GUI-Programmierung 1: Windows Forms

Proseminar *Objektorientiertes Programmieren mit .NET und C#*

Johannes Faltermeier

Institut für Informatik
Software & Systems Engineering

Abstract: Dieses Dokument beschreibt die Grundlagen für die Programmierung grafischer Benutzeroberflächen mit Windows Forms und C# im .NET-Framework. Als Designer wird Visual Studio 2010 von Microsoft verwendet.

1 Einleitung

Windows Forms ist eine Programmierschnittstelle für die Erstellung grafischer Benutzeroberflächen (engl.: Graphical user interfaces, kurz: GUIs), welche mit .NET 1.0 eingeführt wurde. Sie wird über den Namensraum `System.Windows.Forms` bereitgestellt. Die Steuerelemente (engl.: Controls) wurden für Windows Forms nicht neu entwickelt, sondern entsprechen ihren Pendants aus der Win32-API.

Mit der Veröffentlichung von .NET 3.0 wurde eine zweite Schnittstelle für die GUI-Entwicklung eingeführt, namentlich die Windows Presentation Foundation (kurz: WPF). Windows Forms wird seitdem nicht mehr weiterentwickelt, wird aber wegen teils fehlender Steuerelemente in WPF noch oft verwendet.

[WEB10b]

2 GUI-Programmierung mit Windows Forms

In diesem Abschnitt werden die Grundlagen für die GUI-Entwicklung mit Windows Forms beschrieben. Dabei wird auf Steuerelemente, Eventhandler, das Arbeiten mit mehreren Fenstern sowie den Umgang mit Visual Studio 2010 eingegangen. Sofern nicht anders gekennzeichnet, stammen die Informationen aus: [TW09, S. 252 - 305]

2.1 Grundaufbau Projekt

Legt man im Visual Studio 2010 ein neues Projekt an, so stellt man fest, dass die Entwicklungsumgebung neun Dateien anlegt. Im Folgenden werden die Aufgaben der verschiedenen Dateien kurz erklärt.

Program.cs Wie der Name Program.cs bereits vermuten lässt, handelt es sich bei dieser Datei um den Einstiegspunkt in unser Programm. Sie beinhaltet die `Main()`-Methode.

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
}
```

Mit der Veröffentlichung von Windows XP wurde das Aussehen der Windows Forms-Controls verändert. Durch den Aufruf `Application.EnableVisualStyles()` werden die neuen visuellen Stile verwendet.

Ebenfalls mit Windows XP wurde die Schnittstelle für die Grafikausgabe, welche Zugriff auf Grafikgeräte wie Grafikkarte und Drucker bietet, von GDI (Graphics Device Interface) auf GDI+ umgestellt. Dies kann zu Problemen bei der Textausgabe führen, da in Windows Forms 1.0 und 1.1 GDI verwendet wird, bei Windows Forms 2.0 aber GDI+. Ruft man die Methode `Application.SetCompatibleTextRenderingDefault(bool defaultValue)` mit `false` auf, so wird GDI+ für die Textausgabe verwendet, mit `true` das mit älteren .NET-Versionen kompatible GDI.

`new Form1()` erstellt ein Objekt unseres Hauptfensters (oft auch Hauptformular oder engl. Mainform). Dieses zeigt man mit der Methode `Application.Run(Form mainForm)` an. Dabei wird auch die für Windows-Programme nötige Nachrichtenschleife des Programms angelegt, die beispielsweise Eingabe- oder Systemereignisse entgegen nimmt.

Form1.Designer.cs/Form1.cs Wirft man einen Blick auf die Klassendefinitionen der beiden Dateien, fällt auf, dass beide mit `partial class Form1` deklariert sind. Das partial class-Prinzip ermöglicht es die Klassendefinition einer Klasse auf mehrere Dateien aufzuteilen. In diesem Fall legt der Designer den von ihm generierten C#-Code mit den Oberflächendefinitionen in Form1.Designer.cs ab. In der Datei Form1.cs wird das Verhalten des Programms durch den Programmierer bestimmt. Dort wird der Code für die Eventhandler geschrieben. Das partial class-Prinzip ermöglicht es somit den designergenerierten von selbst geschriebenem Code zu trennen und sorgt somit für mehr Übersichtlichkeit.

AssemblyInfo.cs In diesem File werden Build-Informationen wie Titel, Version oder Copyright abgelegt.

Resources.resx/Resources.Designer.cs Resources.resx ist eine XML-Datei in der die Definitionen einzelner im Programm verwendeter Ressourcen, wie z.B. Texte oder Bilder, abgelegt werden. Das Visual Studio 2010 bietet einen Editor an, mit dem Ressourcen einfach und schnell hinzugefügt werden können, ohne selbst XML-Code schreiben zu müssen. In Resources.Designer.cs wird computergenerierter Code abgelegt, der es dem Programmierer ermöglicht über `Properties.Resources.Bezeichner` Zugriff auf die Ressourcen zu bekommen.

Settings.settings/Settings.Designer.cs/Settings.cs Settings.settings ist ebenfalls eine XML-Datei. In ihr werden Programmeinstellungen (z.B. Farbeinstellungen) gespeichert, die auch zwischen Benutzersitzungen vorliegen. Hier erfolgt der Zugriff, ermöglicht durch Settings.Designer.cs, über `Properties.Settings.Default.Bezeichner`. In Settings.cs ist es dem Programmierer möglich Code abzulegen, mit dem auf das Laden, Speichern und Ändern der Einstellungen reagiert werden kann.

2.2 Steuerelemente

Da hier nicht speziell auf einzelne Steuerelemente eingegangen werden kann, möchte ich an dieser Stelle nur einen kurzen Überblick über einige wichtige Vertreter geben.

Funktion	**Steuerelement(e)**
Textverarbeitung	TextBox, RichTextBox, MaskedTextBox
Informationsanzeige	Label, LinkLabel, StatusStrip, ProgressBar
Auswahl aus Liste	CheckedListBox, ComboBox
Grafikanzeige	PictureBox
Menüs	MenuStrip, ContextMenuStrip
Befehle	Button
Container	Panel, SplitContainer, TableLayoutPanel, FlowLayoutPanel
Datenanzeige	DataGriedView
Datumseinstellung	DateTimePicker

Weitere Informationen zu diesen und weiteren Steuerelementen findet man in der MSDN-Dokumentation. [WEB10c]

2.2.1 Ausrichten der Steuerelemente

Um die Oberfläche schön zu gestalten kommt dem Ausrichten der einzelnen Steuerelemente besondere Bedeutung zu. Auf das Layout nehmen v.a. Container und die Eigenschaften `Dock` und `Anchor` Einfluss.

Container sind Steuerelemente, die wiederum Steuerelemente enthalten und diese auf eine bestimmte Art und Weise anordnen. Der `SplitContainer` erzeugt zwei Panels, die durch einen Balken getrennt sind. Dieser Balken kann beweglich sein, muss es aber nicht.

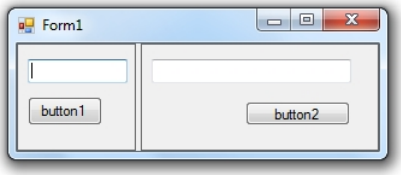

Abbildung 1: SplitContainer

Im `FlowLayoutPanel` werden die Elemente je nach Größe des Fensters in Zeilen bzw. Spaltenform gruppiert. Bei einer Größenänderung verändert sich automatisch die Ausrichtung.

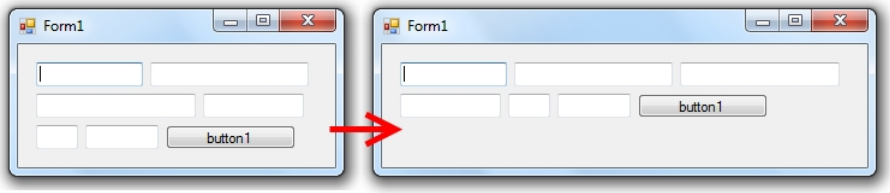

Abbildung 2: FlowLayoutPanel

Mit Hilfe des `TableLayoutPanels` können Steuerelemente in einer festen Tabellenstruktur angezeigt werden.

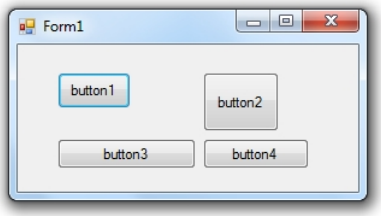

Abbildung 3: TableLayoutPanel

Mit den Eigenschaften `Dock` und `Anchor` lassen sich Steuerelemente an die Außenkanten des übergeordneten Containers bzw. des übergeordneten Formulars binden. Elemente, bei denen die `Dock`-Eigenschaft gesetzt ist, nehmen jeweils die volle Länge der gewählten

Seite oder die verbleibende Fläche des übergeordneten Elements ein. Dies wird auch bei einer Größenänderung beibehalten.

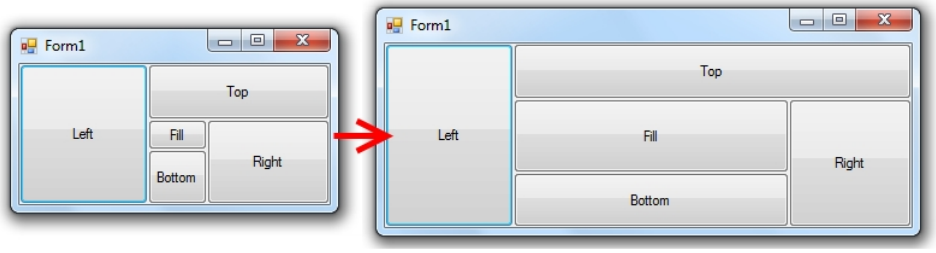

Abbildung 4: Dock

Bei der `Anchor`-Eigenschaft wird das Element relativ zu den Kanten der darüber liegenden Komponente ausgerichtet. Sie ist standardmäßig auf `Left, Top` gesetzt. Das bedeutet, dass bei einer Größenänderung das Steuerelement den Abstand zur oberen und linken Außenkante beibehält. Das folgende Schaubild verdeutlicht die Funktionsweise der `Anchor`-Eigenschaft.

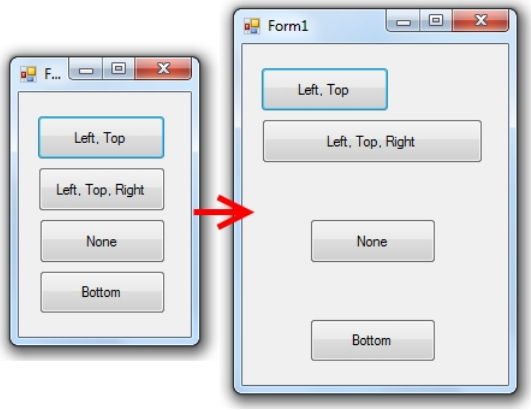

Abbildung 5: Anchor

2.2.2 Steuerelemente zur Laufzeit erzeugen

Es gibt Situationen, in denen es einfacher ist, Steuerelemente zur Laufzeit zu erzeugen anstatt sie vorher im Designer einzeln zu erstellen. Ein Beispiel wäre, wenn man sehr viele Buttons anlegen muss. Dazu erstellt man Objekte der Steuerelemente, weißt ihnen benötigte Eigenschaften zu und fügt sie anschließend zur `Controls`-Collection des Forms hinzu. Dies geschieht mit `Add`, im Falle eines einzelnen Steuerelements, oder mit `AddRange`, im Falle eines Arrays von Controls.

```
//einzelnes Element hinzufügen
Button b = new Button();
...     //Eigenschaften zuweisen
this.Controls.Add(b);

//mehrere Elemente hinzufügen
Button[] buttons = new Button[100];
for (int i = 0; i < 100; i++)
{
    buttons[i] = new Button();
    ...     //Eigenschaften zuweisen
}
this.Controls.AddRange(buttons);
```

2.2.3 Eigene Steuerelemente erstellen

Hierbei handelt es sich um ein sehr komplexes Thema, weswegen verschiedene Arten zur Erstellung von Steuerelementen nur kurz vorgestellt werden können. (vgl. [WEB10a]) Zunächst lohnt es sich einen Blick auf den Ableitungsbaum eines Steuerelements, z.B. eines Buttons, zu werfen.

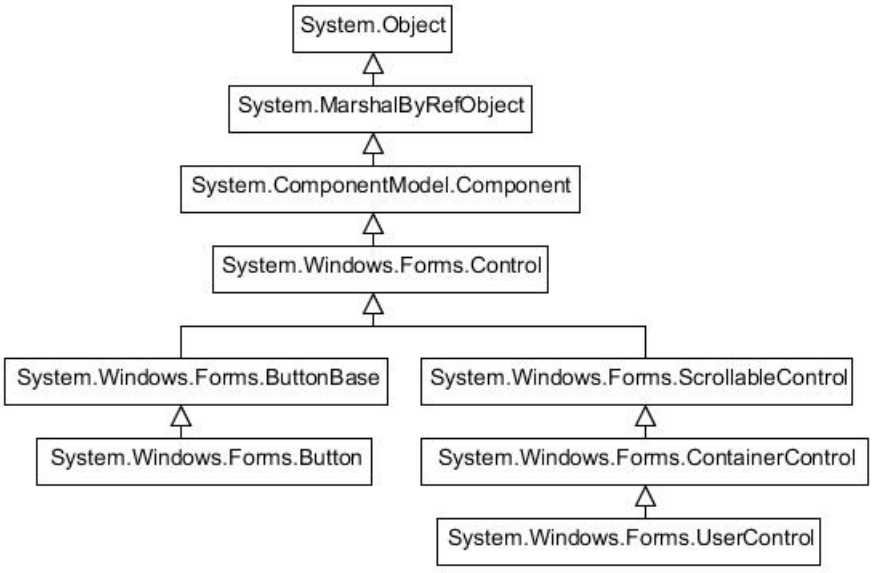

Abbildung 6: Ableitungsbaum Button und UserControl

Möchte man einem bestehenden Steuerelement lediglich neue Funktionalität hinzufügen bzw. eine alte ändern, so bietet es sich an direkt von dem Steuerelement zu erben (z.B. von `Button`). So reicht es aus, neue Methoden und Eigenschaften hinzuzufügen bzw. bestehende zu überschreiben. Dies gilt auch, wenn man nur das Aussehen einer bestehenden Komponente verändern will.

Möchte man hingegen ein komplett neues Steuerelement mit neuem Aussehen und neuer Funktionalität erstellen, so muss man von der `Control`-Klasse erben. Diese Klasse ist die Basis-Klasse für Steuerelemente in Windows Forms. Aussehen und Funktionalität zu implementieren ist aber eine sehr umfangreiche Aufgabe, weswegen dieser Weg nur gewählt werden sollte, wenn es wirklich kein ähnliches Steuerelement gibt, von dem es sinnvoll wäre zu erben.

Mit Hilfe der Klasse `UserControl` kann man mehrere bereits bestehende Steuerelemente zu einem neuen kombinieren. Beispielsweise kann man ein Steuerelement erstellen, welches mehrere Textfelder für bestimmte Benutzereingaben (z.B. Name, Vorname, EMail) gruppiert. Dieses neue Steuerelement kann in der Anwendung auch öfters verwendet werden. Da `UserControl` von `Control` erbt, handelt es sich um ein vollwertiges Steuerelement.

2.3 Eventhandler

Eventhandler sind Methoden, die bestimmte Programmlogik enthalten, die ausgeführt werden soll, wenn ein bestimmtes Benutzerereignis ausgelöst wird.

2.3.1 Beispiel

Als Beispiel wird im Folgendem ein Eventhandler betrachtet, der auf einen Button-Klick reagiert. In Visual Studio 2010 wird ein Eventhandler automatisch mit einem Doppelklick auf das Element, in diesem Fall den Button, hinzugefügt.

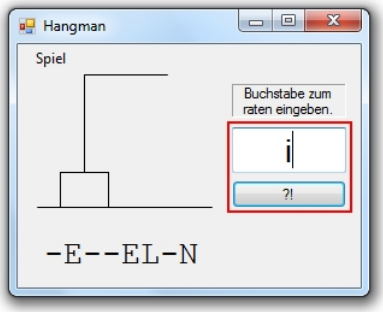

Abbildung 7: Programmbeispiel: Das Spiel Hangman

```
private void button_Click(object sender, EventArgs e)
{
    if (gameStarted)
    {
        cur = textBox.Text;
        textBox.Text = "";
        evaluate();          //wertet Benutzereingabe aus
    }
}
```

`sender` ist das Objekt, welches das Ereignis ausgelöst hat. Da `sender` nur eine `object`-Referenz ist, ist ein Zugriff auf Button-spezifische Eigenschaften und Methoden nur über einen Cast möglich.

```
if (sender is Button)
    (sender as Button).Text = "Done";
```

`e` ist vom eigentlichen Erergnis abhängig. In diesem Fall wird damit nur mitgeteilt, dass der Button gedrückt wurde. Würde z.B. ein `KeyPress`-Ereignis ausgelöst werden, würde `e` u.a. das gedrückte Zeichen enthalten.

Der Eventhandler muss beim auslösenden Objekt und dem jeweiligen Ereignis in den Oberflächendefinitionen registriert werden. Dies geschieht automatisch, wenn man den Eventhandler mit einem Doppelklick im Designer hinzugefügt hat. Wenn dies nicht der Fall ist, geschieht es mit folgendem Ausdruck:

```
this.button.Click +=
    new System.EventHandler(this.button_Click)
```

2.3.2 Ein Eventhandler für mehrere Steuerelemente

Es ist nicht immer sinnvoll Eventhandler mit einem Doppelklick im Designer zu erstellen, da immer ein neuer Eventhandler erstellt und registriert wird. Sollen z.B. ein Menüeintrag und ein Button die gleiche Funktionalität bereitstellen, würde entweder redundanter Quelltext entstehen, man müsste von einem Eventhandler auf den anderen weiterleiten oder man müsste die Funktionalität komplett in eine eigene Methode auslagern. Die beste Möglichkeit wäre es in diesem Fall einen Eventhandler für beide Ereignisse zu schreiben und sie selbst zu registrieren, da dadurch direkt klar wird, dass die gleiche Funktionalität implementiert wird.

2.4 Arbeiten mit mehreren Fenstern

Bei größeren Anwendungen wird man kaum mit einem einzigen Fenster auskommen, weswegen nun das Arbeiten mit mehreren Fenstern betrachtet wird.

2.4.1 Nichtmodale und modale Fenster

Möchte man ein zweites Fenster auf dem Bildschirm anzeigen, steht man vor der Wahl ein modales oder ein nichtmodales Fenster anzuzeigen. Modale Fenster (auch Dialoge genannt) eignen sich, wenn man eine Interaktion mit dem Anwender durchführen will. Modale Fenster können vom Hauptformular nicht verdeckt werden, behalten also immer den Fokus. Sie werden mit der `ShowDialog()`-Methode angezeigt.

```
private void neuToolStripMenuItem_Click(object sender,
EventArgs e)
{
    Form2 input_Dialog = new Form2(); ;
    input_Dialog.ShowDialog();
}
```

Die `ShowDialog()`-Methode hat zudem einen Rückgabewert vom Typ `DialogResult`. Mögliche Rückgabewerte sind `Abort`, `Cancel`, `Ignore`, `No`, `None`, `OK`, `Retry` und `Yes`. Anhand dieser Rückgabewerte kann das aufrufende Programm auf die Benutzereingabe schließen und dementsprechend darauf reagieren. Der Rückgabewert wird entweder für das ganze Formular oder für einzelne Steuerelemente gesetzt.

```
//Rückgabewerte für einzelne Steuerelemente setzen
public Form2()
{
    InitializeComponent();
    button1.DialogResult = DialogResult.OK;
    button2.DialogResult = DialogResult.Cancel;
}

//Rückgabewert für Formular setzen
public Form3()
{
    InitializeComponent();
    DialogResult = DialogResult.OK;
}

//Fenster anzeigen und Rückgabewert prüfen
if (input_Dialog.ShowDialog() == DialogResult.OK) {...}
```

Windows Forms bietet zudem viele vorgefertigte Dialoge, z.B. den `OpenFileDialog` oder den `FolderBrowserDialog`, so dass der Programmierer diese häufig benötigten Dialoge nicht selbst schreiben muss.

Nichtmodale Fenster eignen sich hingegen beispielsweise um weitere Informationen für den Anwender darzustellen. Sie können verdeckt werden und das aufrufende Formular erfährt standardmäßig nicht davon, ob es schon geschlossen wurde oder nicht, da die Anzeigemethode `Show()` keinen Rückgabewert hat. Für Interaktionen sind sie also weniger geeignet.

```
private void neuToolStripMenuItem_Click(object sender,
EventArgs e)
{
    Form2 input_Dialog = new Form2(); ;
    input_Dialog.Show();
}
```

2.4.2 MDI-Anwendungen

MDI-Anwendungen (Multiple Document Interface) sind Programme, die es erlauben in einem Hauptfenster mehrere Kindfenster gleichzeitig geöffnet zu haben. So kann an mehreren Dokumenten gearbeitet werden ohne dass das Hauptprogramm mehrmals gestartet werden muss. Windows Forms bietet einfache Möglichkeiten MDI-Anwendungen zu gestalten. Ein Hauptfenster erstellt man, indem man die `IsMdiContainer`-Eigenschaft des Formulars auf `true` setzt. Zudem ist darauf zu achten, alle Steuerelemente des Fensters mit der `Dock`-Eigenschaft an die Außenkanten des Fensters zu binden, so dass zentral ein Freiraum für die Kindfester bleibt.

Kindfenster erzeugt man, indem man die `MdiParent`-Eigenschaft setzt und sie dann mit der `Show()`-Methode anzeigt. Das Setzen dieser Eigenschaft ist nur zur Laufzeit möglich.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    //Datei -> Neu
    private void neuToolStripMenuItem_Click(object
        sender, EventArgs e)
    {
        Form f2 = new Form2();
        f2.MdiParent = this;
        f2.Show();
    }
}
```

Abbildung 8: MDI: Haupt- und Kindfenster

Um die Anordnung der Kindfenster zu erleichtern, bietet der MDI-Container die `LayoutMdi`-Methode an. Als Parameter wird ein Layout vom Typ `MdiLayout` übergeben. Mit `TileHorizontal` und `TileVertical` werden die Kindfenster untereinander bzw. nebeneinander angezeigt. `Cascade` ordnet die Fenster überlappend an, `ArrangeIcons` hingegen ohne Überlappung.

Zugriff auf alle Kindfenster erhält man über die `MdiChildren`-Collection des Hauptfensters. Möchte man nur auf das aktuelle Kindfenster zugreifen, gibt es im Hauptfenster die Eigenschaft `ActiveMdiChild` vom Typ `Form`. Über einen Cast ist es möglich auf spezielle Eigenschaften und Methoden eines erweiterten Forms zuzugreifen.

```
if (this.ActiveMdiChild is Form2)
    ((this.ActiveMdiChild as Form2).label1.Text =
        "Ich bin aktiv";
```

3 Zusammenfassung und Zukunft

Windows Forms ist eine Bibliothek im .NET-Framework für die GUI-Programmierung. Sie ermöglicht das einfache und schnelle Erstellen grafischer Benutzeroberflächen. Vor allem die vielen vorgefertigten Steuerelemente sind ein großer Pluspunkt, da sie die Anwendungspalette sehr gut abdecken.

Ein Nachteil ist, dass Steuerelemente nur von Programmierern angepasst werden können, da die Steuerelemente in C# beschrieben sind. Ein reiner Grafikdesigner müsste sich erst in C# einarbeiten, um z.B. das Layout zu verändern. In diesem Zusammenhang ist es ein weiterer Nachteil, dass das Zeichnen der Steuerelemente nur über GDI/GDI+ möglich ist, da dies eine Einschränkung bedeutet. U.a. aus diesen Gründen setzt Microsoft seit .NET 3.0 auf WPF, obwohl diesem teils Steuerelemente fehlen.

Literatur

[TW09] Thomas Gewinnus und Walter Doberenz. *Der Visual C# Programmierer*. Hanser, 2009.

[WEB10a] WEBSITE: Arten von benutzerdefinierten Steuerelementen. `http://msdn.microsoft.com/de-de/library/ms171725%28v=VS.100%29.aspx`, Abrufdatum: 15 . 12 . 2010.

[WEB10b] WEBSITE: Erklärung des Begriffs: Windows Forms (WinForms) Was ist Windows Forms (WinForms)? `http://www.it-visions.de/glossar/alle/578/Windows%20Forms.aspx`, Abrufdatum: 15 . 12 . 2010.

[WEB10c] WEBSITE: Windows Forms-Steuerelemente nach Funktion. `http://msdn.microsoft.com/de-de/library/xfak08ea.aspx`, Abrufdatum: 15 . 12 . 2010.