
- Ranking
- Hervorhebung der Suchbegriffe in den Ergebnissen
- Suche nach Metadaten
- Speicherung von vollindizierten Dokumenten

Durch die Benutzung der [Online Demo](#) kann man sich von der Geschwindigkeit und von einigen Features überzeugen.

Im gleichen Atemzug kann man sich auch den - auf der Seite von DotLucene vorgeschlagenen - Indexing Server [Seek a File](#) ansehen. Der Server läuft als Windows Dienst und kann folgende und weitere Dateitypen indizieren:

- DOC
- PDF
- XLS
- PPT
- RTF
- HTML
- TXT
- XML

Ein Blick auf diese Tools sollte sich als durchaus lohnen.

5.2. [GUI für Windows Installer XML \(WiX\)](#)

[Windows Installer XML](#) dürfte dem einen oder anderen durchaus bekannt sein.

The Windows Installer XML (WiX) is a toolset that builds Windows installation packages

from XML source code. The toolset supports a command line environment that developers may integrate into their build processes to build MSI and MSM setup packages.

Nun, auch hier wird es ähnlich wie bei [Sandcastle](#) mit der Konfiguration ein wenig zeitintensiver. Dafür gibt es aber Abhilfe.

[CalmWix](#). Es gibt zwar einige grafische Oberflächen für WiX, aber CalmWix verspricht, die funktionsreichste zu sein bzw. generell zu werden. Ein erster [Screenshot](#) zeigt die ansprechende Oberfläche (im Stil von Visual Studio 2005 gehalten) und läßt auf wesentlich mehr hoffen.

Ein Release gibt es noch nicht, aber wer möchte, kann sich den [Sourcecode](#) genauer ansehen.

5.3. [Sandcastle Helferleins](#)

Wer [Sandcastle](#) als Ersatz für beispielsweise [NDoc](#) einsetzt oder einsetzen möchte, dem sei ein Blick auf zwei Helferlein zu empfehlen:

[Sandcastle Help File Builder](#)

Eine grafische Oberfläche, welche die Bedienung und Konfiguration von Sandcastle vereinfacht.

[Sandcastle Add-in](#)

Add-in für Visual Studio 2005.

Und wer noch immer nicht genug hat, der kann noch einen Blick auf die [MSBuild Scripts für Sandcastle](#) werfen.

5.4. [ReSharper UnitRun: kostenloser Testdriven.NET Gegenspieler](#)

Von JetBrains gibt es ein kostenloses Visual Studio Add-In mit dem Unit Tests ausgeführt werden können: ReSharper TestRun [1].

Unterstützt werden folgende Testing-Frameworks:

NUnit
csUnit

Ein näherer Blick auf dieses Tool lohnt sich auf jeden Fall.

[1] [UnitRun Homepage](#)

5.5. [CCTray: CruiseControl.NET Build-Fortschritt im Überblick](#)

Thomas Darimont hat mich auf ein weiteres hilfreiches Tool für CruiseControl.NET hingewiesen. CCTray [1]. Damit läßt sich der Build Progress überwachen und das Tool erlaubt es, in einige Operationen einzugreifen.

[1] [CCTray Website](#)

5.6. [CCNetConfig: CruiseControl.NET Konfigurationsdateien mittels Win-Forms-Anwendung erstellen](#)

Wer eine CruiseControl.NET [1] Installation sein Eigen nennt, dem dürfte CCNetConfig [2] als Unterstützung hilfreich sein. Mittels grafischer Oberfläche können damit CC.NET Konfigurations-Dateien einfachst erstellt werden.

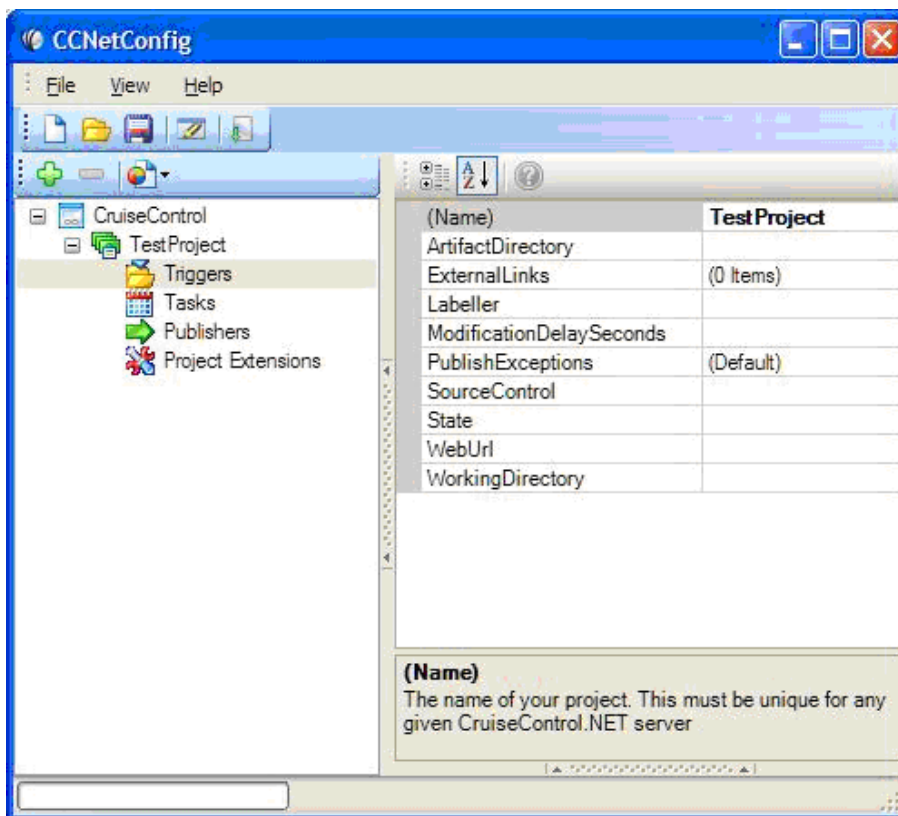


Abbildung 35: CCNetConfig

[1] [CruiseControl.NET](#)

[2] [CCNetConfig](#)

5.7. [WMI Code Creator](#)

Mehr zufällig als gewollt, fand ich gestern ein kleines Tool von Microsoft zur Generierung von Code für die Arbeit mit der WMI. Dieses Tool erlaubt es in einer GUI WMI Namespaces und Klassen auszuwählen und erstellt entsprechend der Auswahl passenden Sourcecode.

Folgende Sprachen werden unterstützt:

- C#
- Visual Basic .NET
- Visual Basic Script (VBS)

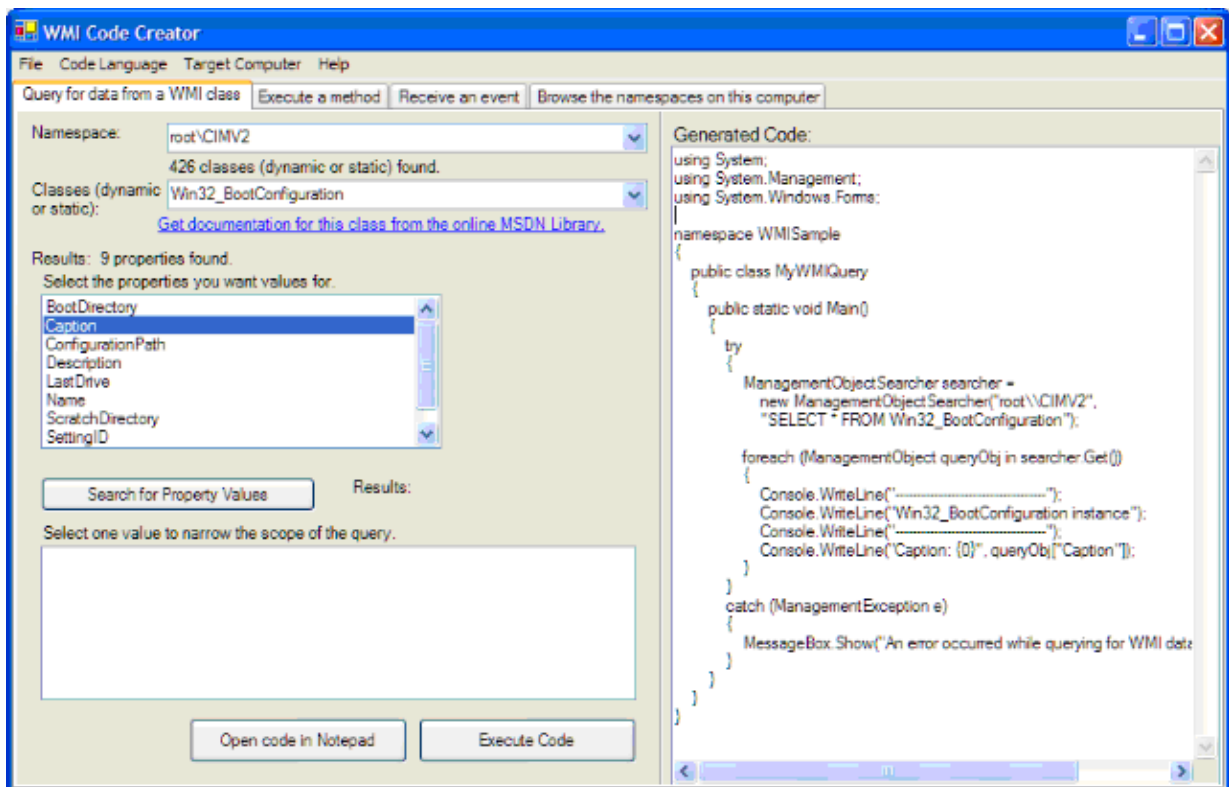


Abbildung 36: WMI Code Creator

Weitere Features

- Browsing namespaces
- Method execution
- Support of remote computer
- and more ...

[Download WMI Code Creator 1.0](#)

5.8. [Tipp: Guidande Explorer](#)

Für die Entwicklung unter .NET gibt es ja viele hilfreiche Tools und Anwendungen. Den [Guidance Explorer](#) muss man hier jedoch immer wieder einmal erwähnen.

Was bietet dieses Tool? Nun ja, Guidelines, Patterns, Anti-Patterns, Checklists, Code Samples, und Test Cases. Das Angebot ist wirklich gut, wird ständig erweitert und sollte bei keinem .NET Programmierer fehlen.

Downloaden und ansehen! Echt hilfreich.

5.9. [Documentation Generator: CodeDoc](#)

Aus Zufall bin ich heute auf den Documentation Generator CodeDoc [1] gestoßen. Das Teil ist Freeware und scheint die notwendigsten Funktionalitäten mitzubringen. Die Demo sieht mal ganz gut aus. Ein Test wird sich wohl nicht vermeiden lassen. Wer also noch auf der Suche nach einem entsprechenden Tool ist, der könnte sich CodeDoc durchaus mal genauer ansehen.

[1] [CodeDoc Documentation Generator](#)

5.10. [LINQ - Kennst du schon?](#)

Vergangenes Wochenende habe ich mich ein wenig mit LINQ beschäftigt. Mit Hilfe dieser Erweiterung lassen sich sehr einfach Datenquellen abfragen. LINQ erweitert C# und VB.NET um eine native Sprach-Syntax, Datenquellen abzufragen. Von LINQ gibt es zwei unterschiedliche Varianten:

Mit DLINQ können relationale Datenquellen abgefragt werden. Die zweite Variante nennt sich XLINQ und ermöglicht die Abfrage von XML-Daten.

Und so sieht zum Beispiel eine DLINQ-Abfrage aus:

```
public void Linq7() {  
    List products = GetProductList();  
  
    var productNames =  
        from p in products  
        select p.ProductName;  
  
    Console.WriteLine("Product Names:");  
    foreach (var productName in productNames) {  
        Console.WriteLine(productName);  
    }  
}
```


Weitere Informationen zu LINQ sind auf der [LINQ Project Homepage](#) zu finden.

5.11. [ASP.NET Deployment Tool](#)

Visual Studio 2005 bietet von Haus aus schon sehr gute Möglichkeiten eine ASP.NET Anwendung zu veröffentlichen. Allerdings gibt es keine Möglichkeit ein bestimmtes Deployment Schema zu speichern oder festzulegen.

Das ASP.NET Deployment Tool von [Andreas Kraus](#) bietet genau dieses Feature. Hat man erst mal ein Projekt angelegt, gibt es verschiedene mögliche Optionen.

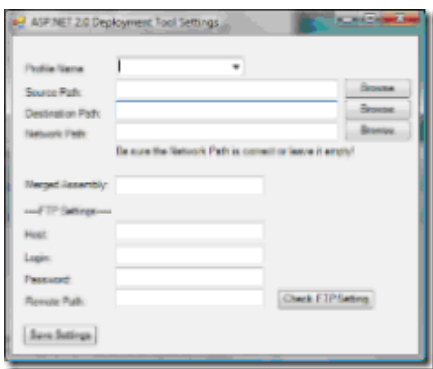


Abbildung 37: ASP.NET Deployment Tool 1

Neben dem einfachen lokalen Deployment ist es möglich die Dateien per Netzwerkpfad oder FTP zu veröffentlichen. Außerdem kann eingestellt werden, ob das Zielverzeichnis gelöscht, die Web.Config übertragen oder alle Assemblies in ein einziges zusammengefasst werden sollen.

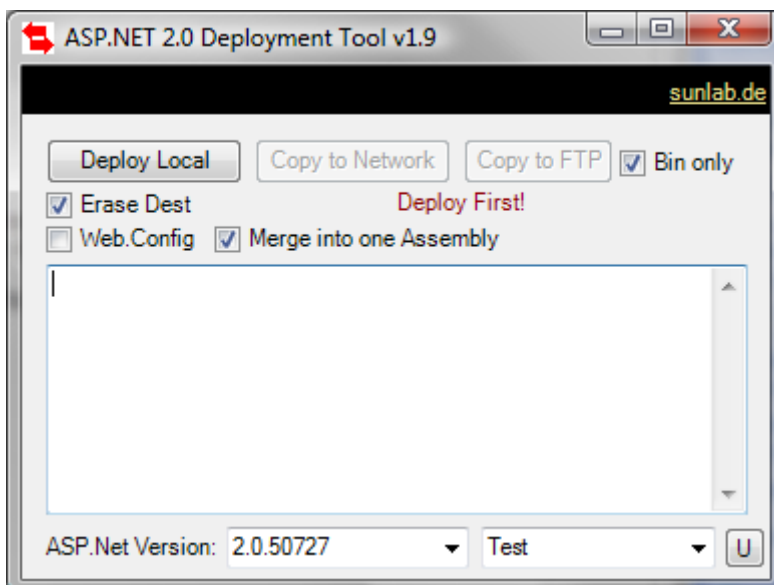


Abbildung 38: ASP.NET Deployment Tool 2

Auch wenn das Programm selbst noch mit kleinen Bugs zu kämpfen hat, ist es eine sehr gute Alternative zum Visual Studio integrierten Deployment.

Mehr Informationen und den Download gibt es unter <http://www.sunlab.de/Tools.aspx>

5.12. Lokalisierung des Community Servers

Nach erfolgreicherer [Installation des Community Servers](#), folgt in den meisten Fällen die Lokalisierung. Zm Glück gibt es bereits ein deutsches Language-Pack, deren Installation hier kurz erläutert werden soll.

Den Download des Language-Packs für die Version 2.1 findet man auf den offiziellen Seiten im Downloadbereich. Der direkt Link lautet:

<http://communityserver.org/files/folders/language>

Allerdings ist hierfür eine Registrierung notwendig.

In dem herunter geladenen Zip-Archiv findet man neben der `ReadMe.txt` den Order `de-DE`, in dem die Sprachdateien enthalten sind. Dieser Order muß per FTP in den Ordner `Languages` des Community Server Webs übertragen werden. Anschließend sollte man von folgenden Dateien ein Backup anlegen:

```
\Languages\languages.xml  
\communityserver.config  
web.config
```

Zunächst muß Datei `\Languages\languages.xml` abgeändert werden. Folgende Zeile muß in die `<root>` Section eingefügt werden:

```
<language name="German" key="de-de" />
```

Diese Zeile gibt an, daß nun eine weitere Sprache in dem Ordner `de-de` vorhanden ist.

Möchte man die Standardsprache des Community Server auf Deutsch festlegen, muß eine Zeile in der Datei `\communityserver.config` geändert werden. Das Attribut `defaultLanguage` des `Core Nodes` enthält per Default den Wert `en-US`. Dieser muß mit dem Wert `de-DE` ersetzt werden.

```
<Core defaultLanguage="de-DE"
```

Nun muß nur noch die Ausgabe der Uhrzeiten u. des Datums angepasst werden, da diese weiterhin im US Format ausgegeben werden. Hierfür ist es nötig die `Web.config` zu ändern. Falls folgende Zeile nicht bereits vorhanden ist, muß sie eingefügt, ansonsten abgeändert werden. Wichtig ist die Platzierung in der `<system.web>` Section.

```
<globalization culture = "de-DE"></globalization>
```

Nachdem die `Web.Config` gespeichert wurde, ist die Lokalisierung abgeschlossen und der Community Server sollte nun die deutsche Sprache verwenden.

5.13. [Community Server Installation HowTo](#)



Abbildung 39: Community Server Installation 1

Sucht man eine ASP.NET Foren-Software trifft man schnell auf dem Community Server, der von der Firma telligent entwickelt wurde. Die Feature Liste des Community Server ist lang. Nach der Installation erhält man nicht nur ein einfaches Forum, sondern vielmehr eine Kommunikations-Plattform mit zusätzlicher Unterstützung von WebLogs und Image Galleries, also alles was eine moderne Community Site ausmacht.

Neben der kostenlosen Express Edition, ist es möglich eine Standard, Professional oder Enterprise Version mit mehr Features zu erwerben. Eine genaue Übersicht der Versionen findet man unter der URL <https://store.telligent.com/FeatureMatrix.aspx#Edi...>

In diesem Artikel wird die Version 2.1.61025.2 des Community Servers für ASP.NET 2.0 per Web installiert. Grundsätzlich ist hierfür zunächst nur der Zugriff auf das Web per FTP nötig. Der Download des Community Server ist nach erfolgreicher Anmeldung unter der URL <http://www.communityserver.org> möglich. Neben der Installation per Web-Oberfläche ist es ebenfalls möglich (entsprechende Rechte vorausgesetzt) mit dem Windows Installer die zukünftige Community einzurichten.

Vorbereitung

Nachdem der Download erfolgreich war und das Zip-Archiv entpackt wurde, findet man zwei Ordner mit den Namen Web u. SqlScripts vor. In diesem Beispiel wird eine neue Community eingerichtet, somit kann der Inhalt des Ordners SqlScripts vernachlässigt werden. Aktualisiert man eine vorhandene Installation, findet man dort die entsprechenden Scripte für die Aktualisierung der Datenbank.

Zunächst muß in der Datei /web/installer/default.aspx die Variable `INSTALLER_ENABLED` auf `true` gesetzt werden. Diese Einstellung aktiviert die Installation per Web-Oberfläche.

```
bool INSTALLER_ENABLED = true;
```

Anschließend kann der Inhalt des Web-Ordners per FTP auf den gewünschten Server übertragen werden.

Zusätzlich zur Übertragung der Dateien per FTP, muß eine SQL-Server Datenbank eingerichtet werden. Während der Web-Installation werden der Datenbankname, der

Benutzername und das Passwort benötigt. Zusätzlich muß der Benutzer den Datenbank-Rollen `public` und `db_owner` zugeordnet werden.

Installation

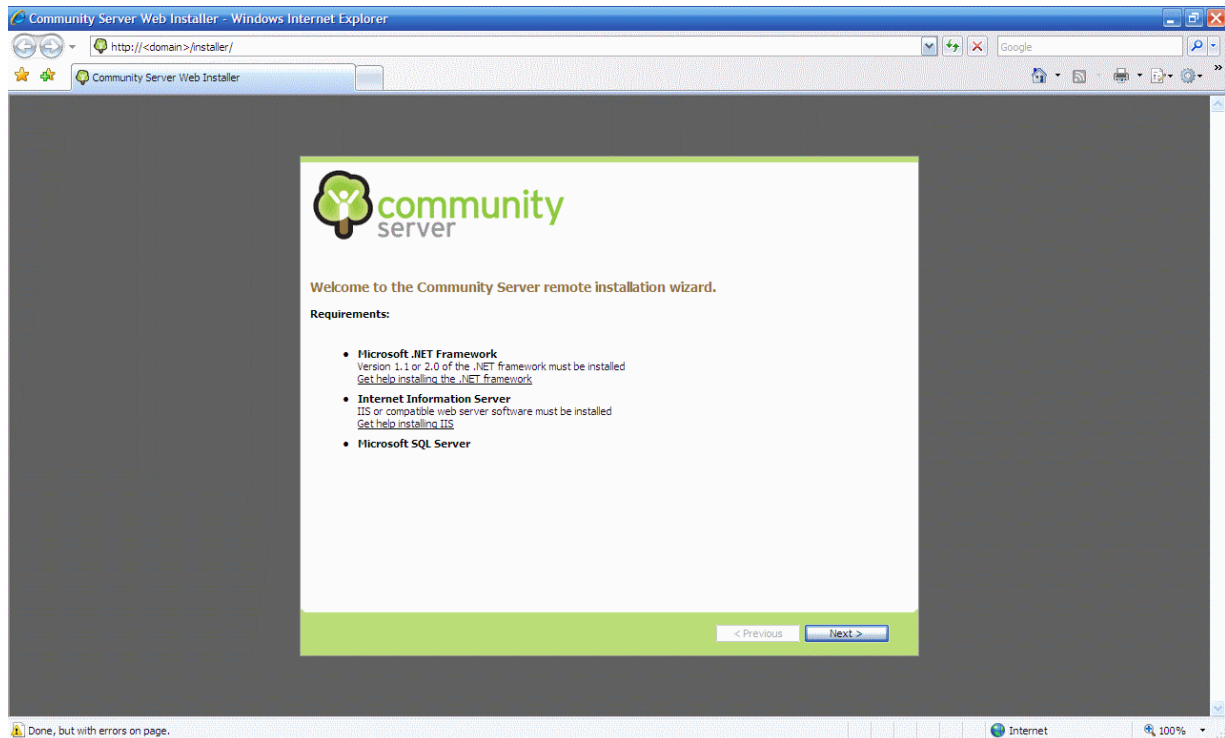


Abbildung 40: Community Server Installation 2

Der Aufruf der URL <http://<domain>/installer> startet den Web-Installer. Nachdem den Lizenzbestimmungen zugestimmt wurde, muß die Verbindung zur Datenbank angelegt werden. Hierfür wird wie oben bereits geschrieben, der Benutzername der Datenbank und das Passwort benötigt, außerdem muß festgelegt werden, ob sich die Anwendung per Windows oder Sql Server Authentication verbinden soll. Da in diesem Artikel ein Datenbank-Benutzer angelegt wurde, fällt die Wahl auf "Sql-Server Authentication".

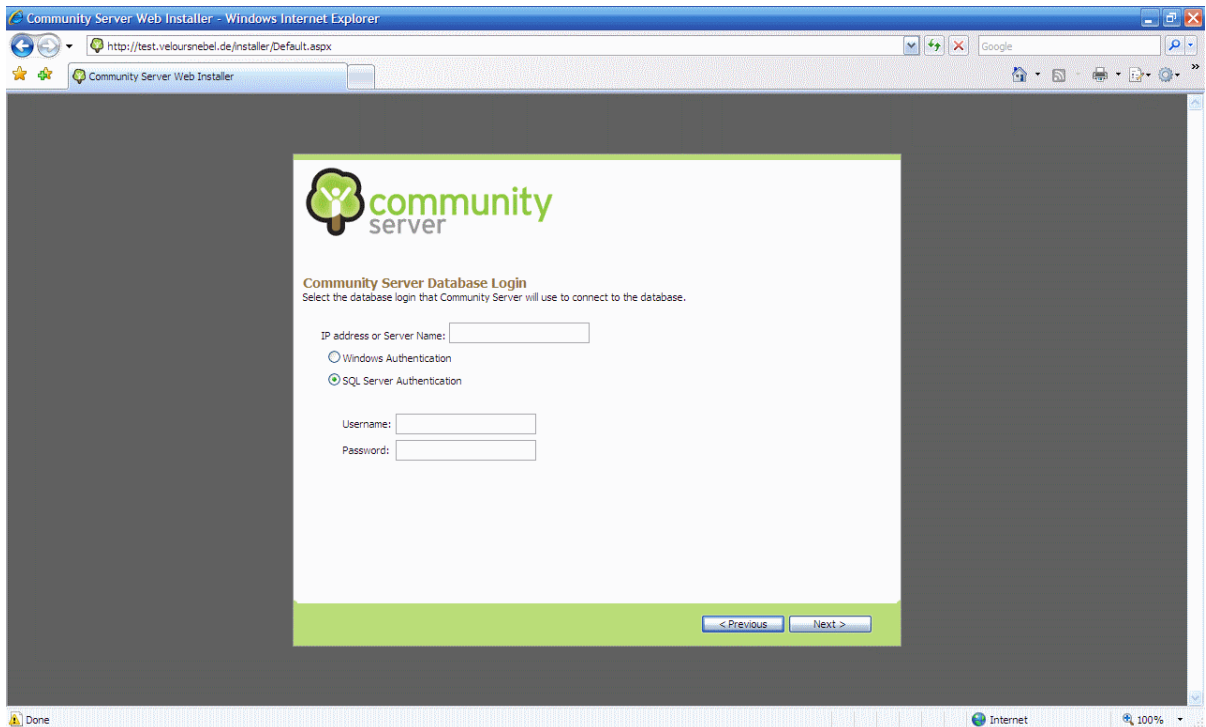


Abbildung 41: Community Server Installation 3

Der Installer versucht nun eine Verbindung mit der Datenbank herzustellen. War dies erfolgreich, kann die gewünschte Datenbank ausgewählt werden.

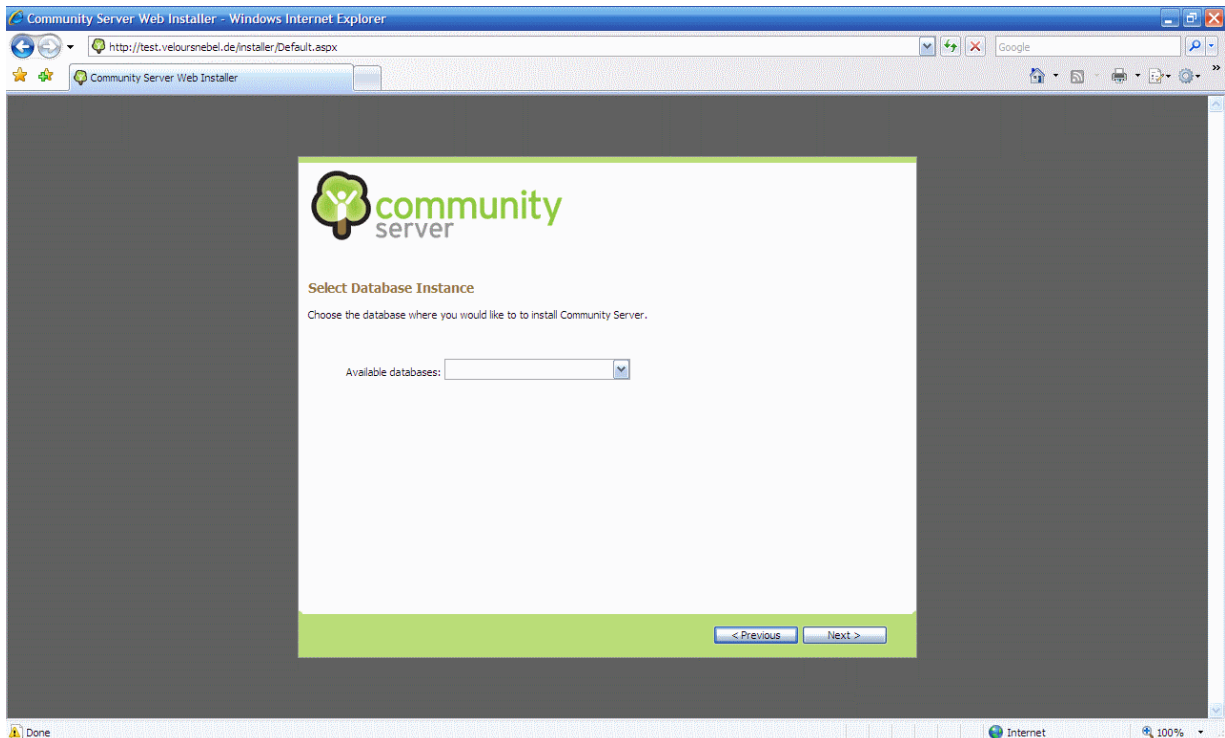


Abbildung 42: Community Server Installation 4

Nun können verschiedene Optionen ausgewählt werden, die die Einrichtung der Datenbank beeinflussen.

Script ASP.NET MemberRoles

Diese Option installiert die ASP.NET Memberroles. Da lediglich der Community Server die Datenbank verwendet, kann diese Option aktiviert bleiben. Nutzen mehrere Anwendungen die ASP.NET Memberroles, muß diese Option deaktiviert werden.

Script Community Server

Ist diese Option aktiviert, wird das Standard Community Server Schema installiert. Bei einer Neuinstallation wird diese Option natürlich benötigt.

Create Community

Auch diese Option wird benötigt, wenn eine Neuinstallation statt findet und installiert eine neue Community.

Im nächsten Schritt wird der erste Benutzer der Community angelegt, der gleichzeitig auch die Aufgabe des Administrators übernimmt. Neben der URL der Community, wird ein Username, ein Passwort, und die E-Mail Adresse für den zukünftigen Administrator benötigt. Außerdem wird in diesem Beispiel die Option *Create Sample Data* ausgewählt. Benötigt man die Beispieldaten nicht, kann man diese Option natürlich auch deaktivieren.

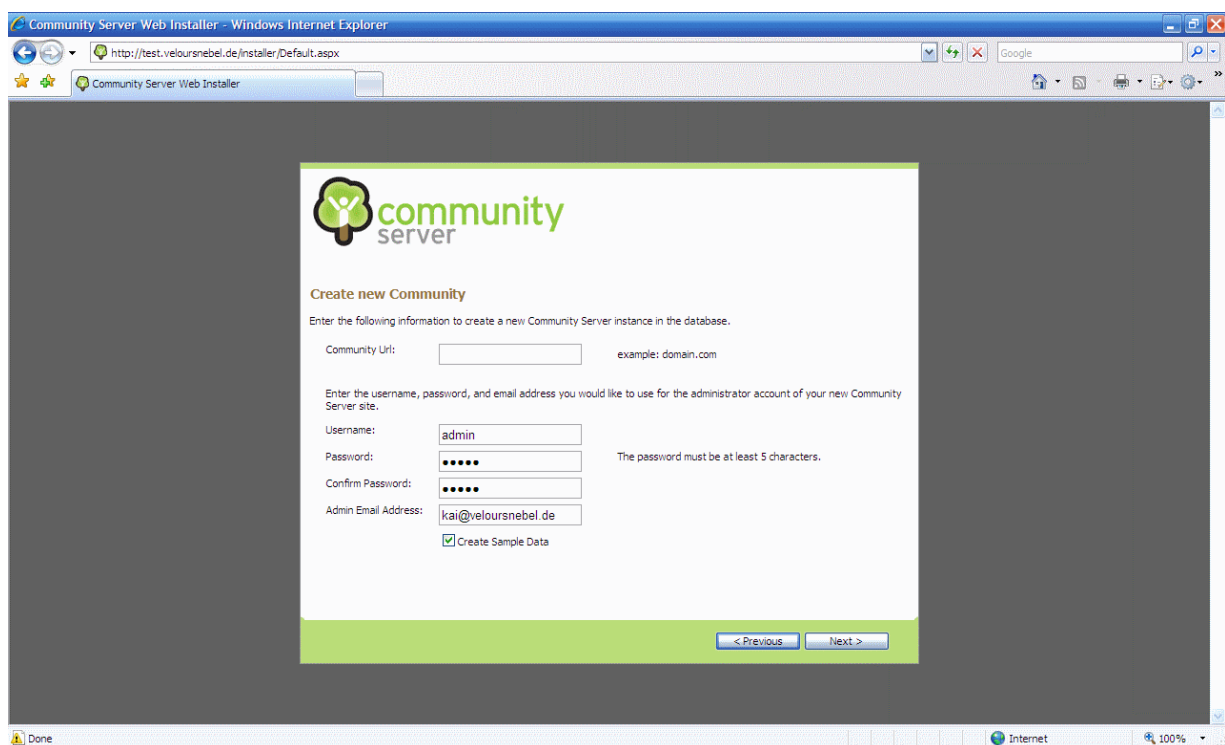


Abbildung 43: Community Server Installation 5

Nachdem der Installer fertig gearbeitet hat, ist die Installation abgeschlossen. Ist bereits eine Web.config in dem Web vorhanden zeigt der Installer die nötigen Informationen an, die dann per Copy & Paste nachgepflegt werden können. Aus Sicherheitsgründen sollte das Verzeichnis Installer anschließend gelöscht werden.

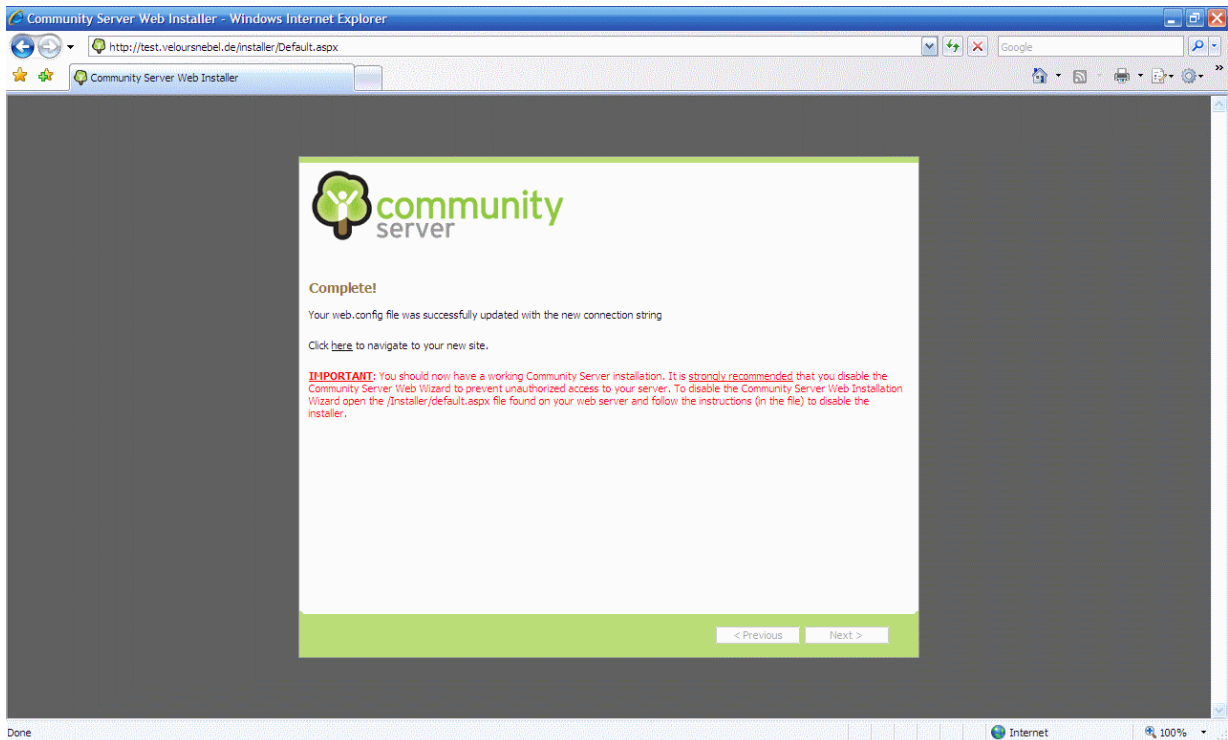


Abbildung 44: Community Server Installation 6

Installation abgeschlossen

Sofern bei der Installation die Option *Create Sample Data*, findet man nun einen bereits eingerichteten Blog, eine eingerichtete Image Gallery und natürlich das Forum mit Test-Einträgen vor. Nachdem man sich als Administrator eingeloggt hat ist es möglich über den Menüpunkt Control Panel, die weitere Konfiguration der Community vorzunehmen.

Weitere Informationen zur Installation und Konfiguration des Community Server findet man unter der URL <http://docs.communityserver.org/>. Außerdem ist ein weiteres deutsches Tutorial von [Alex Bierhaus](#) unter der URL

<http://www.codezone.de/DetailPage.Codezone?GUID=b6454095-59bf-474d-be09-cb68a126c424>

verfügbar.

Erfolgreich eingesetzte Installation findet man unter folgenden Adressen, insbesondere erwähnenswert ist das offizielle ASP.NET und xBox Forum von Microsoft.

www.hive.net
blogs.msdn.com
forums.asp.net
forums.xbox.com
www.glengamoi.com

5.14. [SQL Server Web Data Administrator](#)

Es gibt nicht nur den "ASP.Net EnterpriseManager[1]" um per Web-Frontend SQL Server Datenbank zu administrieren, sondern auch den "SQL Server Web Data Administrator[2]" von Microsoft.

[1] <http://www.aspenterprisemanager.com/>

[2] <http://www.microsoft.com/downloads/details.aspx?FamilyID=C039A798-C57A-419E-ACBC-2A332CB7F959&displaylang=en>

6. MICROSOFT OFFICE

6.1. [Word 2007: Custom Ribbons erstellen](#)

Im Beitrag [Word 2007 Winamp Controller](#) habe ich bereits ein Beispiel für Custom Panes gezeigt.

Dieser Beitrag zeigt nun, wie einfach ein Custom Ribbon (Ribbons werden die neuen Menüleisten unter Office 2007 genannt) erstellt werden kann. Für den Moment muss man sich mit dem Sourcecode zufrieden geben. Entsprechende Tutorials sind in Arbeit und werden demnächst veröffentlicht.

Soviel sei verraten: Ein wenig [Windows Presentation Foundation](#) und schon kann es losgehen.

Das Endergebnis des Beispiels [1] sieht folgendermaßen aus:

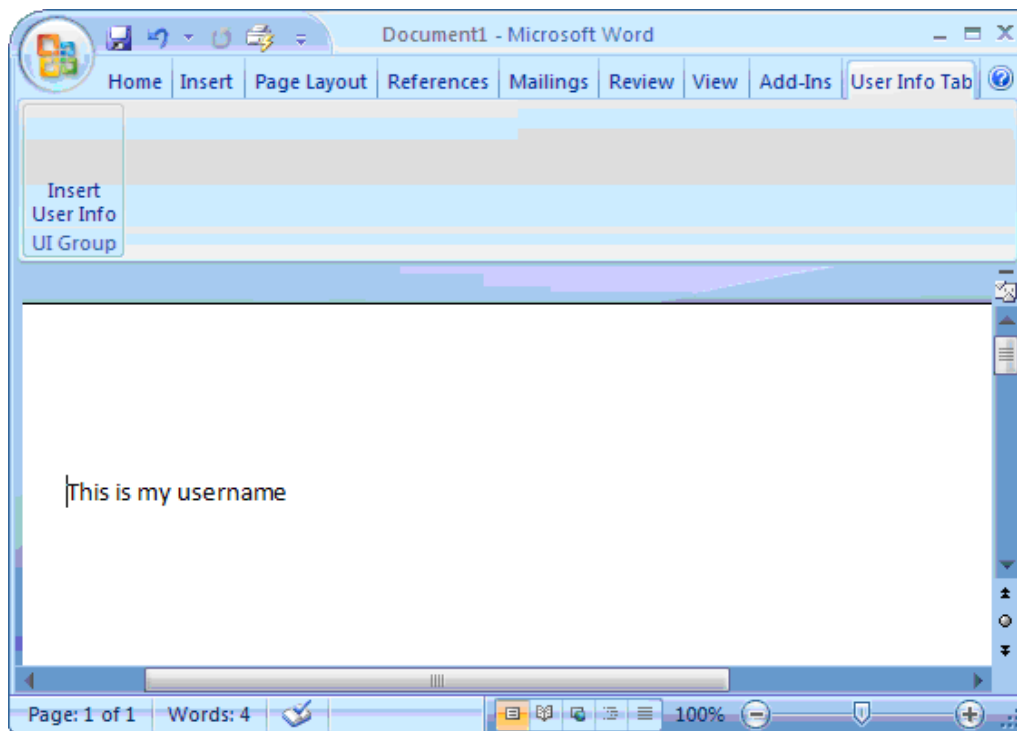


Abbildung 45: Word 2007 - Custom Ribbons erstellen

Das Beispiel liegt in einer Visual Studio 2005 Solution vor und wurde in C# erstellt.

Zu diesem Thema steht bereits ein Tutorial [2] zur Verfügung.

[1] [Download Word 2007 Custom Ribbon Beispiel](#)

[2] [Zur Tutorials-Page](#)

6.2. [MS Outlook - Makros reloaded](#)

So, nachdem es gestern den "Mail mittles Shortcut als unread markieren"-Tag gegeben hat, hab ich noch ein kleines Makro gebaut, welches mir mittels Shortcut den "Junk E-Mail"-Folder und auch gleich den "Gelöschte Objekte"-Order leert. Damit ist's per Tastendruck sauber im Outlook ;-)

Hier nun das Makro:

```
Sub RemoveJunkAndDeleted()  
  
Dim mItem As MailItem  
Dim mNamespace As NameSpace  
Dim junkFolder As MAPIFolder  
Dim deletedFolder As MAPIFolder  
  
Set mNamespace = Application.GetNamespace("MAPI")  
  
Set junkFolder = mNamespace.GetDefaultFolder(olFolderJunk)  
  
For Each mItem In junkFolder.Items  
mItem.Delete  
Next  
  
Set deletedFolder =  
mNamespace.GetDefaultFolder(olFolderDeletedItems)  
  
For Each mItem In deletedFolder.Items  
mItem.Delete  
Next  
  
End Sub
```

Dieses Makro ist wieder wie gestern im Beitrag [1] einzubinden. Als Shortcut habe ich ein k vergeben, da dies noch nicht benutzt wird.

PS: Den Junk-E-Mail-Folder immer schon sauber halten. Denn Outlook archiviert diesen Ordner standardmäßig mit, was ich zwar nicht verstehe, ist aber so. Einfach die Folder-Eigenschaften öffnen und die Archivierung deaktivieren.

[1] [MS Outlook: Mails mit Shortcut als gelesen markieren](#)

6.3. [MS Outlook - Makros reloaded 2](#)

Vor einiger Zeit habe ich ein kleines VBA-Makro veröffentlicht, mit dem es einfach möglich ist, die Ordner "Junk E-Mail" und "Gelöschte Objekte" zu leeren (siehe [MS Outlook - Makros reloaded](#)).

[Alex Bierhaus](#) hat mich nun auf einen Fehler dieses Makros hingewiesen, den ich in der Hitze des Gefechtes übersehen hatte:

In den gelöschten Objekten befinden sich natürlich nicht nur gelöschte Emails, sondern auch gelöschte Kontakte, Notizen, Aufgaben etc. Hierbei wurde ein Fehler ausgelöst und die

Objekte wurden nicht gelöscht. Hier nun eine marginal geänderte Variante, die nun einwandfrei funktionieren sollte.

```
Sub RemoveJunkAndDeleted()  
  
Dim mItem As Object  
Dim mNamespace As NameSpace  
Dim junkFolder As MAPIFolder  
Dim deletedFolder As MAPIFolder  
  
Set mNamespace = Application.GetNamespace("MAPI")  
  
Set junkFolder = mNamespace.GetDefaultFolder(olFolderJunk)  
  
For Each mItem In junkFolder.Items  
mItem.Delete  
Next  
  
Set deletedFolder =  
mNamespace.GetDefaultFolder(olFolderDeletedItems)  
  
For Each mItem In deletedFolder.Items  
mItem.Delete  
Next  
  
End Sub
```

Getestet wurde mit gelöschten Emails, Kontakten, Aufgaben und Notizen unter Verwendung von Microsoft Office 2003.

6.4. [MS Outlook: Mails mit Shortcut als gelesen markieren](#)

Das 'gelesen markieren' funktioniert unter Outlook nur, wenn man zuvor die entsprechenden Mails markiert und dann STRG-Q drückt. Manchmal möchte man jedoch alle Mails in einem ausgewählten Folder als gelesen markieren - und zwar mit Hilfe eines Shortcuts.

Hier gibt es die Info wie das funktioniert.

Zuerst ist folgendes Makro anzulegen:

```
Sub SetAllRead()  
Dim mItem As MailItem  
If Outlook.ActiveExplorer.CurrentFolder.UnReadItemCount > 0 Then  
For Each mItem In Outlook.ActiveExplorer.CurrentFolder.Items  
mItem.UnRead = False  
Next  
End If  
End Sub
```

In meinem Beispiel nennt sich das Makro SetAllRead. Nach dem Erstellen ist das Menü Extras/Anpassen zu wählen. In der Lasche Befehle nun einfach den Eintrag Makros in der

linken Liste auswählen. Das gerade erstellte Makro sollte in der rechten Liste verfügbar sein. Mit der linken Maustaste einfach das Makro anfassen und in eine Toolbar ziehen.

Jetzt kommt der entscheidende Teil bezüglich Shortcut: Mit einer rechten Maustaste, auf die eben erstellte Schaltfläche (Achtung, das Anpassen-Fenster muss noch geöffnet sein) erscheint ein Menü, in dem ein Name für die Schaltfläche vergeben werden kann. Hier ist ein Name zu wählen, in dem ein Buchstabe mit einem kaufmännischen Und (&) hervorzuheben ist. Dieser wird dann als Shortcut verwendet. Hier sollte ein Buchstabe gewählt werden, für den noch kein Shortcut vergeben ist. Ich denke, ein m würde sich hier anbieten.

Ab sofort kann man einzelne Mailfolder auswählen, ALT+M drücken und schon werden die darin enthaltenen MailItems als gelesen markiert.

7. DATENBANK- MANAGEMENT-SYSTEME

7.1. [SQL Server 2000 Replikation und Error 18483](#)

Die Replikation läßt sich nicht einrichten, da der Benutzer 'distributor_admin' nicht als Remotebenutzername eingetragen ist. Und wo genau liegt nun das Problem? Die Antwort ist eigentlich ganz einfach:

Das Problem liegt daran, dass der Servername des SQL Servers nicht mit der eingetragenen ServerName-Property überein stimmt. Vermutlich durch eine Umbenennung etc. Diese Einstellungen kann man mit folgender Abfrage herausfinden:

```
SELECT @@SERVERNAME, SERVERPROPERTY('ServerName')
```

Die beiden Felder des Resultates müssen hier gleich sein. Wenn nicht, dann ist folgendes Script auszuführen:

```
USE master
GO

DECLARE @serverproperty_servername varchar(100),
        @servername varchar(100)

SELECT @serverproperty_servername = CONVERT(varchar(100),
        SERVERPROPERTY('ServerName'))

SELECT @servername = CONVERT(varchar(100), @@SERVERNAME)

EXEC sp_dropserver @server=@servername, @droplogins='droplogins'

EXEC sp_addserver @server=@serverproperty_servername,
        @local='local'
```

Damit wird die Serverregistrierung gelöscht und neu gesetzt, mit dem Wert aus dem Feld `SERVERPROPERTY('ServerName')`. Wird ein Server umbenannt, betrifft dies nur diese Eigenschaft, der eigentliche Servername bleibt jedoch davon unberührt. Weiters sollte die Stored Procedure `sp_dropserver` mit `@droplogins` aufgerufen werden, um etwaige vorhandene Remotelogins ebenfalls zu löschen. Andernfalls kann der Server nicht neu registriert werden.

Nach diesen Schritten ist der SQL Server neu zu starten und es kann nun versucht werden, die Replikation zu konfigurieren. Nun sollte es funktionieren.

Der ursprüngliche Tipp ist auf der Microsoft-Support-Seite [1] zu finden, jedoch ohne den `@droplogins` Hinweis.

[1] [Replication setup is not successful when SQL Server is deployed by using a disk image](#)

7.2. [SQL Server 2005: Output-Klausel](#)

Eines der neuen Features des SQL Server 2005 ist die OUTPUT-Klausel, die Transact-SQL um eine nützliche Funktion erweitert.

Durch die Angabe der OUTPUT-Klausel können in INSERT, UPDATE und DELETE Anweisungen, die betroffenen Datensätze, oder Teile davon (beispielsweise die Primary-Key-Werte) zurückgegeben werden.

Dies ist beispielsweise bei einem INSERT sehr interessant, wenn die vergebene ID im weiteren Ablauf benötigt wurde. Diese konnte bisher via SELECT @@Identity bezogen werden. Danach musste ein weiteres SELECT-Statement abgesetzt werden, um den Datensatz zu erhalten. Durch die OUTPUT-Klausel ist dies nicht mehr notwendig. Die Ergebnisse werden sofort zurückgeliefert, sofern gewünscht.

Beispiel

```
USE AdventureWorks;
GO
DELETE tUser
OUTPUT deleted.*
WHERE ID = 7;
GO
```

Dieses Beispiel löscht den Datensatz mit der ID 7 aus der Tabelle tUser und gibt den gesamten Datensatz zurück. Es können jedoch auch nur einzelne Felder zurückgeliefert werden:

```
USE AdventureWorks;
GO
DELETE tUser
OUTPUT deleted.ID, deleted.Firstname, deleted.Lastname
WHERE ID = 7;
GO
```

Hier würden nur die ID, der Vorname und der Nachname als ResultSet zurückgegeben werden.

Wie kann dies unter dem .NET Framework benutzt werden? Es verhält sich sehr einfach. Die einzelnen Statements werden mit einem Command-Objekt (beispielsweise SqlCommand) abgesetzt. Im Falle von INSERT, UPDATE bzw. DELETE Abfragen wird dazu die Methode ExecuteNonQuery verwendet. Wird nun die OUTPUT-Klausel verwendet, ist anstatt der Methode ExecuteNonQuery beispielsweise die Methode ExecuteReader zu verwenden. Mit dem zurückgegebenen DataReader kann durch die einzelnen Datensätze iteriert werden.

Referenzen

[1] [MSDN](#)

7.3. Objekte und relationale Daten in einer Datenbank

Auf die Beiträge von [Thomas](#) und [meiner Wenigkeit](#) hat sich auch [Alex](#) zu Wort gemeldet. Sein Ansatz (angelehnt an den Microsoft SQL Server 2005) verfolgt das Ablegen von Objekten und relationalen Daten in einer einzigen DB.

Auch hierzu ein paar Gedanken von mir:

Prinzipiell gibt es den Ansatz der ORDBMS (Objektrelationale Datenbank Management Systeme). Dabei wird ein relationales System um objektorientierte Fähigkeiten erweitert. Oft wird auch nur eine objektorientierte Zugriffsschicht darübergelegt. In der Tat werden allerdings keine Objekte mit relationalen Daten vermischt.

Wie würde das Speichern von Objekten zusammen mit relationalen Daten in einer Datenbank in der Praxis aussehen?

Prinzipiell stehe ich dem Vermischen von Objekten mit relationalen Daten eher negativ gegenüber. Der Ansatz von XML-Feldern in einer Datenbank (siehe SQL Server 2005) ist durchaus praktisch und in manchen Fällen auch sinnvoll. Dies jedoch zu nutzen, um Objekte abzulegen macht eher weniger Sinn. Aus folgenden Gründen (wenn ich von Serialisierung spreche, beziehe ich mich auf die XML-Serialisierung):

Aufwand der Serialisierung. Bedingt durch diesen Aufwand können die Daten ohnehin oder in eine relationale Struktur gequetscht werden. Performancemässig wird es hier (allerdings hab ich das jetzt nicht getestet) nicht sehr viel Unterschied geben.

Weiters verleitet dieser Ansatz dazu, eine Tabelle mit mehreren XML-Feldern zu erstellen und darin serialisierte Objekte abzulegen. Eventuell noch von unterschiedlichen Typen. Spätestens dieser Punkt würde durchaus Probleme aufwerfen.

Durch das einfache serialisierte Ablegen würden in der Datenbank Referenzen nicht mehr ersichtlich sein. In einem reinen relationalen oder reinen objektorientierten System bleibt dieser Vorteil erhalten. Referenzen in XML sind zwar möglich, aber selbst bei einem eigenen XML-Serializer nützt das Einfügen dieser Referenzen wenig, wenn sie durch die Datenbank nicht dargestellt werden können.

Abfragen sind natürlich auch in dieser Form noch auf der Datenbank möglich ohne alle Daten zu laden, aber durch die Vermischung von SQL und XPATH etc. würde ich meinen, dass der Performance-Vorteil eher gering ausfällt -> dürfte wohl im Minusbereich angesiedelt sein.

Und zu guter Letzt: Diese Variante würde kein gemeinsames Ablegen von Objekten und relationalen Daten implizieren. Das einzige was Sinn machen würde, wäre das Ablegen von relationalen Informationen (Metdaten etc.) zu Objekten - aber ich denke das übernimmt ohnehin die Indizierung für uns.

Soweit mal meine ersten Gedanken dazu. Weitere werden vermutlich noch folgen, wenn ich allen "neuen" Gedanken nachgegangen bin.

7.4. [SQL Server 2005: Erstellung einer Replikation](#)

Wer eine Replikation unter dem SQL Server 2005 erstellen muss bzw. will, der findet im nachfolgenden Tutorial eine Anleitung. Alle notwendigen Schritte werden anhand von Screenshots und Beschreibungen aufgezeigt und sollten recht einfach nach zu vollziehen sein.

[Datenreplizierung unter dem SQL Server 2005](#) (1.1 MB)

7.5. [SQL Server 2005: Managed Code](#)

Dieses Tutorial soll zeigen, wie managed Code (in diesem Fall wird C# benutzt) unter dem SQL Server 2005 verwendet werden kann.

[Managed Code unter dem SQL Server 2005](#) (213 KB)

7.6. [SQL Server 2005: Custom Datatypes](#)

Eine Einführung in die benutzerdefinierten Datentypen unter dem SQL Server 2005 bietet das unten verlinkte Dokument.

[Datentypen unter dem SQL Server 2005](#) (338 KB)

7.7. [C# und SQL Server 2005](#)

Immer wieder stoße ich auf Anfragen, wie denn genau eine Anbindung an den SQL Server 2005 mit C# (oder VB.NET) funktioniert. Grundsätzlich gibt es dazu zahlreiche Tutorials im Internet, die meisten jedoch auf Englisch. Diese Tatsache scheint dann doch sehr viele abzuschrecken. Daher habe ich mich entschlossen, ein Tutorial zu diesem Thema zu verfassen, um eine entsprechende deutschsprachige Ressource bereitstellen zu können.

Inhalt

1. Einführung
2. Verbindung herstellen
3. Daten abfragen
4. Daten hinzufügen
5. Zusammenfassung

1. Einführung

Für den Zugriff auf den SQL Server (2000 oder 2005 ist hierbei unerheblich) werden vom .NET Framework alle notwendigen Funktionen zur Verfügung gestellt. Diese verstecken sich im Namespace System.Data.SqlClient. Für den SQL Server könnten zwar auch die Klassen aus dem Namespace System.Data.OleDb verwendet werden, jedoch ist der DataProvider unter

SqlClient auf den SQL Server optimiert und sollte (ausser andere triftige Gründe sprechen dagegen) verwendet werden.

2. Verbindung herstellen

Nun, starten wir damit, eine Verbindung zum SQL Server herzustellen. Dazu verwenden wir die Klasse `SqlConnection` aus dem oben angegebenen Namespace. Der Klasse `SqlConnection` muss ein `ConnectionString` übergeben werden. Dieser definiert wo der SQL Server zu finden ist, welche Datenbank verwendet werden soll und wie die User-Informationen lauten. Beispielsweise würde ein `ConnectionString` wie folgt aussehen:

```
Data Source=Aron1;Initial Catalog=pubs;User  
Id=sa;Password=asdasd;
```

Die Data Source stellt den Computer dar, auf dem der SQL Server läuft. Zu achten ist hierbei, dass der SQL Server 2005 mit einem Instanznamen angelegt wird, welcher im `ConnectionString` entsprechend anzugeben ist, da eine Verbindung sonst nicht aufgebaut werden kann. Die Einstellung `Initial Catalog` beschreibt den Namen der Datenbank auf welche verbunden werden soll. `User Id` ist der zu verwendende Username und `Password` sollte selbstsprechend sein. Weitere Informationen zu den `ConnectionStrings` sind unter [1] zu finden.

In C# sieht ein Verbindungsaufbau nun wie folgt aus:

```
SqlConnection conn = new SqlConnection(@"Data  
Source=COMPUTERNAME\SQLEXPRESS;Initial Catalog=DatabaseDemo;User  
Id=sa;Password=MyPassword;");  
conn.Open();  
  
conn.Close();
```

In diesem Fall ist ein SQL Server 2005 Express installiert. `COMPUTERNAME` stellt den Namen des Computers dar und `SQLEXPRESS` ist der Instanzname des SQL Servers. Dieser Name wird im Normalfall bei der Installation eingegeben. Ist nicht sicher wie dieser lautet, dann kann dies in der Dienste-Liste ausgelesen werden. Dazu einfach die Dienste anzeigen lassen, den SQL Server Dienst suchen, in Klammer dahinter steht der entsprechende Instanzname.

Durch den obigen C# Code kann man sich nun zur Datenbank verbinden. Für das Öffnen der Verbindung ist die Methode `Open()` zu verwenden. `Close()` sorgt für das Schließen der Verbindung.

3. Daten abfragen

Da wir nun eine Verbindung aufbauen können, erfolgt der nächste Schritt: Daten abzufragen. Hierfür werden die Klassen `SqlCommand` und `SqlDataReader` verwendet.

Folgender Code übernimmt das für uns:

```
SqlCommand com = new SqlCommand("SELECT * FROM tPerson", conn);  
  
SqlDataReader reader = com.ExecuteReader();  
while (reader.Read())
```

```
{  
    Console.WriteLine("ID: {0}, Firstname: {1}, Lastname: {2}",  
        reader[0].ToString(), reader[1].ToString(),  
        reader[2].ToString());  
}  
reader.Close();
```

Hier passiert nun folgendes: Zuerst erstellen wir einen `SqlCommand`, der auf der Datenbank ausgeführt werden soll. Als Parameter übergeben wir ein SQL-Statement und die geöffnete Verbindung `conn`.

Danach instanzieren wir einen `SqlDataReader`. Dieser ist für das Auslesen der Daten zuständig. Der Befehl `com.ExecuteReader()` sorgt dafür, dass der Command ausgeführt wird und ein `DataReader` zurückgegeben wird. Mit dem `DataReader` kann Datensatz für Datensatz durch das Ergebnis iteriert werden, was wir auch mit Hilfe der `while`-Schleife tun. Nachdem die Daten bezogen wurden, ist der `DataReader` wieder zu schließen. Darin geben wir die Daten lediglich auf der Console aus. Die Ausgabe würde wie folgt aussehen:

```
ID: 1, Firstname: Norbert, Lastname: Eder  
ID: 2, Firstname: Test, Lastname: Person  
ID: 3, Firstname: Franz, Lastname: Nachname
```

Es empfiehlt sich, auch die weiteren Möglichkeiten der Klasse `SqlCommand` im MSDN [2] nachzuschlagen, da viele weitere Funktionen zur Verfügung stehen. Alle zu beschreiben würde jedoch den Umfang dieses Tutorials sprengen.

4. Daten hinzufügen

Da wir nun eine der Möglichkeiten gesehen haben wie Daten abgefragt werden können, werden wir nun einen Datensatz hinzufügen. Dies passiert wie folgt:

```
SqlCommand com = new SqlCommand("INSERT INTO tPerson (Firstname,  
    Lastname) VALUES (@firstname, @lastname)", conn);  
com.Parameters.Add(new SqlParameter("@firstname", "fritz"));  
com.Parameters.Add(new SqlParameter("@lastname", "huber"));  
com.ExecuteNonQuery();
```

Wieder erstellen wir einen `SqlCommand`, der ein SQL Statement übergeben bekommt sowie die Datenbankverbindung. Hier passiert dann noch etwas Spezielles. Aus Sicherheitsgründen werden die Daten nicht direkt ins SQL Statement geschrieben, sondern via `SqlParameter`. Dieser Schritt hat einige Vorteile was das Thema SQL Injection etc. betrifft. Sollte also immer in dieser Form angewandt werden. Mittels der `ExecuteNonQuery`-Methode des `SqlCommand`-Objektes wird der Befehl auf der Datenbank ausgeführt und der Datensatz eingefügt.

4. Zusammenfassung

Dies war ein kurzer Einblick in die Datenbank-Thematik unter C#. Alle Klassen und Methoden können auf die gleiche Art und Weise unter VB.NET verwendet werden. Es empfiehlt sich, die angegebenen Klassen im MSDN genauer anzusehen, sowie auch die vorhandenen Beispiele zu den einzelnen Methoden durchzuarbeiten. Danach sollte dieses Thema kein Problem mehr darstellen.

[1] <http://www.connectionstrings.com>

[2] <http://msdn2.microsoft.com> oder <http://msdn.microsoft.com>

7.8. OODBMS- Object Oriented DataBase Management Systems

Bezugnehmend auf den Eintrag von [Thomas](#) möchte ich hier noch ein paar Worte zu OODBMS verwenden.

Objektorientierte Datenbanken sind schon sehr nett, aber leider auch nicht für alles einsetzbar. Das schöne daran ist, dass die Objekte an sich gespeichert werden, nicht ein relationales Mapping davon. Dies bedeutet, dass das Abbilden von relationalen Daten auf das ursprüngliche Objekt nicht mehr notwendig ist.

Allerdings sehe ich hier doch noch einige Probleme, die nicht unausgesprochen bleiben sollten:

1. Fehlende Standards

Einheitliche Standards zu ORDBMS bzw. OODBMS gibt es nicht - auch wenn es diese Systeme nicht erst seit gestern gibt. Dadurch ist es nicht gewährleistet, dass alle Systeme auf gleiche Art und Weise arbeiten. Object-oriented SQL (OSQL) ist ansich recht nett, aber auch hier gibts die gleichen "Fehler" wie beim herkömmlichen SQL -> Jeder implementiert eine eigene Variante.

2. Backup, Restore, Verteilung etc.

Objekte sind wesentlich komplexer als relationale Strukturen. Entsprechend komplexer wird dadurch auch ein Backup, ein Restore, eine Verteilung oder überhaupt nur die Aktualisierung der Daten, das Nachziehen von Referenzen usw. Diese Dinge sind mit hohem Aufwand verbunden, daher ist eine OODBMS in diesen Gebieten sicherlich langsamer als ein herkömmliches RDBMS.

3. Angst?

Das Thema OODBMS ist ansich genau so alt, wie es objektorientierte Sprachen sind. Letztere haben sich in einigen Bereichen (Enterprise etc.) durchgesetzt. Erstere nicht. Warum? Nun, relationale Datenbanken sind einfacher zu handhaben und wurden über lange Zeit erprobt. Viele haben davor Angst, auf eine "neue" Technologie zu setzen (Anm.: die jedoch nicht viel jünger ist). Dadurch fließt auch nicht soviel Entwicklungsarbeit in objekt-orientierte Ansätze bei Datenbank-Management-Systemen.

4. Fehlende Unterstützung?

Viele Lösungen entstehen und verschwinden entsprechend auch wieder, da die Nutzung nicht vorhanden ist. Ich persönlich kenne nur ein OODBMS welches sich lange gehalten hat. Caché [1]. Alle anderen sind entweder "gemeinnützige" Projekte mit mehr oder weniger geringem Support.

Natürlich gibt es da noch andere Punkte. Der Vollständigkeit halber stehen dem jedoch auch sehr viele Vorteile gegenüber - keine Frage. Allerdings sehe ich hier db4o [2] nicht unbedingt das als Gelbe vom Ei an. Aber der Weg den db4o beschreitet ist prinzipiell nicht so schlecht.

[1] <http://www.intersystems.com/cache/>

[2] <http://www.db4o.com/>

7.9. [SA User unter SQLServer 2005 umbenennen](#)

Unter dem Microsoft SQL Server 2000 ist es ja nicht möglich, den sa User umzubenennen bzw. zu deaktivieren. Der SQL Server 2005 bietet allerdings diese Möglichkeit. Durchgeführt können diese beiden Aktionen mit ALTER LOGIN werden.

```
ALTER LOGIN sa DISABLE;  
ALTER LOGIN sa WITH NAME = [sys-admin];
```

7.10. [SQL Server 2000: Felder zur Replikation hinzufügen](#)

In manchen Fällen ist es notwendig, zu einer Datenbank neue Felder hinzuzufügen. Wird nun diese Datenbank auf einen weiteren Server repliziert ergeben sich hier die einen oder anderen Probleme.

Die beste Erfahrung habe ich mit der Stored Procedure `sp_repladdcolumn` gemacht. Mit Hilfe dieser Stored Procedure wird das Feld in der zu replizierenden Datenbank (Publication) eingetragen und zur Replikation hinzugefügt.

Wichtig hierbei ist, dass die Replikation vor der Änderung gestoppt werden muss.

```
exec sp_repladdcolumn  
@source_object = 'Tabellename',  
@column = 'Spaltenname',  
@typetext = 'float NULL',  
@publication_to_add = 'all'
```

Danach muss der Snapshot erneut erstellt und nach Abschluss dessen die Replizierung wieder gestartet werden.

Mit der Stored Procedure `sp_addarticle` kann übrigens eine neue Tabelle erstellt werden, die ebenfalls zur Replizierung hinzugefügt wird.

7.11. [SQL Server 2000: Einschränkungen bei Replikation](#)

Bei der Replikation unter dem Microsoft SQL Server 2000 gibt es eine vorkonfigurierte Einschränkung bei text-, ntext- und image-Feldern. Die Feldgröße bei der Replizierung ist auf 65536 Zeichen beschränkt. Wird nun in ein Feld ein größerer Wert eingefügt, kommt es zum folgenden Fehler:

Englisch:

Length of text, ntext, or image data (x) to be replicated exceeds configured maximum 65536.

Deutsch:

Länge der zu replizierenden text-, ntext- oder image-Daten übersteigt das konfigurierte Maximum 65536.

Dies kann mit folgender Stored Procedure umgangen werden:

```
CREATE PROC usp_configure_maxReplSize
    @NewSize int = 1000000
AS
    exec sp_configure 'max text repl size'
    exec sp_configure 'max text repl size', @NewSize
    RECONFIGURE WITH OVERRIDE
    exec sp_configure 'max text repl size'
GO
```

In diesem Fall wird die maximale Größe auf eine Million Zeichen gestellt.

7.12. [SQL Server Replizierung: Alt oder doch neu?](#)

Wer sich derzeit mit den Gedanken über eine SQL Server Replizierung spielt und sich nicht sicher ist, ob er den altbewährten SQL Server 2000 oder dann doch den neuen 2005er nehmen soll, dann kann ihm geholfen werden.

Werden in Zukunft Änderungen an Tabellen etc. vorgenommen?

Wenn nein, dann sehe ich kein Problem bei der Verwendung der 2000er-Version. Handelt es sich dabei allerdings um ein wachsendes System, d.h. die Struktur der Tabellen etc. kann bzw. wird sich verändern, dann rate ich eindeutig zum 2005er.

Der Grund liegt darin, dass Änderungen an der Tabellenstruktur im SQL Server 2000 nur schwer nachgezogen werden kann. Wird beispielsweise im Verteiler (der Server, der die Daten an seine Abonnenten verteilt) eine neue Tabelle angelegt, muss die Publikation, als auch die Abonnenten am Verteiler gelöscht und neu erstellt werden. Ändern sich nur einzelne Datenfelder, können diese nur mit der Stored Procedure `sp_repladdcolumn` zur

Replikation hinzugefügt werden. Dies ist auf Dauer doch recht mühsam. Der SQL Server 2005 bietet diese Einschränkungen nicht mehr.

7.13. [Autoinkrement-Wert nach INSERT INTO auslesen, die Zweite](#)

Im letzten Beitrag zu diesem Thema zeigte ich kurz auf wie es möglich ist, einen inkrementellen Identitätswert nach einem INSERT auszulesen. Dazu verwendete ich die Konstante @@IDENTITY.

Allerdings ist die Verwendung dieser Konstante nicht immer zu empfehlen, da @@IDENTITY unabhängig des aktuellen Gültigkeitsbereiches arbeitet. Verwendet man z.B. einen Trigger, der nach dem eigentlichen INSERT ein weiteres auf eine andere Tabelle absetzt, erhält man den letzten Identitätswert. In diesem Fall also den Wert des zweiten INSERT.

Mit der Verwendung von SCOPE_IDENTITY() umgeht man dieses Problem, da diese Funktion in abhängigkeit des aktuellen Gültigkeitsbereichs arbeitet.

```
CREATE PROCEDURE [sp_Test]
    (@name [varchar](300))
AS
    SET NOCOUNT ON
    INSERT INTO [Tabelle1]([name]) VALUES (@name)
    SELECT SCOPE_IDENTITY()
    SET NOCOUNT OFF
    INSERT INTO [Tabelle2] (Tabelle1_ID,[name]) VALUES
    (SCOPE_IDENTITY(),@name)
GO
```

Die Anwendung ist ähnlich einfach.

Vielen Dank an [Frank Kalis](#), [Floyd](#) und [Tanja](#) für die Hinweise

Jedem, der mehr zum Thema SQL und SQL-Server wissen möchte, kann ich die Internetseite [InsideSQL.de](#) empfehlen. .

7.14. [Zufälligen Datensatz ausgeben](#)

Möchte man einen zufälligen Datensatz aus einer Datenbank-Tabelle anzeigen, kann man dies aufwendig per C# Code lösen. Allerdings gibt es eine viel einfachere Methode direkt per SQL.

```
SELECT TOP 1 Feld FROM Tabelle ORDER BY NEWID()
```

Die Anweisung NEWID() generiert einen eindeutigen Wert vom Typ uniqueidentifier. In Verbindung mit einem ORDER BY erhält man nun einen "zufälligen" Datensatz.

Dieses Vorgehen funktioniert ebenfalls unter MySQL, allerdings mit einer etwas abgewandelten Syntax.

```
SELECT Feld FROM Tabelle ORDER BY RAND() LIMIT 1
```

7.15. Autoinkrement-Wert nach INSERT INTO auslesen

Häufig verwendet man in Tabellen einer Datenbank einen inkrementellen ID-Wert um Datensätze eindeutig zu identifizieren. Möchte man nun diesen ID Wert nach dem INSERT direkt weiterverwenden, drängt sich im ersten Moment die Lösung auf, direkt nach dem INSERT ein SELECT auf die höchste ID abzusetzen und den Wert somit auszulesen.

Verwendet man diese Lösung, läuft man allerdings Gefahr eine falsche ID zu bekommen. Es ist durchaus möglich, dass zwischen dem INSERT und dem SELECT einer Session, bereits ein weiterer Datensatz einer anderen Session eingefügt wurde.

In diesem Beispiel verwende ich eine [Stored Procedure](#) um zwei Datensätze in unterschiedliche Tabellen einzufügen. Die erste Anweisung gibt einen inkrementellen Wert zurück, welcher in der zweiten Anweisung verwendet wird.

```
CREATE PROCEDURE [sp_Test]
    (@name [varchar](300))
AS
    SET NOCOUNT ON
    INSERT INTO [Tabelle1]([name]) VALUES (@name)
    SELECT @@IDENTITY
    SET NOCOUNT OFF
    INSERT INTO [Tabelle2] (Tabelle1_ID,[name]) VALUES
    (@@IDENTITY,@name)
GO
```

Zunächst wird per SET NOCOUNT ON angewiesen, dass die Anzahl der betroffenen Zeilen nicht als Teil des Ergebnisses des INSERT INTO zurückgegeben wird. Per INSERT INTO Anweisung wird nun der Datensatz in die erste Tabelle eingefügt. Anschließend wird per SELECT @@IDENTITY der ID-Wert des eingefügten Datensatzes ausgelesen. Die Variable @@IDENTITY enthält immer den zuletzt eingefügten Identitätswert. Nun kann per zweiter INSERT INTO Anweisung der nächste Datensatz eingefügt werden.

Natürlich ist es möglich diese Lösung ohne Stored Procedure zu verwenden, um z.B. die ID direkt im Code weiter zu verwenden.

```
SET NOCOUNT ON;INSERT INTO [Tabelle1]([name]) VALUES
(@name);SELECT @@IDENTITY AS NewID
```

Die inkrementelle ID wird nun als Feld NewID ausgegeben.

7.16. Import einer CSV Datei mit Hilfe von BULK INSERT

BULK INSERTS ermöglichen ein sehr schnelles Einfügen von vielen Daten mit einer Sql-Anweisung. Voraussetzung für diese Art Import ist allerdings der Microsoft SQL-Server. Des Weiteren muss sich die Import Datei ebenfalls auf dem Sql-Server befinden.

Für den Import bereiten wir zunächst die Tabelle in der Datenbank vor. Der Aufbau muss identisch mit der CSV Datei sein. In meinem Beispiel verwende ich die Spalten Daten1, Daten2, Daten3 und Daten4.

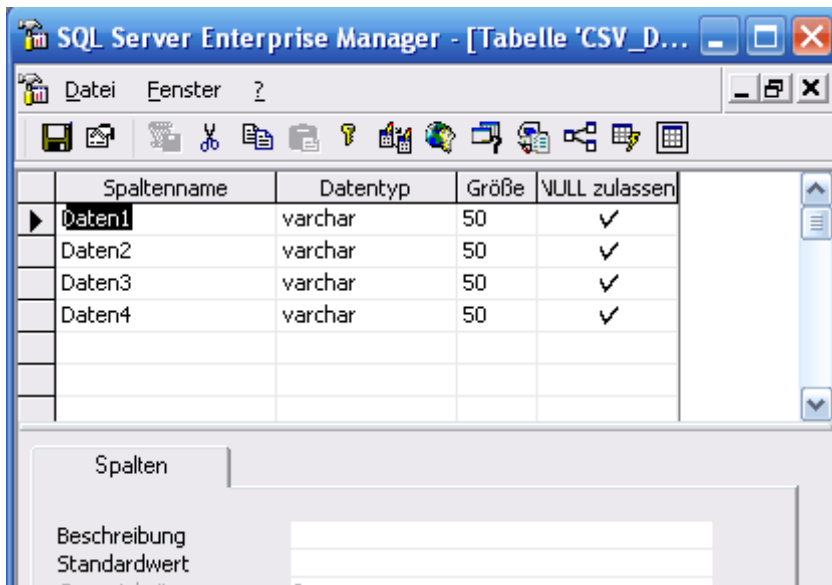


Abbildung 46: Bulk Inserts 1

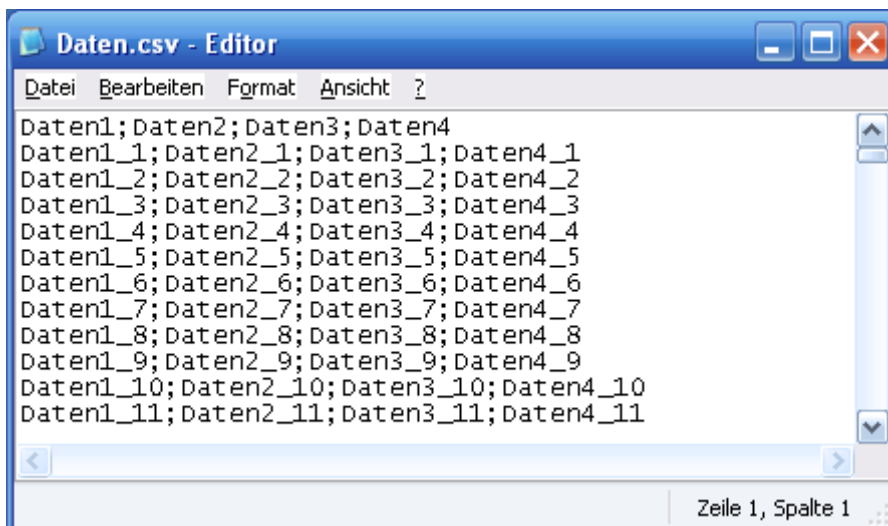


Abbildung 47: Bulk Inserts 2

Wichtig für die BULK INSERT Anweisung ist nun das Trennzeichen der einzelnen Daten und Zeilen, so wie die Zeilennummer der ersten Daten. Die Daten der CSV-Datei werden üblicherweise durch ein Semikolon und die Zeilen durch einen Umbruch getrennt. In der ersten Zeile sind die Spaltenüberschriften angegeben.

Die BULK INSERT Anweisung sieht dann wie folgt aus:


```
BULK INSERT <Tabelle> FROM <CSV-Datei>WITH (FIELDTERMINATOR =  
';', ROWTERMINATOR = '\r\n', TABLOCK, FIRSTROW = 2)"
```

Natürlich muss noch die Zieltabelle und der Pfad zur CSV-Datei angegeben werden. An die Parameter FIELDTERMINATOR und ROWTERMINATOR übergeben wir das Semikolon und den Zeilenbruch. Per TABLOCK sperren wir die Tabelle für die Zeit des Imports und mit FIRSTROW = 2 geben wir an, dass unsere Daten ab der zweiten Zeile beginnen.

Die komplette Anweisung kann nun an ein SqlCommand Objekt übergeben und ausgeführt werden.

Wenn es keinen Fehler gibt, sollte das Programm unmittelbar nach dem Start bereits fertig sein.

Natürlich kann man dieses Beispiel noch beliebig ausbauen und auch die BULK INSERT Anweisung bietet noch wesentlich mehr Parameter an. Wie man anhand der oberen Parameter vielleicht schon erkennen konnte, kann man einen BULK INSERT auch auf andere Dateiformate als das CSV Format anwenden.

Das Beispielprojekt findet man unter der URL:

<http://www.veloursnebel.de/Code/BulkInsert.zip>

8. SONSTIGES

8.1. [Verwendete Ports von Microsoft Produkten](#)

Das ist zwar jetzt nicht das Kerngebiet meines Weblogs, aber nachdem ich zufällig auf diese Übersicht gestoßen bin, möchte ich sie nicht vorenthalten, da es vielleicht für den einen oder anderen nützlich sein könnte.

[Network Ports Used by Key Microsoft Server Products](#)

8.2. [IIS Fehler: Fehler beim Zugriff auf die IIS-Metabasis. Was tun?](#)

Für diejenigen, die nachfolgenden Fehler erhalten, habe ich eine ganz simple Lösung bei der Hand:

```
Serverfehler in der Anwendung /TestIIS.
```

```
-----
```

```
Fehler beim Zugriff auf die IIS-Metabasis.
```

```
Beschreibung: Unbehandelte Ausnahme beim Ausführen der aktuellen  
Webanforderung. Überprüfen Sie die Stapelüberwachung, um weitere  
Informationen über diesen Fehler anzuzeigen und festzustellen, wo  
der Fehler im Code verursacht wurde.
```

```
Ausnahmedetails: System.Web.Hosting.HostingEnvironmentException:  
Fehler beim Zugriff auf die IIS-Metabasis.
```

```
Das zur Ausführung von ASP.NET verwendete Prozesskonto muss über  
Lesezugriff auf die IIS-Metabasis (z.B. IIS://servername/W3SVC)  
verfügen. Informationen zum Ändern der Berechtigungen für die  
Metabasis finden Sie unter  
http://support.microsoft.com/?kbid=267904.
```

```
Quellfehler:
```

```
Beim Ausführen der aktuellen Webanforderung wurde einen  
unbehandelte Ausnahme generiert. Informationen über den Ursprung  
und die Position der Ausnahme können mit der  
Ausnahmestapelüberwachung angezeigt werden.
```

```
Stapelüberwachung:
```

```
[HostingEnvironmentException: Fehler beim Zugriff auf die IIS-  
Metabasis.]
```

```
System.Web.Configuration.MetabaseServerConfig.MapPathCaching  
(String siteID, VirtualPath path) +3492202  
System.Web.Configuration.MetabaseServerConfig.  
System.Web.Configuration.IConfigMapPath.MapPath  
(String siteID, VirtualPath vpath) +9
```

```
System.Web.Hosting.HostingEnvironment.MapPathActual  
(VirtualPath virtualPath, Boolean permitNull) +163  
System.Web.CachedPathData.GetConfigPathData(String configPath)  
+382  
System.Web.CachedPathData.GetConfigPathData(String configPath)  
+243  
System.Web.CachedPathData.GetApplicationPathData() +68  
System.Web.CachedPathData.GetVirtualPathData  
(VirtualPath virtualPath, Boolean permitPathsOutsideApp) +3385711  
System.Web.Configuration.RuntimeConfig.GetLKGRuntimeConfig  
(VirtualPath path) +189
```

Der User mit dem dein IIS läuft hat keinen Zugriff auf die IIS-Datenbank. Entweder du stellst den User um mit dem der IIS läuft, oder du führst folgenden Befehl im Verzeichnis %Systemroot%\Microsoft.NET\Framework\v2.0.50727 aus:

```
aspnet_regiis -ga <WindowsUserAccount>
```

8.3. [C#: Google Web API schon getestet?](#)

Nein? Dann wird's - wie bei mir - nun absolut höchste Zeit.

Als ersten Schritt muss man sich die Google API unter <http://www.google.com/apis/> downloaden und sich einen Account erstellen. Nach dem Account bekommt man eine Google ID zugesendet, mit der 1000 Requests pro Tag durchgeführt werden können.

Nun, als nächsten Schritt einfach ein neues Projekt im Visual Studio erstellen. Nun eine Web-Referenz auf <http://api.google.com/GoogleSearch.wsdl> erstellen und dem Teil am besten den Namen Google geben. Nun ist das wildeste erledigt.

Eine Abfrage sieht dann in weiterer Folge so aus:

```
Google.GoogleSearchResult r = s.doGoogleSearch(googleID,  
keywords,  
0, 10, false, "", false, "", "", "");  
int estResults = r.estimatedTotalResultsCount;  
  
Google.ResultElement[] elements = r.resultElements;
```

Damit lässt sich dann schon etwas machen. Und beispielsweise könnte eine sehr einfache Abfrage-Anwendung so aussehen:

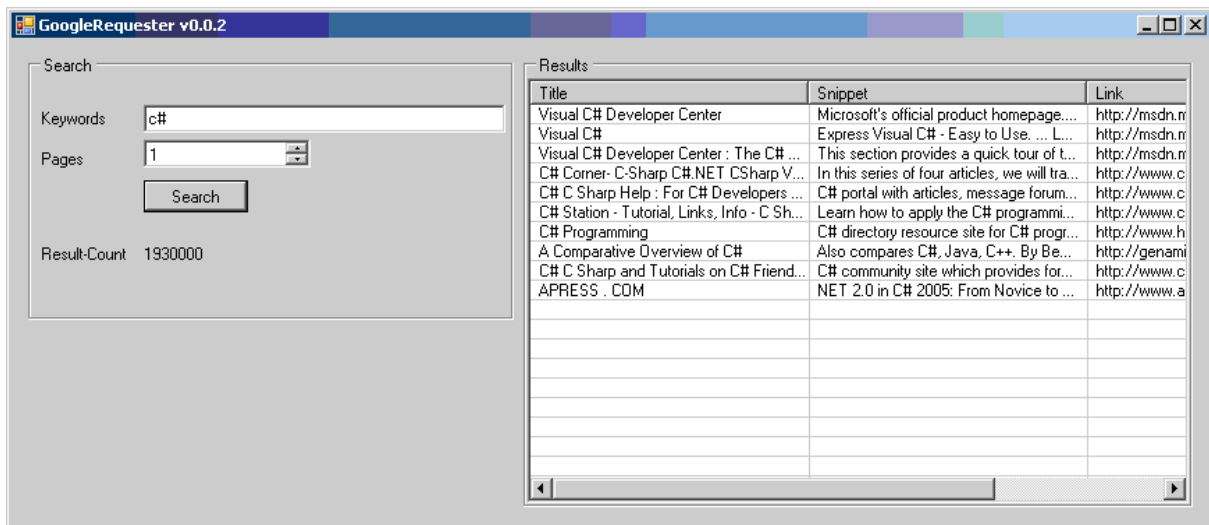


Abbildung 48: GoogleRequester

8.4. [Recent Projects in VS 2005](#)

Wer wie ich momentan noch viel mit dem neuen Visual Studio testet, wird bald eine überfüllte "Recent Projects" Liste haben. Leider habe ich bisher noch keine direkte Möglichkeit in Visual Studio gefunden, diese Liste zu leeren oder Einträge zu entfernen.

Per Regedit lässt sich die Liste allerdings aufräumen. Im dem Schlüssel

```
HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\8.0\ProjectMRUL  
ist
```

findet man alle Projekteinträge. Diese müssen jetzt nur noch entfernt werden. Möchte man Einträge behalten, sollte man auf die korrekte Nummerierung achten (File1, File2, usw.)

8.5. [IE 6.0 Contextmenü](#)

Es gibt Programme, die sich in das Contextmenü des IE's eintragen. Möchte man diese Einträge entfernen, sollte man unter folgendem Pfad in der Registry schauen:

```
HKEY_CURRENT_USER -> Software -> Microsoft -> Internet Explorer -  
> MenuExt
```

Dort einfach die unerwünschten Einträge löschen.

9. ABBILDUNGSVERZEICHNIS

Abbildung 1: System.Environment Visualizer.....	15
Abbildung 2: Reflection Speed Test	26
Abbildung 3: Assembly Key File.....	29
Abbildung 4: Ressourcen schonen - IL-Code	48
Abbildung 5: Fehlender Namespace 1	50
Abbildung 6: Fehlender Namespace 2	50
Abbildung 7: Double Trackbar	57
Abbildung 8: Matrix Test Game	57
Abbildung 9: Bilder im GridView 1	59
Abbildung 10: Bilder im GridView 2	59
Abbildung 11: Mehrere CheckBox Controls markieren	61
Abbildung 12: "Wirklich löschen" im DetailsView 1	63
Abbildung 13: "Wirklich löschen" im DetailsView 2.....	63
Abbildung 14: Login Control ohne returnUrl	64
Abbildung 15: Custom Controls und IntelliSense.....	67
Abbildung 16: "Wirklich löschen" im GridView 1.....	68
Abbildung 17: "Wirklich löschen" im GridView 2.....	69
Abbildung 18: "Wirklich löschen" im GridView 3.....	69
Abbildung 19: PopUp per Response.Redirect.....	70
Abbildung 20: Readonly-Datensätze im GridView 1	71
Abbildung 21: Readonly-Datensätze im GridView 2	73
Abbildung 22: Captcha erstellen	76
Abbildung 23: Caching von Bildern verhindern	78
Abbildung 24: GridView und XmlDataSource	80
Abbildung 25: Reservierte ASP.NET Projektnamen	84
Abbildung 26: WPF Rotation 1.....	88
Abbildung 27: WPF Rotation 2.....	89
Abbildung 28: Visual Studio Default-Browser	92
Abbildung 29: Visual Studio Class Template	93
Abbildung 30: Webprojekte und Firefox 1	95
Abbildung 31: Webprojekte und Firefox 2	95
Abbildung 32: Webprojekte und Firefox 3	96
Abbildung 33: Project Line Counter	97
Abbildung 34: Korrespondierende Klammer	98
Abbildung 35: CCNetConfig	115
Abbildung 36: WMI Code Creator.....	116
Abbildung 37: ASP.NET Deployment Tool 1	118
Abbildung 38: ASP.NET Deployment Tool 2	118
Abbildung 39: Community Server Installation 1	120
Abbildung 40: Community Server Installation 2	121
Abbildung 41: Community Server Installation 3	121
Abbildung 42: Community Server Installation 4	122
Abbildung 43: Community Server Installation 5	123

Abbildung 44: Community Server Installation 6	123
Abbildung 45: Word 2007 - Custom Ribbons erstellen	125
Abbildung 46: Bulk Inserts 1	139
Abbildung 47: Bulk Inserts 2	140
Abbildung 48: GoogleRequester	142