Schritt- für- Schritt Anleitung MS 2010 C#


Inhalt

Literatur ......................................................................................................................................................... 2
Kurs Intro .......................................................................................................................................................... 3
Erste Schritte Programm „Hello World“ ........................................................................................................ 4
Vorbereitungen ............................................................................................................................................... 5
Portieren der exe- Datei auf andere PC ........................................................................................................ 8
Löschen von Komponenten und Source Code ............................................................................................ 8
Kurs 1 Spiel .................................................................................................................................................... 9
Zur Transparenz einige Infos: .................................................................................................................... 15
Kurs 2 Rechnen, Multiwindowing ............................................................................................................. 17
Komma- Problematik .................................................................................................................................... 19
Mehrere Fenster ......................................................................................................................................... 20
Kurs 3 Professionelles Design .................................................................................................................... 26
Menüs .......................................................................................................................................................... 26
PopUp-Menüs mit ContextMenuStrip ....................................................................................................... 27
StatusBar mit StatusStrip .......................................................................................................................... 27
Hints mit ToolTips ..................................................................................................................................... 28
Kurs 4 weitere Komponenten ..................................................................................................................... 29
TabControl .................................................................................................................................................. 30
Komponenten zur Gruppierung und Selektion .......................................................................................... 32
Kurs5 Textverarbeitung mit Stringlisten .................................................................................................. 34
Der Typ String ........................................................................................................................................... 34
Der mehrzeilige Text ................................................................................................................................... 35
Sortieren ..................................................................................................................................................... 38
Leerzeilen weg .......................................................................................................................................... 38
Größe von StringListen ............................................................................................................................ 39
Wörter zählen ............................................................................................................................................... 40
Kurs 6 Projekt Diashow ............................................................................................................................. 41
Projektaufgabe 1 Diashowprogramm ..................................................................................................... 43
Kurs 7 Zeichnen mit dem Graphics - Objekt ............................................................................................ 44
Das Graphics-Objekt ............................................................................................................................... 44
Kurs 8 Chart2D ......................................................................................................................................... 52
Kurs 9 DataTools - Einbinden von Tool- Units .......................................................................................... 56
Kurs 9a Zeichnen von Kurven mit Chart DotNet 4.0 ............................................................................... 58
Kurs 10 Weitere Komponenten ............................................................................................................... 59
WebBrowser ............................................................................................................................................. 59
Mediaplayer ............................................................................................................................................. 60
Kurs 11 Email- Programm ........................................................................................................................ 62
Kurs 12 TreeView ..................................................................................................................................... 62
Kurs 13 Drucken ........................................................................................................................................ 66
Kurs 14 DLL ............................................................................................................................................... 68
Zusammenfassung AD- DA mit dll für ME2600, USB Orlowski ............................................................... 72
ME2600 ...................................................................................................................................................... 72
USB- Orlowski – Karte – alte USB 1.1 - Version ....................................................................................... 72

File: Kurs MS10C# Bayerlein4.6.docx  Seite 1
Neue DLL / Klassenbibliothek mit C# erzeugen ................................................................. 73
Kurs 15 Zeitmessung und 1ms Timer .................................................................................. 74
Kurs 16 RS232 ...................................................................................................................... 76
Kurs 17 bis 20 entfernt .......................................................................................................... 79
Kurs 21 DataGridView ......................................................................................................... 79
Kurs 22 C# DirectX ............................................................................................................. 82
Kurs 23 MySql und C# ......................................................................................................... 82
Kurs 24 Meilhaus ME 4660 .................................................................................................. 83
Kurs 25 NI mit NIDAQmx .................................................................................................... 83
Kurs 26 USB- Orlowski 2.0 ................................................................................................. 84
Kurs 27 Interface mit TCP/IP .............................................................................................. 84
WindfC#, mein großes Regelungstechnik- Toolprogramm .................................................. 85
Einige Hinweise ................................................................................................................... 86
  Problematik mit Dock ........................................................................................................ 86
  Parameter in Funktionsaufrufen und Arrays ...................................................................... 86
  Zeilennummern im Editor ................................................................................................. 87

Literatur

**Kurs Intro**
Alle Screenshots sind mit dem MS Studio 2008 IDE Visual C#, Prof. Version in deutscher Menüversion auf einem Vista-Laptop aufgenommen worden. Aktuell benutzte Version .NET 3.5 !!
Start der IDE (Entwicklungsumgebung) mit

Dann in Datei → Neu → Projekt im folgenden Bild

Windows Forms-Anwendung wählen.


Noch ein Hinweis zum Editor: er unterstützt die Blockbildung von Strukturböcken z.B. von einer if oder for-Anweisung. Das Einrücken wird automatisch durchgeführt, so dass durch diese Schreibweise das Programm besser lesbar wird.

```csharp
if ...condition...)
{
    ...statement...
}
```
Setzt man den Cursor vor eine geschweifte Klammer „{" oder hinter eine geschlossene Klammer „}“, so werden die beiden zu einem Block gehörigen Klammern grau eingefärbt und stehen direkt untereinander. Die folgende oft gesehene Syntax-Form lehne ich ab:

```csharp
if ...condition...){
    ...statement...
}
```

**Erste Schritte Programm „Hello World“**

Vorbereitungen

Zu Beginn sollte das Projekt gespeichert und ein Pfad festgelegt werden. Der Name des Projektes wird anschließend der Programmname der exe-Datei. Bitte nicht auf ein über Netz angeschlossenes Laufwerk, sondern immer auf lokaler Festplatte arbeiten, da z.T. doch große und viele Dateien hin und her geschoben werden.

Bei mir wird in einem Pfad D:MSC#/Kurs 0/ gespeichert, der automatische Name ist WindowsApplication9.

Unit- Source Code

Alle Design- Informationen des Formulars

Jetzt kann die erste Komponente auf das Formular platziert werden. Dazu muss die Toolbox geöffnet werden (Menü Ansicht → Toolbox)
Wir fügen einen Label und einen Button hinzu und setzen sie irgendwo auf das leere Formular

Das Ergebnis:
Mit dem Eigenschaften Fenster können nun Eigenschaften der einzelnen Komponenten sichtbar und verändert werden. Im Beispiel ist der `button1` angewählt (Quadratpunkte als Objektmarkierungen) und wir haben die Eigenschaft „Text“ verändert auf „Erster Test“. Z. B. ist Text immer die Beschriftung einer Komponente, (Name) der Name, der C-Namenskonventionen entsprechen muss (z. B. keine Leer- oder Sonderzeichen). Mit Name kann die Komponenten im C-Source-code angesprochen werden. Die ersten wichtigsten Eigenschaften:


Noch ist keine einzige Zeile C eingegeben worden. Dies geschieht jetzt.
Es öffnet sich der Editor für den C#- Source- Code und man kann jetzt den C#- Source eingeben, der ablaufen soll, wenn der User später im laufenden Programm auf die Taste klickt.
Wir geben folgenden rot umrandeten Text ein:

```csharp
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "Hello World";
}
```

Nun kann das erste Programm kompiliert werden, entweder mit F5 oder mit Klick auf oder über Menü:

Das Programm wird neu abgespeichert, kompiliert, gelinkt und anschließend gestartet. Man erhält letztendlich folgendes Ergebnis:

Vor Klick: 

Nach Klick: 

Es sind nun eine Menge neuer Dateien auf der Festplatte gelandet:

Die anderen Dateien enthalten weitere Projektinfos, welche, spielt hier erst mal keine Rolle.

Die ex ist jetzt das ausführbare Programm, wohl logo.


Bei Endlosschleifen, die vom IDE gestartet werden, ist ein Abbruch mit Shift + F5 oder über Menü Debuggen/Debugging beenden möglich.

Noch ein paar Tipps zum Zurechtfinden:
Öffnen des Source- Codes mit Menü Ansicht → Code (F7)
Öffnen des Form- Designers mit Menü Ansicht → Designer (Shift+F7)
Öffnen der Toolbox mit den Komponenten: Ansicht → Toolbox (Strg+w, dann x)
Öffnen des Eigenschaftenfensters mit Ansicht → Eigenschaftenfenster (Strg+w, dann p)

Portieren der exe- Datei auf andere PC

Löschen von Komponenten und Source Code
Also am Besten: Erst alle Verweise auf diese Komponente löschen, dann alle Event- functions lernen, dann erst die Komponente entfernen.
Also: in obigem Programm nur die Zeile
labell1->Caption="Hello World";
löschen, sonst nichts mehr!

Wenn es nun doch passiert ist und Sie die Funktion komplett gelöscht haben, erscheint nach F5 folgende linke Fehlermeldung.

Jetzt mit Strg+z (Undo) den gelöschten Text wiederholen und nur Ihren Text löschen.

Die function muss erst im Eigenschaftenfenster im Event Click entfernt werden, z. B. durch rechten Mausklick auf Click und dann Reset wählen.

Danach ist wieder alles OK.

Jetzt noch ein paar einfache Übungen:

```csharp
private void button1_Click(object sender, EventArgs e)
{
    labell1.Text = "Hello World"
    button1.Top++
    button1.Left++
    button1.Width++
    button1.Height++
}
```

Das Gleiche über Size und Location, der Code wird etwas komplizierter:

Verändern von Lage und Position des Buttons per Programm über Top / Left usw.
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "Hello World";
    button1.Size = new Size(120, 30); //gleichbedeutend mit Width=120, Height=30
    button1.Location = new Point(20, 50); //gleichbedeutend mit Left=20, Top=50;
}


Gibt man in dieser Eventfunction/Ereignisfunktion die neue Zeile

    label1.Text = button1.Size.Height.ToString();


**Kurs 1 Spiel**

In diesem Kurs soll gezeigt werden, wie einfach und schnell und mit wie wenig echtem C#-Code schon funktionierende Programme geschrieben werden können.

Wir schließen alles Vorherige und starten eine neue Applikation (Datei → Neu → Projekt), dann geben wir folgende Daten im Fenster “Neues Projekt“ ein und wählen Windows Forms-Anwendung. Dann wird alles in dem Verzeichnis C:\Kurs1 Game gespeichert.

Jetzt brauchen wir 2 Bilder, die vom ftp-server geladen und auch dort auf die Festplatte kopiert werden sollten. Es können selbstverständlich auch eigene Bilder sein, aber nicht alle Formate arbeiten korrekt. Hintergrundbild:
Blumen.JPG aus meinem FTP- Verzeichnis MS Studio2008 C#/Kurs1# Spiel und das Mohrhn grHuhn5 echt.bmp.
Dieses Spiel wollen wir jetzt programmieren. Dazu brauchen wir einige Komponenten, die wir auf das leere Formular ziehen:

Erst zwei Komponenten PictureBox von der Seite Allgemeine Steuerelemente.
pictureBox1 soll das Hintergrundbild werden, wir aktivieren die Eigenschaft Image im Objektinspektor durch Klicken auf die Taste mit den drei Punkten und können nun unser Bild Blumen.JPG mit „Lokale Ressource“ und Importieren laden.

Nach Laden sehen wir in der Komponente einen Ausschnitt des geladenen Bildes. Erst ganz zum Schluss wollen wir dieses Bild unter das ganze komplette Formular legen.

Die zweite pictureBox2 füllen wir mit dem Moorhn. Wenn es nach dem ersten Bild auf das Formular gezogen wurde, dann liegt es über dem ersten Bild.
Man kann die Reihenfolge aber in dem PopUp-Menu jeder Komponente leicht verändern:
Dort kann man „In den Vordergrund“ oder „In den Hintergrund“ wählen. Das Moorhn muss vor dem Hintergrundbild liegen.
Soll das Bild die pictureBox vollkommen ausfüllen, so setzt man die Eigenschaften SizeMode auf StretchImage. Den gleichen Effekt erhält man, wenn man auf den kleinen Pfeil oben rechts in der PictureBox klickt und dann die SizeMode/Größenmodus entsprechend wählt.
Übrigens können Eigenschaften der Sorte Bool mit einem Doppelklick auf true oder false in die jeweils andere Eigenschaft geschaltet werden.

Will man jetzt das Hintergrundbild formatfüllend in der Form darstellen, auch dort die SizeMode auf StretchImage stellen und die Dock-Property auf „Fill“ stellen.

Soll das Programm den Bildschirm komplett füllen, dann setzt man in der Form die Eigenschaft WindowsState auf Maximized. Diese Voreinstellungen kann man auch programmieren, das sieht dann so aus:

```csharp
public Form1()
{
    InitializeComponent();
    pictureBox1.Dock = DockStyle.Fill;
    pictureBox2.SizeMode = PictureBoxSizeMode.StretchImage;
}

private void Form1_Activated(object sender, EventArgs e)
{
    Form1 ActiveForm.WindowState = FormWindowState.Maximized;
}
```

Hinter public Form1() kann man Initialisierungen durchführen, jedoch kann man die Form erst auf Maximized stellen, wenn sie aktiviert ist. Die Ereignisfunktion Form1_Activated erzeugt man durch Markieren von Form1, selektieren der Events im Property Manager und dann Doppelklick hinter Activated. Jetzt füllt das Bild mit den Blumen allerdings noch nicht die volle Form aus, da es noch nicht „gestreckt“ wurde. Also auch in PictureBox1 die SizeMode anpassen.

Jetzt werden noch benötigt:

Weiter 2 Buttons, eine Name=buttonStart, Text=Start, die andere Taste Name=buttonStop, Text=Stop.

Userfreundliche Programmierer spendieren noch eine Close-Taste. Diese ist mit einem *Button* leicht zu erreichen. Habe diese Taste *buttonClose* genannt, *Text* auf *Close* und in *Image* das Bild *close.jpg* einbinden, *TextAlign* auf *MiddleRight* und *ImageAlign* auf *MiddleLeft*. In der Click Funktion dann folgenden Text:

```csharp
private void button1_Click(object sender, EventArgs e)
{
    Close();
}
```

Will man aber mehrere solcher Buttons mit Icons haben, so warte man auf die dafür viel bessere Komponente *Toolstrip*.


Diese Tasten lassen sich jetzt z. B. mit ALT+“o“ bedienen.

Danach sollte alles wie gewünscht aussehen, nur es funktioniert noch nichts. Zu einem Spiel wird es erst mit den nächsten Schritten. Was soll passieren: Nach Start soll die Position von PictureBox2 mit einem Zufallsgenerator verändert werden. Klickt der User auf das Bild PictureBox2, soll ein Zähler inkrementiert werden, trifft er daneben (also auf PictureBox1), sollen diese verfehlten Treffer gezählt werden. Bei Stop soll alles anhalten, bei Start die Zähler wieder auf null gesetzt werden.

Dazu brauchen wir eine Timer-Komponente, die regelmäßige Ereignisse zum Verstellen der Position erzeugen kann. Diese finden wir in der Toolbox unter „Komponenten“ als Timer. Er hat die Eigenschaft „Enabled“, die im Eigenschaftsfenster auf False gesetzt wird. Wenn Enabled auf True steht, dann erzeugt er alle in Interval spezifizierten Millisekunden den Event Tick. Also folgende Schritte sind zu tun: timer1 auf Form1 ziehen, Enabled auf False, Interval auf 1000.

Jetzt Start und Stop-Taste programmieren mit Doppelklick auf die jeweilige Taste und füllen der Click-Funktion mit untenstehendem rot umrahmtem Text.

```
private void buttonStart_Click(object sender, EventArgs e)
{
    timer1.Enabled = true;
}

private void buttonStop_Click(object sender, EventArgs e)
{
    timer1.Enabled = false;
}
```

Im Ereignis timer1_Tick wird mit dem Zufallszahlengenerator Random() die Position des oberen Pixels vom Moorhuhn in PictureBox2 im erlaubten Bereich verändert. Der erlaubte Bereich berechnet sich aus der freien inneren Dimension der Form1 (abgefragt in ActiveForm.ClientSize.Width und …Height) abzüglich der Dimension der PictureBox2.

Der vollständige Code in timer1_Tick lautet:

```
Random rand = new Random();
pictureBox2.Left = rand.Next((Form1 ActiveForm.ClientSize.Width - pictureBox2.Width));
pictureBox2.Top = rand.Next((Form1 ActiveForm.ClientSize.Height - pictureBox2.Height));
```

Mit der Methode .Next(zahl) wird eine Zufallszahl zwischen 0 und zahl-1 erzeugt. Jetzt austesten mit F5. Das Moorhuhn muss nach Start über den Bildschirm springen.


```
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    timer1.Interval = (int)numericUpDown1.Value;
}
```

Das explizierte Casting auf int ist nötig, da Value ein Typ „Decimal“ ist.

Den Zähler realisiert man mit zwei Variablen außerhalb der Funktionen. Dieses sind keine globalen Variablen (gibt es in C# nicht mehr), sondern zwei Variablen des Objektes Form1.
Ich mache es mit den beiden Variablen cntOK und cntNOK. Also bei der Start- Taste auf null setzen, bei Click auf `pictureBox1` (daneben)(cntNOK ++), bei Click auf `pictureBox2` Treffer cntOK++ und Ergebnis in `label1` / `2` anzeigen mit der Methode `ToString()`, die den Intgerwert in einen String umwandelt.

Fertiger Source:

In `timer1_Tick`:
```csharp
    label1.Text = "Hits = " + cntOK.ToString();
    label2.Text = "Misses = " + cntNOK.ToString();
```

In `pictureBox2_Click`
```csharp
cntOK++;
```

In `pictureBox1_Click`
```csharp
cntNOK++;
```


Im neuen Source sind `timer1_Tick` und `pictureBox2_Click` unten abgebildet. Die Reihenfolge im Source ist beliebig veränderbar. Im Editor kann man vor jeder Funktion mit den + oder – Zeichen mit quadratischem Rahmen den Code verbergen oder öffnen.

```csharp
int cntOK = 0;
int cntNOK = 0;
bool free = false;

private: void Form1_Activated(object sender, EventArgs e)...
private: void button1_Click(object sender, EventArgs e)...
private: void buttonStart_Click(object sender, EventArgs e)...
private: void buttonStop_Click(object sender, EventArgs e)...
private: void timer1_Tick(object sender, EventArgs e)
{
    Random rand = new Random();
    pictureBox1.Left = rand.Next(Form1.ActiveForm.ClientSize.Width - pictureBox1.Width);
    pictureBox1.Top = rand.Next(Form1.ActiveForm.ClientSize.Height - pictureBox2.Height);
    label1.Text = "Hits = " + cntOK.ToString();
    label2.Text = "Misses = " + cntNOK.ToString();
    free = true;
    cntNOK++;
}
private: void numericUpDown1_ValueChanged(object sender, EventArgs e)...
private: void pictureBox2_Click(object sender, EventArgs e)
{
    if (free)
    {
        cntNOK--; // Misses corrected
        cntOK++;   // count hits
        free = false; // stop counting
    }
```
Zum Abschluss dieses ersten kleinen Projektes wird jetzt noch die Größe des Zieles variiert mit 
numericUpDown2. Dort soll eine Prozentzahl die tatsächliche Größe bestimmen. Steht dort 50, so sollen 
Height und Width eben nur 50% der ursprünglichen Größe besitzen. Im Explorer kann man die 
Größe des Bildes erkennen mit 117* 226 Pixel. Im Ereignis Changed von numericUpDown2 
will dann folgender Text hinzugefügt:
Man beachte, dass vor Start die Originalwerte Height=226 und Width=117 in pictureBox2 
stehen sollten und dass die EigenschaftSizeMode auf StretchImage steht, sonst geht es nicht.
Code:
private void numericUpDown2_ValueChanged(object sender, EventArgs e)
{
    pictureBox2.Width = 117 / 100 * (int)numericUpDown2.Value;
    pictureBox2.Height = 226 / 100 * (int)numericUpDown2.Value;
}

Ein letzter Schönheitsfehler wird beseitigt. Wenn man die Bewegung des Zieles beobachtet, so 
erscheint ganz kurz aber sichtbar das Ziel erst horizontal neben der alten Position. Das liegt 
daran, dass erst Left und dann Top verstellt wird.
Abänderung durch das Verstellen von Location, dann werden x und y- Koordinate gleichzeitig 
verstellt.
Code in timer1_Tick:
int left= rand.Next(Form1.ActiveForm.ClientSize.Width - pictureBox2.Width);
int top= rand.Next(Form1.ActiveForm.ClientSize.Height - pictureBox2.Height);
Point loc1 =new Point(left,top);
pictureBox2.Location = loc1;

Weiterer Variationen wie z. B. High Score Liste, einstellbare Laufzeit, mehrere Ziele sind 
keine Grenzen gesetzt.

Stellt man die Zeit auf <700 ms, so wird es schon sehr schwierig, alle Ziele zu treffen.

Alle fertigen Dateien sind auf dem ftp-Server unter „Kurs1#Spiel“ zu finden. Dort habe ich 
ach noch den Titel der Form (in Form1.Text) geändert.
Jetzt fehlt eine kleine Schönheitskorrektur: Es ist die Transparenz der Label und des Zieles,
was nicht so trivial ist.

Zur Transparenz 
einige Infos:
Eine Form hat die 
Eigenschaft Opacity, 
die zwischen 0 und 
100% eingestellt 
werden kann. Damit 
werden alle Elemente 
mehr oder weniger 
durchscheinend. Bei 
50% sieht das Spiel 
z.B. so aus und der 
Inhalt dahinter
schimmert schon mächtig durch. Dies ist für Auf- und Abblendungen beim Start wohl mal ganz witzig.

Stellt man die Eigenschaft TransparencyKey z.B. auf White, dann bekommt die Anwendung an allen Stellen, die weiß sind, Löcher und der Hintergrund ist zu sehen. Sowohl in der Überschrift als auch die numericUpDown sind jetzt stellenweise durchsichtig. Selbst die Maus klickt durch die Löcher durch. Das Close-Kreuz ist auch durchsichtig!


Die Label werden mit folgendem Schritt gegen das Blumenbild transparent: pictureBox1 löschen samt aller Verweise darauf. Das Bild wird als Hintergrundbild der Form eingebunden. Dazu setzt man dort die Eigenschaften so, wie im Bild setzen. Dann in allen Labels die Eigenschaft BackColor auf Transparent setzen. Diese ist unter „WEB“ zu finden. Ich hab die Farbe in ForeColor auf Yellow gesetzt, dann sieht ein Label so aus:
Kurs 2 Rechnen, Multiwindowing

Es wird die Berechnung, die Ein- sowie Ausgabe von Zahlen gezeigt sowie der Umgang von Projekten mit mehreren Fenstern.


| Änderen Label->Text Inhalte und Edit->Text Eigenschaften wie links gezeigt. Nach Eingabe einer Zahl in Feld Radius soll die Fläche ausgegeben werden und nach Eingabe der Fläche in Area soll der Radius berechnet werden. Die Formeln lauten bekanntlich:
| $\pi r^2$ sowie $r = \sqrt{\frac{A}{\pi}}.$

Nun muss überlegt werden, in welchem Ereignis die Berechnung durchgeführt werden soll. Es soll kein Button zum Start der Berechnung geben.


```csharp
private void textBoxRadius_TextChanged(object sender, EventArgs e)
{
    double radius = Double.Parse(textBoxRadius.Text);
    double area = radius * radius * Math.PI;
    textBoxArea.Text = area.ToString();
}
```

Hinweise zur Berechnung: Die Ein- und Ausgabe erfolgt über die Text- Eigenschaft vom Typ String. Es muss dann eine Konvertierung von Float nach String und umgekehrt durchgeführt werden. Es wird hier der Floating-Point Typ „Double“ benutzt (ca. 15 Stellen), der viel genauer ist als float (7 Stellen). Die Konstante $\pi$ ist in der mathematischen Bibliothek Math zu finden.

2. Leave: Dann wird das Ereignis ausgelöst, sobald der Focus auf eine andere Komponente übergeht, z. B. durch Klick des Benutzers auf textBox2. Nur der User muss das wissen.


```csharp
private void textBoxRadius_Leave(object sender, EventArgs e)
{
    double radius = Double.Parse(textBoxRadius.Text);
    double area = radius * radius * Math.PI;
    textBoxArea.Text = area.ToString();
}
```

File: Kurs MS10C# Bayerlein4.6.docx

```csharp
private void textBoxRadius_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == 0x0d) textBoxRadius_Leave(this, e);
}
```

Oder

```csharp
private void textBoxRadius_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.KeyChar == (char)Keys.Return) textBoxRadius_Leave(this, e);
}
```

Hinter der if-Abfrage wird nun die Berechnung ausgelöst. Hier könnte eine Kopie der Zeilen unter Leave folgen, aber man kann auch eine existierende Ereignisfunktion direkt aufrufen. Im Zeiger „sender“ wird die Adresse des aufrufenden Containers übergeben. Vervollständigen Sie das Programm zur Berechnung auch noch des Radius aus der Fläche nach Druck auf die Eingabe-Taste, wenn die Textbox für „area“ den Focus hat.

Ein Hinweis sei noch getan: Die Ausgabe einer Floating-point Zahl kann auch formatiert erfolgen, indem man in der Methode Tostring(IFormatProvider) den IFormatProvider benutzt.

Beispiele: Ausgabe von 12345.6789 in einer Textbox mit

<table>
<thead>
<tr>
<th>x.ToString();</th>
<th>Keine Formatierung</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.ToString(&quot;f2&quot;);</td>
<td>Festkomma 2 Stellen</td>
</tr>
<tr>
<td>x.ToString(&quot;g2&quot;);</td>
<td>2 Stellen „general“</td>
</tr>
</tbody>
</table>
Dann gibt es noch die Buchstaben „d“ für Dezimalausgabe, dann muss x jedoch ein Typ „decimal“ sein.

Hexadezimalausgabe: x=255 führt mit x.ToString("x10"); zu 0000000fff
Und mit x.ToString("X10"); zu 00000000FF

Das Ergebnis bis hierhin ist in Kurs 2a# Berechnung auf dem ftp zu finden.

**Komma- Problematis**


Eine Lösung bietet meine selbstgeschriebene eigene ConvertToDouble- Funktion. Wenn in einer Zahl nur ein Trennzeichen auftaucht, wird dies als Dezimaltrenner aufgefasst. Wenn mehr als ein Trenner auftaucht, so ist das rechte immer der Dezimaltrenner, die linken immer Tausendergruppierungen. Also egal ob 1,234,78 oder 1,234,56 beide Zahlen werden richtig als 1234 Komma 78 interpretiert, wobei Komma den Dezimaltrenner meint. Dabei ist es egal, auf was für einem Dezimaltrenner der PC gestellt ist, dies wird vorher von der Funktion geprüft. Der einzige nichtlösbare Konflikt taucht auf, wenn der einzige Trenner in einer Zahl ein Tausendertrenner sein soll. Bei dieser Funktion wird er immer als Dezimaltrenner gelesen. Also egal ob 1,234 oder 1,234, diese Zahl wird dann immer als 1 Komma 234 gelesen.

Beispielanwendung:
Neues Projekt nach Kurs 2c# erzeugen. Zum Projekt Hinzufügen die Datei DataToolsMini.cs:

```
x.ToString("c2"); 123.456,79 € „Currency“ 2 Stellen
x.ToString("e2"); 1.234+005 „exponential“
x.ToString("p2"); 12.345.678,90% Prozent
```

Dann diese Datei DataToolsMini.cs suchen und auswählen.
Außerhalb einer function jetzt eine neue Instanz von DataTools erzeugen, indem man das Wort DataTools eintippt und rechte Maustaste → Auflösen wählt:

```
}
```

Dann wird automatischer der namespace von DataTools hinzugefügt, nämlich WINDF7CSharp.
Weiter:

File: Kurs MS10C# Bayerlein4.6.docx Seite 19
Nun eine Textbox und ein Label hinzufügen. Hinter dem Click- Ereignis des Labels dann folgenden Code ergänzen:

```csharp
DataTools dt = new DataTools();

double x = 0;
dt.ConvertToDouble(textBox1.Text, ref x);
label1.Text = x.ToString();
```

Dann kann man das Programm ausprobieren.

**Mehrere Fenster**


Name sowie Text der zweiten Form: FormMain, Name sowie Text der dritten Form: FormInfo. Größen und Position etwa so wie im nächsten Screenshot.

Nun wird allerdings beim Start immer noch das alte Circle- Fenster geöffnet, nicht wie gewünscht FormMain. Dieses kann man ändern in der Datei Program.cs, die wir jetzt öffnen. Dort in der Funktion `Main()` sieht man folgenden Code:

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

Im roten Rahmen steht der Name der Form, die als erstes sichtbar wird. Hier einfach auf FormMain ändern. Dann öffnet sich nach Start unser FormMain:

```csharp
Application.Run(new FormMain());
```

Bevor wir jetzt weitermachen, sollte erstmal die Location der Formulare auf verschiedene Werte gestellt werden, sonst liegen alle Fenster genau übereinander. Also `Location` von FormMain lassen auf (0;0), FormInfo auf (200;200) und FormCircle auf (100;100).
Jetzt zwei Buttons auf das Hauptfenster, also in *FormMain* plazieren, beschriften mit Circle und Info. Mit „Circle“ soll jetzt unser ursprüngliches Fenster geöffnet werden und zwar „nichtmodal“, d. h. der User kann in andere Fenster klicken. Das Info-Fenster soll „modal“ geöffnet werden, d. h. der User muss erst dieses Fenster schließen, bevor er andere Fenster der Applikation erreichen kann. Welche Sorte sinnvoll ist, bestimmen Sie als Programmierer. Die Methoden zum Öffnen der Forms heißen einfach *Show()* und *ShowDialog()*.

```csharp
FormInfo frmInfo = new FormInfo();
Circle frmCircle = new Circle();

private void buttonCircle_Click(object sender, EventArgs e)
{
    frmCircle.Show();
}

private void buttonInfo_Click(object sender, EventArgs e)
{
    frmInfo.ShowDialog();
}
```

Erst werden Instanzen der Formulare erzeugt mit *new*, dann kann man diese Formulare ansprechen mit den beiden Show- Methoden. Die Instanziierung muss dabei außerhalb einer Funktion erfolgen, damit der Name der Instanz (also z.B. frmainfo) für andere Funktionen erreichbar ist.

Jetzt sollte das Programm wie gewünscht funktionieren. Ausprobieren.

Man kann jetzt beliebig zwischen *Circle* und *FormMain* hin und her springen, aber *FormInfo* muss geschlossen werden, um wieder an *Circle* oder *FormMain* zu gelangen. Wird *FormMain* geschlossen, werden die anderen Fenster auch geschlossen und aus dem Speicher entfernt. Nun gibt es ein Problem. Schließt man die Info- Form, dann kann man sie beliebig oft wieder öffnen, die Instanz wird beim Schließen nicht entfernt, sondern nur unsichtbar. Öffnet man aber eine Form mit *Show()* und schließt diese mit *Close()* ist, dann wird die Instanz aus dem Speicher
entfernt. Beim nächsten Klick auf die Taste circle gibt es dann eine Fehlernmeldung, da auf eine Instanz zugegriffen wird, die es nicht mehr gibt:

```csharp
private void buttonCircle_Click(object sender, EventArgs e)
{
   frmCircle.Show();
}
```  

Übrigens jetzt über Menü *Debug* nach *Stop Debugging* gehen und Programm beenden.

Abhilfe 1: Man verlegt die Instanzierung in die Click-Funktion des Buttons. Ergebnis ist, dass nach jedem Wiederöffnen von circle die Daten weg sind und schlimmer bei jedem Klick auf circle ein neues Fenster aufgemacht wird. Im Bild unten gibt es jetzt 3 Fenster.

Abhilfe 2:
Man spendiert eine Close- Taste auf Circle, in der nur die Form unsichtbar gemacht wird mit Aufruf der Methode *Hide()*.

```csharp
private void buttonCircle_Click(object sender, EventArgs e)
{
   Circle frmCircle = new Circle();
   frmCircle.Show();
}
```
private void buttonClose_Click(object sender, EventArgs e)
{
    Circle.ActiveForm.Hide();
}

Dann sollte man aber die Close-Taste im Kopf des Windows-Rahmens auch verschwinden lassen. Z. B. mit der Property *False* gesetzt wird. *ControlBox* der Form, die dann auf *false* gesetzt wird.

Noch besser: *ControlBox* lassen (der User erwartet das ja irgendwie und kennt es) und den Close- Befehl abfangen mit folgendem Event:

```csharp
private void frmCircle_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel = true;
    Visible = false;
}
```

Dann geht es.


```csharp
private void buttonTransfer_Click(object sender)
{
    frmInfo.label1.Text = textBoxRadius.Text;
}
```

In FormInfo soll mit Druck auf einen Button der Inhalt von *textBoxRadius.Text* von Circle herübergeholten werden. Realisierung folgendermaßen:

Der Zugriff von einer Form auf Properties, Methoden und Events anderer Forms ist möglich, sofern man von der Main Form auf die dort instanzierten Forms zugreifen will. Also im obigen Beispiel kann man von *FormMain* auf *frmCircle* und *frmInfo* zugreifen. Was muss man tun: Beispiel: Auf FormMain soll mit einem Button „buttonTransfer“ der Inhalt (.Text) von label1 auf FormInfo verändert werden. Versucht man...

Jetzt wird in `FormMain` ein Zugriff auf `label1` angeboten. Fertiger Code:

```csharp
private void buttonTransfer_Click(object sender, EventArgs e)
{
    frminfo.label1.Text = "This Message comes from FormMain";
}
```

Somit ist also ein Zugriff vom Hauptprogramm, in dem der Konstruktor mit dem Schlüsselwort `new` aufgerufen wird, problemlos möglich.

Aber wie kann jetzt vom Sub- Fenster (Forminfo oder Circle) auf das `FormMain` oder von `Sub` nach `Sub` zugreiften werden. Das geht folgendermaßen. Für den Zugriff braucht das Programm in der Sub- Form eine Referenz (ist natürlich so etwas wie ein `Pointer`/`Zeiger`). Diese Referenz muss vom `Main` an die Sub- Fenster übergeben werden.

Dazu wollen wir jetzt folgendes Beispiel realisieren. Ein Tastendruck auf `FormInfo` soll jetzt den Inhalt einer `TextBox` auf `FormMain` auslesen in ein Label. Ein zweiter Tastendruck soll aus Circle den Radius aus der entsprechenden `Textbox` auslesen und in ein Label in `FormInfo` schreiben.

In `FormMain` folgende Veränderung durchführen:

```csharp
FormInfo frminfo;
public FormMain()
{
    InitializeComponent();
    frminfo = new FormInfo(this);
}
```

In `FormInfo` folgende Veränderung durchführen:

```csharp
FormMain meinMain;
public FormInfo(FormMain m)
{
    InitializeComponent();
    meinMain = m;
}
```
In der ersten Zeile wird ein globaler Zeiger auf FormInfo ohne Konstruktor deklariert. Im Konstruktor (nach new) wird jetzt mit Schlüsselwort `this` der Zeiger auf FormMain an den Konstruktor in FormInfo übergeben.

| In der ersten Zeile wird ein globaler Zeiger auf FormMain ohne Konstruktor deklariert. Im Konstruktor (nach new) wird jetzt mit Schlüsselwort `this` der Zeiger auf FormMain an den Konstruktor in FormInfo übergeben. |

Mit der Zeile

```csharp
label1.Text = meinMain.textBox1.Text;
```

Kann dann in FormInfo die Textbox auf FormMain ausgelesen werden.

Der Zugriff von FormInfo auf Circle geht jetzt nicht direkt, sondern über FormMain. Will man den Radius von Circle nach FormInfo auslesen, geht das einfach mit

```csharp
label1.Text = meinMain frmCircle.textBoxRadius.Text;
```

Das war's.
Kurs 3 Professionelles Design
In diesem Kurs werden die Komponenten StatusBar, Menu, PopupMenu mit einigen wichtigen Eigenschaften wie ToolTips (früher: Hint) vorgestellt.

**Menüs**
Wir wollen zur Übung einfach mal die ersten Einträge des Menüs vom C# abkupfern, allerdings vorerst mal ohne Icons. Bevor man ein Menü definiert, muss man sich als Programmierer sehr genau die Bedienphilosophie überlegen. Dieses ist natürlich nicht Gegenstand dieser Vorlesung, wie man das macht. Also erst mal das Menü unter „Datei“ ansehen und dann nach und nach eingeben:

![WindowsApplication1 - Microsoft Visual Studio](image1)


Das Menü ist natürlich noch ohne Funktion. Jetzt muss für jeden Menüeintrag natürlich noch ein OnClick- Ereignis definiert werden. Entweder macht man einen Doppelklick auf einen Menüeintrag, so dass die IDE ein Funktionsaufruf erzeugt oder man wählt in der Ereignisliste ein vorhandenes Ereignis aus. Wir ergänzen einen dritten Hauptmenüeintrag „Ansicht“ und erzeugen darunter die beiden Menüs „Circle“ und „Info“. Diesen ordne man dann die den bisherigen Tasten buttonCircle oder buttonInfo_Click zugehörigen Click-Ereignis zu. Dann können wir auch mit dem Menü die beiden anderen Fenster öffnen.

Einige zusätzliche Möglichkeiten: Ein Menü lässt sich als Ein/Ausschalter verwenden. Dann muss die Eigenschaft CheckOnClick auf True stehen. Bei jedem Klick auf diesen Menupunkt wird dann die Eigenschaft Checked zwischen true und false umgeschaltet, erkennbar an einem Häkchen vor dem Eintrag.

\textbf{PopUp-Menüs mit ContextMenuStrip}


\textbf{StatusBar mit StatusStrip}

Mit Doppelklick auf „Start Progress“ geben wir dort folgenden Text in die Click-Funktion ein:

```csharp
private void ghiToolStripMenuItemItemClick(object sender, EventArgs e)
{
    toolStripProgressBar1.Value += 10;
}
```

So wird bei jedem Klick auf diesen Eintrag zur Laufzeit der Progress- Balken weiter nach links wandern.

Einen Panel- Text erreicht man mit der Syntax `toolStripStatusLabel2.Text`.


```csharp
private void timer1_Tick(object sender, EventArgs e)
{
    toolStripStatusLabel2.Text = DateTime.Now.ToString();
}
```

**Hints mit ToolTips**


Jedes Programm sollte meiner Meinung eine Möglichkeit haben, diese gelben Dinger auszuschalten. Den erfahrenen User nerven sie nur. Hat das Projekt aber viele 1000 Komponenten mit ebenso
vielen Hint-Texten, ist es mühsam. Will man z. B. alle Tooltips der gesamten Applikation an / und ausschalten, dann muss man nur die Eigenschaft Active in der Komponente Tooltip auf false setzen.

Will man per Programm den Inhalt des ToolTips verändern so gibt es abhängig von der Komponente zwei Versionen, wie das geht:

1. Steht im Eigenschaften- Editor „ToolTip on toolTip1“, dann geht das mit der Methode SetToolTip( object,string); Beispiel:

   ```csharp
toolTip1.SetToolTip(Button1, "Schließen");
   ```

2. Steht im Eigenschaften- Editor ToolTipText, dann einfach z.B.

   ```csharp
   ToolButton1LoadFile.ToolTipText = "Lade Datei";
   ```

Mit folgendem Code kann der ToolTip- Text auch z.B. in der Statusleiste angezeigt werden:

```csharp
private void toolTip1_Popup(object sender, PopupEventArgs e)
{
    toolStripStatusLabel1.Text = toolTip1.GetToolTip(e.AssociatedControl);
}
```

Das geht aber leider nicht, wenn die tooltips inaktiv sind (Active=false). D.h. wenn einer diese ToolTips abgeschaltet hat, steht auch nichts in der Statusleiste.

**Kurs 4 weitere Komponenten**

**ToolBar**

Start dieses Kurses mit leerer neuer Anwendung.


Beim Hinzufügen von Images zwei Hinweise: Zum Einen muss die Bildgröße passend sein zur Bildgröße Imagesize in der ToolStrip-Komponente, dann sollte sie als png oder gif mit transparenter Außenfarbe versehen werden.
Zum anderen muss man sich überlegen, ob diese Bildchen nicht bei größeren Projekten mehrfach Verwendung finden. Dann sollte man sie in die „Projektresourcendatei“ einfügen und steht dann überall im Projekt zur Verfügung. Nach Klick auf eigenschaft „Image“ des Buttons dann so:

**TabControl**

Mit der TabControl-Komponente können Sie in einem Fenster die Oberfläche mehrfach nutzen. Das Umschalten geschieht wie in einem Karteikasten. So kann man solche Oberflächen wie in typischen Windowsprogrammen programmieren. Also aus dem Folder Container eine TabControl auf das Formular ziehen. Normalerweise wird diese Komponente nach unten gedockt. Damit automatisch bei Fenstergrößenänderung die einzelnen Elemente sich vernünftig mit ändern, sollte man folgende Strategie anwenden (hab
noch ein Panel als reservierte Fläche für Tasten oder andere Elemente auf das Formular gezogen):

Oben ToolBar auf Top, dann ToolStrip auf Top, dann Panel (eigener Container aus „Containers“) wieder Top, der klebt dann unter dem ToolStrip.

Nun wieder zurück zur ToolControl.

Neue Pages werden mit rechtem Mausklick→PopUpMenu erzeugt, dort „Registerkarte hinzufügen“. Bitte jetzt mal 7 neue Pages erzeugen. Diese heißen automatisch tabPage1 bis 7.

Jetzt achte man darauf, wie man einzelne Pages im IDE anhält. Erst Klick auf den Anfasser (Tab), dann ist aber die TabControl angewählt. Dann nochmal unten auf die Seite klicken. Man achte auf die weißen kleinen Punkte. Erst dann ist die Page selektiert ( Kontrolle im Eigenschaftenfenster).

TabControl selektiert  

Page selektiert

multiline = false  
multiline = true

Reicht die Seitengröße für alle Pages nicht aus, so erscheinen die von anderen Anwendungen gewohnten beiden Pfeile zur Verschiebung der Anfasser. Will man bei zu engem Platz mehrere Zeilen für die Anfasser vorsehen, so setze man in der TabControl (erst selektieren!) die Eigenschaft Multiline auf true.

Hier müssen Images per ImageList ergänzt werden, die Auswahl der Icons in den einzelnen Seiten wieder mit der Eigenschaft ImageIndex. Also vorher eine ImageList auf die Form ziehen und mit einer Liste von Bildchen füllen. Die Größe der Icons in ImageList einstellen.
**Komponenten zur Gruppierung und Selektion**

Jetzt sollen Komponenten zur Gruppierung und Selektion gezeigt werden. Wir fügen auf tabPage1 hinzu (alles Standard):

![Image](image-url)

1. GroupBox
2. Panel

Auf tabPage1 3*RadioButton1-3
Auf groupBox1 3*RadioButton4-6
Auf das Panel3* RadioButton7-9

Bewegt man nun einen „Container“, z. B. das Panel, dann bewegen sich die darauf befindlichen Elemente mit. Zuordnung geht mit Hinenschieben,

Die CheckBox-en lassen eine einzelne Auswahl zu. Ob angewählt oder nicht, ist in Eigenschaft Checked zu sehen.


Panel und GroupBox unterscheiden sich eigentlich nur in der Überschrift der GroupBox. Das Aussehen des Randes von Panel kann man leider nicht gleich dem der GroupBox machen, dann nehme man halt eine Group-Box und lösche die Beschriftung in Text.

Auf tabPage2 weitere Selektionsmechanismen:

Weitere Selektionskomponenten sind ListBox, ComboBox aus Standard und CheckListBox aus „Allgemeine Steuerelemente“. Alle drei Komponenten enthalten als Textinhalt eine Collection oder „Auflistung“ namens Items. Alle Funktionen, die mit Collections getan werden können, sind auch hier möglich (dazu siehe späteres Kapitel). Füllen der einzelnen selektierbaren Zeilen mit dem Collectioneditor im Eigenschaftenfenster unter Items. Hier mal ein paar Zeilen Quatschtex in alle drei Komponenten eingeben.

Bei der ComboBox sieht man das selektierte Element oben im Edit- Feld. Es benötigt nur wenig Platz, man muss es halt mit einem zusätzlichem Mausklick aufmachen.

Bei der ListBox sieht man alle Möglichkeiten, wenn die Komponente groß genug ist. Sonst gibt es einen Scrollbalken. Mit der Eigenschaft Sorted=true lassen sich die Einträge alphabetisch sortieren. Wenn
**SelectionMode auf One steht, dann kann man nur ein Element selektieren.** Bei **MultiSimple** wird jede Zeile selektierbar und die Selektion **toggelt**, bei **MultiExtended** selektiert man wie beim Explorer gewohnt, die Shift und Strg- Tasten sind dann auch möglich. Die Abfrage, welche Zeile selektiert ist, kann man mit der Eigenschaft *SelectedIndices[]* durchführen. Es ist eine Eigenschaft vom Typ int- Array und ist im Eigenschaftenfenster nicht zu sehen. Jede Zeile startet bei Nr. 0. Ist z.B. Zeile 3 selektiert, dann ist der Wert von *SelectedIndices.Count* gleich 1 und in *SelectedIndices[0]* steht die Nummer der Zeile, also 2. Dieses mache man sich deutlich mit folgendem einfachen Code in dem Ereignis *SelectedValueChanged*.

```csharp
private void listBox1_SelectedValueChanged(object sender, EventArgs e)
{
    label1.Text = listBox1.SelectedIndices[0].ToString();
    label2.Text = listBox1.SelectedIndices.Count.ToString();
}
```

In *label1* wird die Nummer der ersten selektierten Zeile und in *label2* die Anzahl der selektierten Zeilen dargestellt.

Die **CheckListBox** ist ähnlich, nur man sieht vor jeder Zeile ein **CheckBox- Feld.** Diese kann man wie bei einer **CheckBox** bei jedem Klick toggeln. Abfrage nun allerdings über 
*CheckedIndices[]*, auch hier ein int Array mit den Zeilennummern. Auch ein kleines Programm dazu: Nur wenn die dritte Zeile einen Pfeil trägt, ist die Farbe der *tabPage* rot, sonst weiß.Code:

```csharp
private void checkedListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if ((checkBox1.CheckedIndices.Count == 1) &&
        (checkBox1.CheckedIndices[0] == 2))
        tabPage2.BackColor = Color.Red;
    else tabPage2.BackColor = Color.Transparent;
}
```

Der Zugriff auf den Text der Zeilen geht mit .Items[0] wie bei einem Array, die Nummer wird einem NumericUpDown der Eigenschaft Value entnommen :

```csharp
label2.Text = listBox1.Items[(int)numericUpDown1.Value].ToString();
label3.Text = checkedListBox1.Items[(int)numericUpDown1.Value].ToString();
```

Bei der CheckListBox kann man das Kästchen mit folgendem Befehl direkt setzen (true) oder zurücksetzen (false):

```csharp
checkedListBox1.SetItemChecked((int)numericUpDown1.Value, true);
```

Abfragen mit

```csharp
checkedListBox1.GetItemChecked((int)numericUpDown1.Value)
```

liefert bool- Wert zurück.

**Ergebnis: Siehe Kurs4# weitere Kompos.**
Kurs5  Textverarbeitung mit Stringlisten


**Der Typ String**

Auf der ersten Seite TabPage1 werden zum Testen einiger Stringeigenschaften 8 Label untereinander plaziert. Mit Doppelklick auf Label1 eine Click-Event-Funktion erzeugen, in der einiges gemacht werden soll. Die Tooltip- oder Text- Eigenschaften der Komponenten sind auch Eigenschaften vom Typ string. Will man eigene Variablen deklarieren, so verwende man den Typbezeichner string (wohlgeräumt mit kleinem s!), String gibt es auch, ist Name der Klasse!).

Als erstes folgende Zeilen in diese label1Click() eingeben, grüner Text kann natürlich weg, dient nur zur Info, was nicht geht.

```csharp
private void label1_Click(object sender, EventArgs e)
{
    //char a[20]="12345678901234567890"; funktioniert in C# nicht
    char[] a = { '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '1', '2' };
    string b = new String(a);
    label1.Text=b; // ...=a; // ...=a.ToString(); funktioniert nicht
    String...
}
```

Es wird in der ersten Zeile ein Char-Array deklariert und initialisiert. In der zweiten Zeile wird dieses Array dem String b zugewiesen. Erst dann kann man diese Zeichenkette nach Text von Label1 zuweisen.

Die Zuweisung anders herum geht allerdings so nicht. Versucht man dem char-Array einen String zuzuweisen, geht es schief, da "a" ein Zeiger ist. Diese Zuweisung geht nur mit der Methode ToCharArray etwa so:

```csharp
a = b.ToCharArray();
```

Der ganz große Nachteil von Char-Arrays in Standard herkömmlichen C ist die statische Größe. Das ist jetzt in C# anders. Das Char-Array wird mit dynamischer Größe gemanaged!!! Siehe dazu folgender Code:

```csharp
for (int i = 0; i < 1000; i++)
    label4.Text = label4.Text + b;
label5.Text = label4.Text.Length.ToString();
```


```csharp
char[] a = { '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '1', '2' };
string b = new String(a);
label1.Text=b; // ...=a; // ...=a.ToString(); funktioniert nicht
a = b.ToCharArray();
```

Siehe dazu folgender Code:
a = label4.Text.ToCharArray();
label6.Text = a.Length.ToString();


Wichtige Methoden der Strings sind noch .Substring(x,y), wobei hier ein Teilstring als Ergebnis herauskommt mit Startzeichen bei x anfangend mit 0 und Länge y. Also liefert

label7.Text = label2.Text.Substring(13, 5);

als Ergebnis das Wort “probe”. Die Methode IndexOf(„suchstring“) kann die Position eines Teilstrings suchen. So liefert die Zeile

label8.Text = label2.Text.IndexOf("probe").ToString();

die Zahl 13, da in label2 der Teilstring „probe“ bei Zeichen 13 beginnt.

Weitere Methoden finden Sie in der Hilfe: Cursor auf das Wort string, dann F1: ToLower() ersetzt alle Großbuchstaben durch kleine
ToUpper() ditto durch Großbuchstaben
Replace() ersetzt einen Teilstring.

Fazit strings: niemals wieder sich mit Char-Arrays herumquälen.

**Der mehrzeilige Text**


```csharp
string[] abc = new string[5];
```

Nur ist die Anzahl der Elemente dann fest auf 5 und kann nicht verändert werden.
Besser wäre dann die Collection:

```
Collection<string> strcoll = new Collection<string>();
```


Hinzufügen eines Elementes mit: strcoll.Add("12345");

Zugriff wie Array mit label2.Text = strcoll[(int)numericUpDown1.Value];

Die Größe wird dynamisch verwaltet.

Nun zu Komponenten mit mehrzeiligem Text:
Die TextBox wird mehrzeilig, wenn man die Eigenschaft Multiline auf true stellt. Wir ziehen eine textBox auf tabPage2, und ein RichTextBox auf tabPage3 und setzen beide Dock=Fill. Schauen wir uns zuerst einmal textBox1 an. Einige wichtige Eigenschaften:

- `->ReadOnly` bestimmt, ob der User Text verändern kann (false) oder nur angezeigt wird (true)
- `->Font` bestimmt Schriftart Farbe etc
- `->ScrollBars` bestimmt, ob diese vorhanden sind. Es sollte immer eine Vertikale ScrollBar eingestellt werden
- `->WordWrap`, wenn true, dann werden längere Zeilen umgebrochen, sonst verschwinden sie rechts, sind mit Scrollbar aber zu erreichen: ich würde immer auf false stellen, dann bleiben die Zeilenumbrüche von Originaltexten erhalten.
- `MaxLength`, steht defaultmäßig auf 32767, begrenzt die Anzahl der Zeichen des Users auf diese Zahl. Sollte auf 0 stehen, dann ist die Grenze bei 2^32 = 4GByte Zeichen.

Die Vorbelegung des Textbereiches geschieht per Eigenschaftenfenster bei der Eigenschaft Lines. Damit öffnet sich ein Texteditor zum Eingeben des Starttextes.

Wichtig zu wissen: Der Inhalt wird immer in einem einzigen string, nämlich Text gespeichert. Ruft man die Eigenschaft .Lines auf, so wird intern der Einzelstring in ein Textarray umgebrochen, was bei großen Texten sehr zeitraubend sein kann. Will man oft auf große Texte per Lines zugreifen, dann am Besten vorher einmal einen Umbruch erzwingen und dann auf das lokale Textarray zugreifen:

```
string [] LocalArray = textBoxRadius.Lines;
```

Wird der Text verändert, muss man dieses natürlich wiederholen.

Für das Programm in diesem Kurs sind zum Laden und Speichern noch zwei Dialoge wichtig: in Dialogs findet man die Komponenten OpenFileDialog und SaveFileDialog. Bitte jetzt zwei Tasten oben auf die Form, Name Load und Save. Dann die beiden Dialoge hinzufügen.

In der Click- Funktion von Load folgt jetzt der Standard- Code zum Öffnen von Datei- Auswahl- Boxen, die dem üblichen Windows- Design entsprechen:

```
if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {  // ......... Öffnen der Datei, Einlesen Abspeichern etc
}
```

Die Methode ShowDialog() öffnet das typische Auswahlfenster. Nur wenn der User auf die OK- Taste klickt, wird die Aktion ausgeführt, bei Taste Abbruch nicht.

Das Laden einer Text-Datei in die Textbox geschieht nun über die Klasse StreamReader mit folgendem Code: (vorher oben mit using…):

```
using System.IO;

StreamReader myStream1 = new StreamReader(openFileDialog1.OpenFile());
textBox1.Text = myStream1.ReadToEnd();
```

Mit der Methode `ReadToEnd()` wird dann der Textinhalt der Datei (muss eine ASCII sein) in die `textBox` eingelesen. Problem mit dem Text gibt es allerdings mit den deutschen Sonderzeichen ü, ö ä ß, die einfach ausgelassen werden. Dieses kann man verhindern, wenn man die Instanzierung nach `new` mit folgender Ergänzung durchführt:

```csharp
...new StreamReader(openFileDialog1.OpenFile(), Encoding.Default);
```

Tipp: Bei großen Texten dauert insbesondere das “scrolling“ der sich füllenden `textBox` sehr lange, besser vor Einlesen Textbox Visible auf false, danach wieder auf true. Geht so viel schneller.

Abspeichern mit der Save- Taste geht so:

```csharp
if (saveFileDialog1.ShowDialog() == DialogResult.OK) {
    StreamWriter myStream1 = new StreamWriter(saveFileDialog1.OpenFile(), Encoding.Default);
    myStream1.WriteLine(textBox1.Text);
    myStream1.Close();
}
```

Die Dialoge sollten noch etwas angepasst werden: im `openDialog1` folgende Eigenschaften setzen:

- Auf jeden Fall sollten die Filter gesetzt werden, z. B. so wie links. Immer sollte ein Filter „Alle“ mit *.* dabei sein, damit man immer sehen kann, was alles auf dem Verzeichnis vorhanden ist. Die Syntax ist: (*Bezeichner*)|*.xxx|(*Bezeichner*)|*.yyy usw. Also für Text und C#-Files z. B. Filter=Text|*.txt|C#|*.cs|All|*. *
- In `InitialDir` kann das Startverzeichnis gewählt werden, in `Title` die Überschrift der `FileSelectBox` und in `FileName` der voreingestellt Name.

**Ganz wichtig** in der `SaveDialog`-Komponente ist `DefaultExt`, die den Suffix bestimmt, wenn der User keinen eintippt. Dort sollte also immer .txt stehen, dann wird die Datei ohne Angabe eine txt- Datei.

**Ganz wichtig:** der `OverwritePrompt` in Options sollte immer auf true gestellt werden, damit der User vor Überschreiben gewarnt wird.

```csharp
label10.Text = "Lines= " + textBox1.Lines.Length.ToString();
```

Mit den beiden oberen Zeilen werden sowohl Anzahl der Zeichen in `Label9` und Anzahl der Zeilen in `Label10` ausgegeben. `Length` ist eine Eigenschaft vom typ int.

Man kann jetzt einzelne Zeilen erreichen durch Ansprechen der Eigenschaft `Lines` in der `TextBox`.

Dabei benutzt man die Array- Indizierung mit [ ]. Folgender Code liest die Zeile in eine zweite `textBox2`, dessen Zeilenummer startend mit Null von einem `numericUpDown` eingestellt wird. Um Zugriffsfehler zu vermeiden, sollte vorher immer der Maximalwert des `UpDown` auf die Zeilenzahl -1 gestellt werden.

```csharp
private void numericUpDown1_ValueChanged(object sender, EventArgs e) {
    numericUpDown1.Maximum = textBox1.Lines.Length-1;
```
Die `textBox` kann nur alle Zeichen mit einen einzigen Font darstellen, die RichTextBox dagegen lässt Formatierungen für jedes einzelne Zeichen zu.

Nun noch ein bisschen Textverarbeitung:

**Sortieren**

Es sollen alle Zeilen alphabetisch sortiert werden. Nun lässt die `Textbox` dies direkt nicht zu. Also muss man den Umweg über z.B. die `ListBox` machen, die kann sortieren. Eine `Collection` leider auch nicht. Damit das nicht sichtbar wird, kann man die `ListBox` mit `Visible=false` unsichtbar machen. Und natürlich in der `ListBox die Eigenschaft Sorted=true`.

Source hinter einer Taste Sort:

```
1 listBox1.Items.Clear();
2 for (int i = 0; i < textBox1.Lines.Length; i++)
3     {
4         label10.Text = "Lines= " + i.ToString();
5         listBox1.Items.Add(textBox1.Lines[i]);
6         Application.DoEvents();
7     }
8 textBox1.Clear(); textBox1.Visible = false;
9 for (int i = 0; i < listBox1.Items.Count-1; i++)
10   textBox1.AppendText(listBox1.Items[i] + "\0d" + "\0a");
11   textBox1.AppendText(listBox1.Items[listBox1.Items.Count - 1].ToString());
```

Kommentar dazu:

Zeile 1: Zwischenspeicher Listbox leeren.
Zeile 2: for- Schleife zum Umschaufeln und gleichzeitigem Sortieren
Zeile 4: Fortschrittsanzeige, sinnvoll bei langen Texten, dann dauert es sehr lange
Zeile 5: Umschaufeln
Zeile 6: Dieser Befehl arbeitet alle Ereignisse ab, die während der Schleife auflaufen. Damit wird auch ein Neuzeichnen des `label10` erzwungen, sonst würde man den Fortschritt nicht sehen.
Zeile 8 Start Zurückschaufeln mit Leeren. Gleichzeitig wird die Textbox unsichtbar gemacht, um das sehr zeitaufwende Scrollen zu vermeiden.
Zeile 9: Schleife zum Zurückschaufeln
Zeile 11: Die letzte Zeile des Textes soll kein Linefeed bekommen.

**Leerzeilen weg**


```csharp
Collection<string> strcoll = new Collection<string>();
for (int i = 0; i < textBox1.Lines.Length; i++)
if (textBox1.Lines[i].Length > 0) strcoll.Add(textBox1.Lines[i]);
textBox1.Clear(); // copy Back to Textbox
for (int i = 0; i < strcoll.Count - 1; i++)
textBox1.AppendText(strcoll[i] + "\0d" + "\0a");
textBox1.AppendText(strcoll[strcoll.Count - 1]); // last line
```
**Größe von StringListen**

Die Größe ist im Prinzip nicht begrenzt. Wir fügen Taste hinzu, die `textBox1` über `ListBox` immer verdoppelt. Also kopiere `textBox1` nach `ListBox1` und zurück ohne zu sortieren und Textbox zu leeren. Anzeige der Zeilen in `Label9` und der Zeichenzahl in `Label10`:

```csharp
listBox1.Items.Clear(); listBox1.Sorted = false;
for (int i = 0; i < textBox1.Lines.Length; i++)
{
    label10.Text = "Lines= " + i.ToString();
    listBox1.Items.Add(textBox1.Lines[i]);
    Application.DoEvents();
}
textBox1.Visible = false;
for (int i = 0; i < listBox1.Items.Count - 1; i++)
    textBox1.AppendText(listBox1.Items[i] + "\0d" + "\0a");
textBox1.AppendText(listBox1.Items[1].ToString());
textBox1.Visible = true;
lable10.Text = "Lines= " + textBox1.Lines.Length.ToString();
```

Beschleunigen eventuell mit

```csharp
string[] LocalArray = textBoxRadius.Lines;
```

Textbox und `RichTextBox` speichern den Text in einem `String[]` Array namens `Lines`. Es gibt zwar eine Methode `.CopyTo()`, aber der Befehl

```csharp
textBox1.Lines.CopyTo(richTextBox1.Lines, 0);
```

wird ohne Fehler kompiliert, aber es passiert nicht das erwartete Kopieren, **sondern nichts**!!.

`ListBox` hat seine String in `Items`, das scheint eine String Collection zu sein. Dort gibt es eine `CopyTo()` – Methode, sie funktioniert aber nicht so wie man das als User wünscht.

```csharp
Collection<string> strcoll = new Collection<string>();
listBox1.Items.CopyTo(strcoll, 0);// syntax error
```

wird mit einer Fehlermeldung abgebrochen und nicht kompiliert. Wieso nicht??

Leider auch hier: eine Kopiermethode wie AddStrings oder meinetwegen auch CopyTo() von String[] Array nach Collection gibt es nicht. Dabei sind die Daten einfach nur mehrzeilige Texte.

**Wörter zählen**

Noch ein Codebeispiel, das oft nützlich ist: das Auffinden von Teilstrings. Damit kann man z.B. zählen, wie oft ein Wort in einem Text vorkommt. Dazu ist die Methode .Substring(a,b) der Strings noch zu beschreiben. Sie liefert einen String zurück, der im Ausgangsstring bei Buchstabe an Position a (Start von Null) beginnt und b Zeichen enthält. Also nach Ausführung folgenden Codes

```csharp
label2.Text = "dies ist ein probetext";
label7.Text = label2.Text.Substring(13, 5);
```

steht in label7 das Wort “probe”. Ein Zeilenumbruch in Text sind zusätzlich noch einmal 2 Zeichen (\n\r).

Hinter der Taste z.B. „Item Search“ steht folgender Code, der in Textbox1 nachzählt, wie oft dort der Begriff vorkommt, der in Textbox2 steht.

```csharp
string item = textBox2.Text;
int len=item.Length;
int cnt = 0;
for (int j = 0; j < textBox1.Text.Length-len+1; j++)
    if (textBox1.Text.Substring(j, len) == item) cnt++;
label9.Text = "Hits= " + cnt.ToString();
```

Das dauert aber leider bei großen Texten sehr lange. Sucht man in Bibel.txt (Auf ftp bei mir zu finden) nach dem Wort „Gott“, so dauert es Stunden.

Etwas 5 sec dauert gleiche Suche mit folgendem leicht modifiziertem Programm:

```csharp
string item = textBox2.Text;
string text = textBox1.Text;
int len=item.Length;
int textlen=text.Length-len+1;
int cnt = 0;
for (int j = 0; j < textlen; j++)
    if (text.Substring(j, len) == item)
        
        cnt++;
        label9.Text = j.ToString();
        label10.Text = cnt.ToString();
        Application.DoEvents();
    }
label9.Text = "Hits= " + cnt.ToString();
```

In der Schleife mit dem ca 4,6 Millionen Zeichen der Bibel wird nur in einem einfachen string gesucht. Dauert bei meinem Lap ca. 5s.

Eine andere Alternative ist die Methode .IndexOf(a,b), die sehr schnell ist. Außerdem sind jetzt die Ausgaben in die Label, die auch sehr zeitintensiv sind, weg. Neue for- Schleife:

```csharp
for (int j = 0; j < textlen; j++)
    
    startindex=text.IndexOf(item,j);
    if (startindex >=0)
        
        cnt++;
        j = startindex+1;
```

```csharp```
Jetzt merkt man die Rechenzeit kaum. 
Alle diese Teilprogramme sind in dem Verzeichnis Kurs 5# Strings zu finden.

Kurs 6 Projekt Diashow

Ehemals früher als Hausaufgabe, jetzt als Projekt1 wollen wir ein Programm schreiben, das aus einer Serie von Fotos (JPG, gif, Png in einem beliebigen Verzeichnis) in einer Diashow darstellen.
Dazu braucht man als erstes einen Zugriff auf das Filesystem des Rechners. Dazu sind Komponenten nützlich, die auch in Visual Basic benutzt wurden. Diese sind nicht Standard in der Toolbox und man muss sie erst aktivieren. Das geht über das Menü „Extras  Toolboxelemente auswählen“. Im kleinen Check-Kästchen markieren: FileListBox, DirListBox und DriveListBox. Sie sind zwar schon etwas altbacken, aber die notwendigen Funktionen für unseren Zweck stellen sie zur Verfügung. Meines Wissens will .NET diese Komponenten nicht mehr unterstüzen, es gibt aber auch keine Alternative: Danke liebe MS-Informatiker!

Die Plazierung der Komponenten soll jetzt so geschehen, dass ein automatisches Resizing erfolgt, und der User selbst die Aufteilung noch mit der Maus per Splitter verändern kann. Also folgendes Vorgehen ist zu tun:
FileListBox auf neue Seite eines TabControl, Dock=Left.
Dann Komponenten Splitter (Alle Windows Forms), voreingestellt ist schon passende Dock=Left.
Mit einem Panel den Rest der Seite mit Dock=Fill ausfüllen.
Auf dieses Panel dann eine DirListBox plazieren, Dock=Top. Darunter auf das Panel dann die Komponente DriveComboBox.
Die drei File-system- Kompos müssen noch verbunden werden. Das geht leider nicht mehr wie bei Borland mit einfachem Setzen der Eigenschaften im Eigenschaftenfenster, sondern nur noch programmiert z. B. in dem Ereignis Form_Activated. Warum einfach, wenn es auch kompliziert geht: Danke MS-Informatiker!

In dem Ereignis
private void dirListBox1_TextChanged(object sender, EventArgs e)
{
    dirListBox1.Path = driveListBox1.Drive;
}
private void dirListBox1_SelectedValueChanged(object sender, EventArgs e)
{
    if (dirListBox1.Text == null) 
        return;
    string selpath;
    int pos=dirListBox1.Path.IndexOf(dirListBox1.Text);
    if (pos > 0) 
        selpath = dirListBox1.Path.Substring(pos+dirListBox1.Text.Length);
    else 
        selpath = dirListBox1.Path + dirListBox1.Text;
    fileListBox1.Path = selpath;
}

File: Kurs MS10C# Bayerlein4.6.docx Seite 41
//pos is Position of text, if -1 not found
if (pos < 0)
    selpath = dirListBox1.Path + "\" + dirListBox1.Text;
else
    selpath = dirListBox1.Path.Substring(0, pos + dirListBox1.Text.Length);
fileListBox1.Path = selpath;
}

Will man an die Daten in der FileListBox drankommen, so geht das mit der Eigenschaft Items. Folgende for- Schleife kopiert alle Dateien in eine TextBox:

for (int i=0; i<fileListBox1.Items.Count; i++)
    textBox1.AppendText(fileListBox1.Items[i].ToString() + "\0d" + "\0a");

Jetzt soll ein kleines Vorschaufenster programmiert werden. Wenn man auf eine Datei in fileListBox1 klickt, soll in einer PictureBox eine Bildvorschau zu sehen sein. Also Plazieren eines PictureBox auf die Seite mit den obigen Komponenten. Größe 100*100 Pixel. In fileListBox1.SelectedItem steht die Nummer der selektierten (blau eingefärbten) Datei. Jetzt kann man in einem Event FileListBox1Click diese Bilddatei nach PictureBox1 laden. Syntax:

string fn = fileListBox1.Text; // selected Line
int len=fn.Length;
string ext = fn.Substring(len - 4, 4).ToLower();
if ((ext == ".jpg") || (ext == ".png") || (ext == ".bmp")
{
    pictureBox1.Load(fileListBox1.Path + "\" + fn);
}

es wird zuvor abgefragt, ob es sich um eine Bilddatei handelt. Die Methode .ToLower() wandelt alle Großbuchstaben in kleine um. Der Substring-Befehl schneidet die letzten vier zeichen heraus.

Der Dateiname wird mit fileListBox1.Text ausgelesen. Mit der Methode Load kann eine Bilddatei vom Typ *.ico, *.bmp oder *.jpg in die pictureBox1 gelesen werden.

Die Bilder werden jetzt alle nur mit 100*100 Pixel angezeigt, man sieht nur einen Ausschnitt. Das ganze Bild sieht man mit der Eigenschaft pictureBox1->SizeMode=StrechImage;

Die Originalbildgröße kann mit

labell1.Text = pictureBox1.Image.Height.ToString();
label2.Text = pictureBox1.Image.Width.ToString();

in zwei Label dargestellt werden. Das Bild ist zwar in pictureBox1 gestaucht, aber im Speicher noch in Originalgröße erhalten.

Will man jetzt noch das Seitenverhältnis erhalten, dann stelle man die Eigenschaft „SizeMode“ der pictureBox1 auf Zoom.

Will man jetzt noch bei Verstellung der Selektion mit den Up- oder Down- Tasten ein neues Bild sehen, dann wähle man in den Ereignissen der fileListBox1 „SelectedIndexChanged“ die schon existierende Ereigniskfunktion fileListBox1_Click. Testen Sie das Ergebnis. Wenn jetzt eine Bilddatei gewählt ist, wird sie in pictureBox1 angezeigt. Wenn man mit den Pfeiltasten die Selektion hoch und herunterbewegt, wird jeweils das aktuelle Bild angezeigt. Lädt man allerdings eine Datei mit einer anderen als der erlaubten Endung, passiert natürlich nichts. Siehe fertiges Programm in Kurs 6# Prepare HW1.
Projektaufgabe 1 Diashowprogramm


Hinweise:
Eigenschaften des neuen Formulars: Name="FormDia", FormBorderStyle=None; BackColor auf Schwarz. WindowState=wsMaximized. Achtung, bevor Sie es starten und mit Show() aufrufen, müssen Sie das Beenden programmiert haben (das ContextMenuStrip an das Formular binden), da das Fenster nun keine Tasten dafür mehr hat. Sonst geht es nur über Alt F4 oder andere Maßnamen, z. B. „Debugging beenden“ mit Shift + F5!
Beenden mit ContextMenuStrip- Menu. Ein Eintrag „end“, mit Doppelklick in der Ereignisfunktion Click() das Schließen des Fensters auslösen z.B. mit Close();. In dem Ereignis von FormDia namens FormClosing dann wie oben schon erwähnt den Befehl e.Cancel=true; programmieren, damit wird das Schließen und Löschen der Instanz verhindert. Stattdessen dort Hide(); programmieren.
Um den Zugriff von FormDia zurück auf das Hauptformular zu ermöglichen, muss man diese Schritte tun:
In Form1, dem Hauptformular:

```csharp
public Form1()
{
    InitializeComponent();
    frm2 = new FormDia(this);
}
FormDia frm2;
```

In FormDia:
Bei den using muss der Namespace von Form1 angegeben werden. In meinem Beispiel – kann bei ihnen anders heißen:-
```csharp
namespace WindowsApplication14
```
Also in Form Dia:
```csharp
using WindowsApplication14;
```
Wenn nicht der namespace von FormDia nicht sowieso schon den gleichen namen hat.
 Dann dort im Konstruktor die rot gerahmten Zeilen/ Teile ergänzen:
```csharp
Form1 meinmain;
public FormDia(Form1 m)
{
    InitializeComponent();
    meinmain = m;
}
```

Auf FormDia eine pictureBox- Komponente ziehen, diese mit Dock=fill auf das ganze Formular ausdehnen. Dann auch an pictureBox1 das ContextMenuStrip dranlinken.
Jetzt kann man in einem Ereignis MouseClick ein Bild aus der Liste von FileListBox laden.
Um an diese dranzukommen, muss sie erst im Modifier auf public gestellt werden.
Benötigt wird noch ein
```csharp
public int diacnt = 0;
```
mit dem der index der Filelistbox hochgezählt wird.
Das Laden erfolgt wie schon gehabt, alles hinter ereignis MouseClick:

```csharp
string fn = meinmain.fileListBox1.getItems(diacnt); // selected Line
int len = fn.Length;
string ext = fn.Substring(len - 4, 4).ToLower();
if ((ext == ".jpg") || (ext == ".png") || (ext == ".bmp"))
{
    pictureBox1.Load(meinmain.fileListBox1.Path + "\" + fn);
}
diacnt++;
if (diacnt >= meinmain.fileListBox1.Items.Count) diacnt = 0;
```

Jetzt sollte es schon gehen, nur in der FileListBox sollte das Pattern so gestellt werden, dass dort nur Bilder drin stehen, also =*.jpg;*.png;*.bmp.
Damit gleich nach Druck auf Start-Button ein Bild gezeigt wird, im Click- Ereignis des Buttons das MouseClick- Ereignis von FormDia aufrufen. Das geht erst, wenn man dieses Ereignis public macht:

```csharp
public
void FormDia_MouseClick(object sender, MouseEventArgs e)
```
dann in `buttonStartDia_Click(…..
```
frm2.FormDia_MouseClick(null, null);
```
Benötigt wird auf `FormDia` noch ein `Timer, Interval=2000, Modifier=public`, damit er von `Form1` aus erreicht und aktiviert werden kann.

`FormDia`, dort `pictureBox1.SizeMode=Zoom;`

Will man mit der ESC- Taste (Code 27) schließen, muss man auf dem Formular `FormDia` eine `KeyPress-` Ereignisfunktion erzeugen und dort

```csharp
if (e.KeyChar == 27) Visible=false ;
```

programmieren.

Nach Klick auf die Start- Taste muss der Timer bei aktivierter CheckBox für den Timer auf `Enabled=true` geschaltet werden.

Nun nur noch im Timer_Tick – Ereignis das mouseClick- Ereigniss aufrufen.
`FormDia_MouseClick(null, null);`

Beispiel des fertigen Programms in dem Verzeichnis HA1NeuV1, dort natürlich ohne Source.

Damit in der `FileListBox` nur Bilddateien aufgelistet werden, sollte vorher die Eigenschaft `.Pattern` von der `FileListBox` auf ein passendes Filter gesetzt werden. Das Format dafür:

`*.jpg`, wenn ein Typ zugelassen werden soll, wenn es mehrere sind, mit Semikolon trennen:

`*.jpg;*.gif;*.png;*.bmp`

Ein Problem sollte noch abgefangen werden: Wenn gar kein Bild in der `FileListBox` ist, gibt es eine Fehlermeldung, diese wird noch nicht abgefangen.

**Kurs 7 Zeichnen mit dem Graphics - Objekt**

**Das Graphics-Objekt**

Das Objekt `Graphics` dient zum pixelorientierten Zeichnen auf Komponenten als Ausgabeinheiten. Jeder einzelne Pixel kann individuell definiert werden, mit komfortablen Methoden ist ein Zeichnen von fast beliebigen Figuren (Linien, Rechtecken, Kreisen usw.)
möglicher. Auch Text kann der Zeichnung hinzugefügt werden. Mit diesem Objekt lässt sich ein eigenes Paint-Programm leicht selber schreiben.
Es gibt einige Komponenten, die ein Graphics-Objekt haben:
- Panel
- PictureBox
- Form
Will man auf dem Bildschirm eine Grafik erzeugen, so nehme man am besten direkt die Graphics der Form, will man später sein Werk als Bitmap speichern, so muss man erst eine Bitmap definieren, doch dazu später. Diese Bitmap verfügt dann über die Methoden Load() und Save().
Also platzieren wir eine PictureBox auf eine neue Form einer neuen Anwendung. Oben wird wieder ein Panel1 und Dock=Top als Bereich für Tasten definiert, die pictureBox1 füllt den Rest der Form (Dock=Fill).
Einige Beispiele:
erzeugt von folgendem Programm.

```csharp
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    SolidBrush pbrush = new SolidBrush(Color.Blue);
    e.Graphics.FillRectangle(pbrush, 5, 5, 100, 100);
}
```


```csharp
Pen p = new Pen(Color.Red, 3);
Point poi1 = new Point(0, 5);
Point poi2 = new Point(80, 15);
e.Graphics.DrawLine(p, poi1, poi2);
```


Oder mal in einem StatusStrip1:

```csharp
Pen p = new Pen(Color.Green, 1);
Rectangle rec = new Rectangle(200, 3, 250, 15);
e.Graphics.DrawRectangle(p, rec);
```

In DrawRectangle steht mit (200,3) die Koordinate oben links und dann folgt Width und Height.
Allen diesen Figuren werden jetzt sofort zu Anfang beim Start gezeichnet, denn dann wird der Paint- Ereignis ausgelöst. 
Will man in einem anderen Ereignis zeichnen, z.B. in einem Click des Buttons, so gibt es natürlich die PaintEventArgs nicht. Dann kann der Bezug folgendermaßen erzeugt werden:

```csharp
private void buttonLine_Click(object sender, EventArgs e)
{
    Graphics gPB = Graphics.FromHwnd(pictureBox1.Handle);
    SolidBrush pbrush = new SolidBrush(Color.LightSkyBlue);
    gPB.FillRectangle(pbrush, 0, 0, 100, 100);
}
```

es wird eine lokale Instanz von Graphics (gPB) erzeugt, die per Handle auf die pictureBox1 zeigt. Der Rest geht wie gehabt. Jetzt wird auf Click ein Rechteck gezeichnet. Die erste Zeile mit der Methode .FromHwnd kann auch völlig gleichbedeutend mit der Alternativcodierung ersetzt werden. Beides erfüllt den gleichen Zweck:

```csharp
Graphics gPB = pictureBox1.CreateGraphics();
```

Bei einer solchen lokalen Instanz sollte man nach dem Zeichenvorgang die Graphics- Instanz gPB wieder entfernen mit

```csharp
gPB.Dispose();
```

Die Instanzierung von gPB kann auch global erfolgen, dann können andere Ereignisfunktionen auch darauf zugreifen:
Erst global- also außerhalb einer function, deklarieren:

```csharp
public partial class Form1 : Form
{
    Graphics gPB;
    SolidBrush pbrush = new SolidBrush(Color.Blue);
    .......

    Dann instanzieren:

    public Form1()
    {
        InitializeComponent();
        gPB = Graphics.FromHwnd(pictureBox1.Handle);
        pbrush = new SolidBrush(Color.LightSkyBlue);
    }
```

Dann benutzen, z. B. mit zwei Trackbars wird ein Pie gezeichnet, dessen Umrandung eine Ellipse ist und die Randbegrenzungen werden mit einem Winkel (0-360°) definiert, wobei 0° nach rechts zeigt, von dort startet ein Kuchenstück mit der Winkelbreite eingestellt mit trackBar2:

```csharp
private void trackBar1_Scroll(object sender, EventArgs e)
{
    pbrush.Color=Color.Red;
    gPB.FillPie(pbrush, 20, 30, 100, 200, trackBar1.Value, trackBar2.Value);
}
```

Links steht trackBar1 etwa auf 5° und trackBar2 auf 90°

Nun wird direkt auf die Komponenten gezeichnet. Will man eine Zeichnung erzeugen, die man abspeichern kann, so muss man mit einer Bitmap
arbeiten.

Der erste Versuch bietet sich an: Man nehme die fertige BitMap von *PictureBox1* namens *Image*.

Man kann zur Laufzeit ein Bild wie gehabt laden mit

```csharp
pictureBox1.Load("Wolken.BMP");
```


Will man jetzt das Bild mit Zeichnung abspeichern, so stellt man erstaunt fest, dass *pictureBox1* keine Save- Methode hat. Man findet aber bei *Image* ein .Save, nur das funktioniert nicht, warum weiß ich zurzeit nicht:

```csharp
pictureBox1.Image.Save("test.bmp"); // ➔ Fehlermeldung
```

Mal ein Versuch mit einer Bitmap, z. B. so:

```csharp
Bitmap pic = new Bitmap(200, 200);
```

erzeugt eine leere 200*200 Pixel große Bitmap,

```csharp
Bitmap pic = new Bitmap("Blumen.jpg");
```


Wie zeichnet man nun in eine Bitmap und speichert sie dann folgendermaßen ab: Z.B. in einer Click-Funktion könnte man schreiben:

```csharp
Graphics g1 = Graphics.FromImage(pic);
Rectangle rec = new Rectangle(0, 3, 25, 15);
g1.FillRectangle(pbrush, rec);
```

Jetzt wird ein Rechteck nach g1 gezeichnet, nur man sieht nichts. Man kann es aber abspeichern, z. B. mit folgender Zeile:

```csharp
```

Mit dem *ImageFormat* teilt man der *Save*- Methode mit, in welchem Format man es denn gerne hätte. Als .jpg- Datei geht dann so:

```csharp
```

Wenn man sich diese Dateien mit Windows-Programmen ansieht, dann ist das Rechteck zu erkennen, der Hintergrund ist aber schwarz, wenn man mit Bitmap(200,200) eröffnet hat, sonst eben das geladene Bild. Will man die Zeichnung sehen, so muss man in ein Graphics- Object zeichnen, das zu einer sichtbaren Komponente gehört, z.B. unser gPB, das auf die *pictureBox1* zeichnet. Der Code dazu lautet:

```csharp
gPB.DrawImage(pic, 10, 10);
```
wobei die beiden Koordinaten in diesem Aufruf den linken oberen Punkt von `pic` festlegen.

Will man jetzt `g1` von oben nicht auf eine Bitmap, sondern auf das Image-Object von `pictureBox1` legen, so geht das leider auch nicht: Laufzeitfehler

```csharp
Graphics g1PB = Graphics.FromImage(pictureBox1.Image); // ->Laufzeitfehler
```

Man kann nun Zeichenfiguren mit verschiedenen Methoden erzeugen. Die wichtigsten sind:

- `DrawLine(int X, int Y, int x1, int y1);` zeichnet Linie von (X,Y) bis (X1,Y1). Mit 4 verschieden überladenen Parameterlisten, siehe CodeCompletion-Fenster.
- `DrawRectangle(int X1, int Y1, int X2, int Y2);` Zeichnet ein Rechteck mit Ecke oben links bei X1,Y1 und unten rechts bei X2,Y2. Mit 3 verschiedenen überladenen Parameterlisten, siehe CodeCompletion-Fenster. `FillRectange` ähnlich mit Füllung
- `DrawEllipses(int X1, int Y1, int X2, int Y2);` Zeichnet Kreis oder Ellipse mit umgebenem Rechteck mit den Koordinaten wie beim Rectangle.
- `DrawPie(int X1, int Y1, int X2, int Y2, int Y3, int X4, int Y4);` s.o.
- Weitere Methoden von `Graphics` siehe CodeCompletion


Wir brauchen vier Tasten mit Text Artwork, Save Pic, Pen-Color und Fill-Color. Zum Abspeichern ist ein `SaveDialog1` nötig, für die Farbwahl ein `ColorDialog`, für die Strichdicke
brauchen wir noch ein `numericUpDown`-Element, Default Wert für `Value=1`. Das sieht dann so aus:

![Artwork][1]  [Save][2]  [Pen Col][3]  [Fill Col][4]  [1]

Die Komponenten haben folgende Ereignisfunktionen:


```csharp
if (timer1.Enabled) timer1.Enabled = false; else timer1.Enabled = true;
```

**Save**(Button2), es wird als .jpg abgespeichert:

```csharp
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
```

**PenColor und BrushColor**: Hinter Button3 und Button4 wird die Farbe des Stiftes und des Pinsels eingestellt, p und pbrush sind globale Instanzen von Pen und SolidBrush:

```csharp
if (colorDialog1.ShowDialog() == DialogResult.OK)

if (colorDialog1.ShowDialog() == DialogResult.OK)
pbrush.Color = colorDialog1.Color;
```

Mit dem **OnChange**-Event wird die Stiftdicke gewählt:

```csharp
private void numericUpDown1_ValueChanged(object sender, EventArgs e)
{
    p.Width = (int)numericUpDown1.Value;
}
```

Zuerst jetzt die Freihandmalfunktion. Dazu lernen wir die Mausevents kennen. Das Zeichnen selbst geschieht mit dem Ereignis `MouseMove`, der Aufruf sieht so aus:

```csharp
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
    if (draw)
    {
        Point endp = e.Location;
        gPic.DrawLine(p, start, endp);
        gPB.DrawImage(pic, 0, 0, pic.Width, pic.Height);  // Draw on screen
        start = endp;
    }
}
```

Die aktuellen Mouse Koordinaten werden in `e` übergeben, z.B. in `e.X` und `e.Y` oder in `e.Location`. Diese Funktion wird jedes Mal aufgerufen, wenn die Maus sich bewegt. Also wird auch gezeichnet, wenn man keine Maustaste drückt, also immer. Deswegen gibt es nun eine Steuerung mit der bool- Variablen `draw`. Sie wird im `MouseMove`- Ereignis auf `true` und im `MouseUp`- Ereignis zurück auf `false` gesetzt.

Dann wird wie oben beschrieben erst mit `gPic` in die Bitmap `pic` gezeichnet und dann mit `gPB` in `pictureBox1`, damit es auf den Bildschirm gelangt.

Es wird immer eine Linie von `start` nach `endp` gezeichnet, wenn `draw true` ist. Deswegen muss jetzt noch im Ereignis `MouseDown` die globale Variable `start` auf die aktuellen Mauskoordinaten gesetzt werden mit:
private void pictureBox1_MouseDown(object sender, MouseEventArgs e)  
{
    draw = true;
    start = e.Location;
}

Ein paar Bemerkungen:
- Die Zeichenfläche wird beim ersten Zeichnen festgelegt. Ändert man sie hinterher, findet keine Größenänderung mehr statt.
- Lädt man als Hintergrund das Bild „Wolken.BMP“, so gibt es einen Laufzeitfehler. Das liegt daran, weil dieses Bild ein 256-Index-Farben-Bild ist. Wandelt man es in truecolor, so geht es.
- Das Bild darf nicht auf SizeMode =StrechImage; gestellt werden, da dann die Mauskoordinaten nicht mit der Position übereinstimmen.

Nun das Artwork. Im Timer sollen per Random-Generator farbige Rechtecke erzeugt werden. In der Timer- Funktion steht bei mir:

    Random r = new Random();

    int x1 = r.Next(0, pic.Width); //x point top left
    int y1 = r.Next(0, pic.Height); // y point top left
    int x2 = r.Next(0, pic.Width-x1); // width
    int y2 = r.Next(0, pic.Height-y1); // height

    Rectangle rect = new Rectangle(x1, y1, x2, y2);

    int red = r.Next(0, 255);
    int green = r.Next(0, 255);
    int blue = r.Next(0, 255);
    //int alpha = r.Next(0, 255); // this value with trackbar1
    pbrush.Color = Color.FromArgb(alpha, red, green, blue);

    int red2 = r.Next(0, 255);
    int green2 = r.Next(0, 255);
    int blue2 = r.Next(0, 255);

    p.Color = Color.FromArgb(alpha, red2, green2, blue2);
    p.Width = r.Next(1, 10);

    g1pic.FillRectangle(pbrush,rect);
    g1pic.DrawRectangle(p, rect);
    gPB.DrawImage(pic,0,0, pic.Width, pic.Height);//Draw on screen
Die Rechtecke orientieren sich an der Größe von pic. Stellt man *Interval* vom *timer1* auf 100 ms, dann werden alle Sekunde 10 Rechtecke erzeugt.

Die Farben werden mit `.FromArgb()` erzeugt. Schön ist auch, dass der Transparentwert alpha hier auch geändert werden kann. alpha=0 heißt voll transparent, also unsichtbar, alpha = 255 nicht transparent. Im fertigen Programm verändert man alpha mit dem linken *trackBar1*. Das Programm zeichnet erst die Füllung und dann einen Rahmen mit anderer Farbe.

Die fertigen Dateien finden Sie in *Kurs7#Graphics*.

Viel Spaß bei weiteren Experimenten.
Kurs 8 Chart2D

Siehe Hinweis auf die in DotNet 4.0 neu verfügbare Komponente Chart ab S. 85. In diesem und dem nächsten Kurs wird die mächtige Komponente Chart2D vorgestellt. Diese Komponente ist an der FHL neu entwickelt worden und vollständig in C# geschrieben. Sie ersetzt die alte bisher seit Jahren benutzte Komponente Chartfx, die bei Borland Verwendung fand. Damit ist der ganze Ärger mit ActiveX usw. endlich vom Tisch.


![Chart2D Beispiel](image)


Die Datei `Verwendung der Komponente Chart2D in C_2.pdf` beschreibt für den Programmierer, wie man diese Komponente einbindet in seinen eigenen C#- Code. Deswegen soll hier nur kurz beispielhaft die Verwendung beschrieben werden, weitere Details entnehme man diesen Dokumenten.

Zur Installation benötigt man die Datei Chart2D.vsi. Diese darf nur bei ausgeschaltetem Visual Studio gestartet werden. Danach sollte die Komponente in der Toolbox erscheinen. Das Ergebnis bei MS Studio 2010 sieht so aus:
Erstes kleines Beispiel:
In einem neuen Projekt erst ein Panel mit Dock Top und dann eine Komponente auf die Form ziehen mit Dock Fill.
Dann einen Button mit Namen ButtonSin mit text sin ergänzen. Hinter dem Clickereignis dieser Code:

```csharp
int i, nb = 100, serie = 1;
for (i = 0; i < nb; i++)
{
    chart2D1.setPoint(0, i, Math.Sin(i / 16.0 * serie));
}
chart2D1.startDrawing();
chart2D1.autoScaling();
```

Es sieht nach Ausführung dann so aus:

Mit Setpoint wird das interne dynamisch verwaltete Datenarray gefüllt. Die erste Zahl ist die Nummer der Kurve (von 0 bis 9), die zweite der x-Wert und die dritte Zahl der y-Wert eines Punktes. Mit dem Befehl startDrawing() wird die Kurve gezeichnet. Mit autoScaling() findet eine automatische Skalierung statt.
Die Anzahl der Punkte pro Kurve ist beliebig und kann für jede Kurve individuell verschieden sein.
Drückt man diese Taste sin jedoch ein zweites Mal, so erscheint eine blauer Querstrich. Das liegt einfach daran, dass jetzt zwei gleiche Kurven im Speicher hintereinander liegen und alle
Punkte verbunden werden. Also wird der letzte Punkt der ersten Sin-Kurve mit dem ersten Punkt der zweiten verbunden.
Um dies zu vermeiden, sollte vor dem Neuzeichnen erst alter Inhalt einer Kurve mit der Methode clearLast() gelöscht werden.
Eine einfache Variation liegt jetzt hinter einer neuen Taste namens buttonMulti. Folgender Codezeichnet immer neue Kurven:
Außerhalb einer function: short serie = 1;
Dann innerhalb der Click-funktion:
    int i, nb = 100;
    for (i = 0; i < nb; i++)
    {
        chart2D1.setPoint(serie - 1, i, Math.Sin(i / 16.0 * serie));
    }
    chart2D1.startDrawing();
    chart2D1.autoScaling();
    serie++;
Nach der 10. ten Kurve wird nicht mehr gezeichnet.
Jetzt testen wir mal die Echtzeitzeichnungsfunktion: Hinzufügen eines Timers (Interval=20, Enabled=false) und einer Taste ButStart.
Hinter diese Taste:
    if (timer1.Enabled)
        timer1.Enabled = false;
    else
    {
        timer1.Enabled = true;
        chart2D1.clearAll();
        chart2D1.XMax = 100;
        chart2D1.XGap = 10;
        chart2D1.YMax = 1;
        chart2D1.YMin = -1;
        chart2D1.YGap = 0.2;
        serie = 1;
    }
Außerhalb: int itemp = 0;
In timer1_Tick:
    chart2D1.realTimeChart((serie - 1), itemp, Math.Sin(itemp / 16.0 * serie));
    itemp++;
    if (itemp > 100)
    {
        itemp = 0;
        serie++;
    }
    if (serie > 5)
    {
        serie = 1;
        chart2D1.clearAll();
    }
 Dann werden nacheinander 5 Kurven in Echtzeit gezeichnet:
Jetzt noch einige einfache Codebeispiele:
Änderung eines Kurvennamens:
```csharp
chart2D1.CurveNames[0] = "Kurve 1";
chart2D1.updateNames();
```

Titel: `chart2D1.TopTitle = "Top Header Title";`
Löschen der letzten Kurve: `chart2D1.clearLast();`
Ausblenden z.B. der ToolTaste“ Lösche alle Kurven“: `chart2D1.AllClearButton = false;`
Abfrage der Anzahl von Kurven: `Counter, keine Kurve: -1`
Abfrage der Punkte pro Kurve: in `CurvePoints[]`, es ist ein int- Array.
Abfrage von Punktkoordinaten: einmal `getPoint(i,j)`, wobei i die Kurvennummer und j die
Punktnummer ist beide startend mit Null. Beispiel:
```csharp
double[] xy;
xy = chart2D1.getPoint((int)nudKurve.Value, (int)nudPunkt.Value);
labelXval.Text = xy[0].ToString();
labelYval.Text = xy[1].ToString();
```

Natürlich dürfen die Werte in i und j die Arraygrößen nicht überschreiten, sonst Fehler. i und j
werden im Beispiel mit je einem NumericalUpDown eingestellt. Der Typ von Value ist dort ein
decimal, deswegen das „typecast“ mit `(int).

Will man übrigens eine Grafikdarstellung mit der alten chartFX in C# (Balken,
Kuchendiagramme auch in 3D) so lese man diesen Skript in einer Version 3. Ab Version 4 ist
sie verschwunden, da die dort benutzte ActiveX große Installationsprobleme verursacht und
außerdem ein 64-Bit programmierung verhindert. Ich bin froh, dass ich sie los bin.
Kurs 9 DataTools - Einbinden von Tool- Units

In meinem Regelungstechnik- Tool- Programm WindfC# benutze ich Units zu finden dort in DataTools.cs, mit deren Hilfe man z.B. den Inhalt einer Textbox, falls die ein bestimmtes Format aufweist, komplett als Kurve darstellen kann. Das Format sollte so sein, dass pro Punkt die Koordinaten x und y in einer Zeile stehen und mit einem Separator (kann Komma, Tab oder Leerzeichen sein) getrennt sind.

Vorgehensweise ähnlich schon wie im Kurs 2c#:
Neues Projekt nach Kurs 9# abspeichern.
Der Form ein Panel mit Dock Top hinzufügen.
Zweites Panel mit dock Left hinzufügen
Chart2D mit Dock Fill hinzufügen.
Auf panel1 ButtonTB2Chart hinzufügen mit Text TB to Chart
Auf panel2 Textbox hinzufügen mit Dock Fill.
Resultat:

Nun folgende Instanzen erzeugen (außerhalb des Konstruktors von Form1):
```csharp
DataTools dt = new DataTools();
LoadOpt lo = new LoadOpt();
CHMinMax mm = new CHMinMax();
```

Im Konstruktor dann:
```csharp
public Form1()
{
    InitializeComponent();
    lo.init();
    mm.init();
}
```

Hinter der Clickfunction des Button folgenden Code:
```csharp
dt.ConvertTB2ChartAdd(ref chart2D1, textBox1, ref mm, lo);
```
Jetzt mal folgenden Inhalt in die rechte Textbox tippen oder kopieren:

0 0
1 1
2 3
3 5
4 3
5 0

Ergebnis:

Nun kann man ganze Dateien mit Messwerten laden und so einfach grafisch anzeigen. Dazu ein Load-Button spendieren, noch einen openFileDialog1 hinzufügen und hinter die Clickfunctions des Loadbuttons dann wie schon oben in Kurs 5# erklärt eine Datei laden mit:

```csharp
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    StreamReader myStream1 = new StreamReader(openFileDialog1.OpenFile(), Encoding.Default);
    textBox1.Text = myStream1.ReadToEnd();
}
```

Als Beispiel ist im Kurs 9 die Datei 2PT1_2.sim zu finden. Die Grafik sieht dann so aus:

In den LoadOptions kann man dann auch andere Dateiformate, logarithmische Skalierung etc voreinstellen.
Kurs 9a Zeichnen von Kurven mit Chart DotNet 4.0

Seit DotNet 4.0 gibt es nun endlich eine Komponente zum Zeichnen beliebiger Grafiken von Microsoft: Die Komponente Chart: Man findet sie in der Toolbox unter Daten:

Wenn man diese auf das Formular zieht, sieht es so aus:

Die Befüllung mit Daten geschieht so (Sinuskurve mit 100 Punkten):

```csharp
int i, nb = 100, serie = 1;
chart1.Series[0].ChartType = SeriesChartType.Line;
for (i = 0; i < nb; i++)
{
    chart1.Series[0].Points.AddXY(i, Math.Sin(i / 16.0 * serie));
}
```

Will man eine neue Kurve (genannt serie) hinzufügen, so muss man dies machen:

```csharp
chart1.Series.Add("Kurve 1");
chart1.Series[1].ChartType = SeriesChartType.Line;
```


Nur leider gibt es die Toolbar nicht, die dem User bei einem fertigen Programm auch Möglichkeiten gibt, seine Darstellung während der Laufzeit zu ändern. Allerdings sind die Darstellungsmöglichkeiten dieser Chart – Komponente gigantisch. Will man nicht nur Kurven, sondern beliebige Grafiken darstellen, so muss man sich mit dieser sehr komplexen
Komponenten auseinander setzen. Eine sehr gute Einführung bekommt man mit einem Tutorial, das ich bei Microsoft gefunden habe und es auf studpub zur Verfügung gestellt habe: Siehe baye
erle\ MS Studio2010 C# WS2012_13\ DotNetChart\ Entpackt man die dort zu findende Datei WinSamplesChart von MSDN(1).zip, so kann man das Programm WinFormsChartSamples.exe starten und bekommt eine Einführung in alle Möglichkeiten mit passenden Source – Code Zeilen. Besser kann man ein Tutorial nicht aufbauen und ich kann dem nichts mehr hinzufügen. Ich werde sicher auch demnächst mein Toolprogramm Windfc# damit ausstatten, da insbesondere auch Bodediagramme damit sehr gut darstellbar werden, was bisher nicht ging. Folgende Tabelle soll Entscheidungshilfe geben, welche Komponenten man wann nehmen könnte:

<table>
<thead>
<tr>
<th>Chart2D von Herrn Seckler</th>
<th>Chart in DotNet 4.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Code verfügbar und dadurch anpassbar an eigene Wünsche</td>
<td>Anpassbar durch tausende Optionen</td>
</tr>
<tr>
<td>Toolbar für User während Laufzeit</td>
<td>Keine Toolbar</td>
</tr>
<tr>
<td>Einfaches Interface</td>
<td>Komplexes Interface</td>
</tr>
<tr>
<td>Nur Kurven, max. 10</td>
<td>Beliebige Grafiken</td>
</tr>
<tr>
<td>Keine Bodediagramme</td>
<td>Bodediagramme möglich</td>
</tr>
</tbody>
</table>


**Kurs 10 Weitere Komponenten**

**WebBrowser**

Mit der Komponente WebBrowser (in Allgemeine Steuerelemente) kann man sich seinen eigenen Browser basteln. In Anwendungen kann man dann damit auf seine eigene Homepage verweisen oder Hilftexte anbieten. Da in der Regel ein pdf- Reader installiert ist, kann man mit dieser Komponente dann auch pdf- Dateien in seiner Anwendung anzeigen. Wir starten mit einer neuen Anwendung und platzieren in der oberen Hälfte ein Panel (Dock auf Top) für die Bedienelemente unseres Browsers und darunter mit Dock auf Fill den WebBrowser. Mit einer textBox1 und einigen Buttons lässt sich ganz einfach ein Browser zusammenstellen.

```csharp
Hinter der Go- Taste: webBrowser1.Navigate(textBox1.Text);
Hinter der Forward- Taste: webBrowser1.GoForward();
Hinter der Back- Taste: webBrowser1.GoBack();
Hinter der Home- Taste: webBrowser1.GoHome();
```

File: Kurs MS10C# Bayerlein4.6.docx

Seite 59
Hinter der Browse- Taste:

```csharp
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    webBrowser1.Navigate(openFileDialog1.FileName);
}
```

Die Home-Seite ist die, die beim Internet-Explorer als Home definiert ist, die Browse-Taste setzt eine Komponente `openFileDialog1` voraus. Es läuft im Hintergrund wie zu erwarten eine eingebettete Form des auf dem PC installierten IE in der entsprechenden Version. Lädt man mit der Browse-Taste Dateien der Art pdf, doc, jpg usw., so wird je nach Installation auf dem Rechner entweder die Anwendung selbst geöffnet, die der Datei zugeordnet ist oder ein Viewer im IE wird aufgerufen und man verlässt die Anwendung nicht.

**Mediaplayer**


```csharp
using System.Media;
```

Hat man seinem Projekt eine Dialogkomponente `openFileDialog1` hinzugefügt, dann kann man mit folgendem Code eine *.wav-Datei laden und abspielen:

```csharp
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    SoundPlayer player = new SoundPlayer();
    player.SoundLocation = openFileDialog1.FileName;
    player.Load();
    player.Play();
}
```

mp3 oder videos gehen aber nicht.

Die gehen aber mit einem eingebetteten Widows-Media-Player. Dann kann man alles machen, was der Media-Player auch kann: Vorgehen:

1. Toolboxelemente auswählen ➔ COM, dort WindowsMediaPlayer suchen und wählen.
2. Dann Komponente Windows Media Player in Toolbox suchen, bei mir ganz unten, auf Form ziehen.
3. Dann laden einer Media- Datei and abspielen mit

```csharp
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    axWindowsMediaPlayer1.URL = openFileDialog1.FileName;
    axWindowsMediaPlayer1.Ctlcontrols.play();
}
```

Der spielt insbesondere auch mp3, aber auch Videos ab.
Kurs 11 Email- Programm


Kurs 12 TreeView

Wir öffnen ein neues Projekt (ProTreeView) und platziert ein Panel (Dock Top) einen TreeView (Dock Left) und eine FileListBox (erst über Menü „Extras -> Toolboxelemente auswählen“ hinzufügen, wenn nicht in ToolBox bereits verfügbar) mit Dock Fill.
Die Ideen zu diesem Projekt hab ich von Andreas Küehnel übernommen.

Hinter einem Button wird nun der folgende Text gelegt:

```csharp
string[] drives = Directory.GetLogicalDrives();
TreeNode node;
foreach (string drv in drives)
{
    node = treeView1.Nodes.Add(drv);
    if (drv == "C:\") // C: selektieren
        treeView1.SelectedNode = node;
}
```

durchgearbeitet. Mit `Add(drv)` werden Einträge hinzugefügt, in `drv` stehen die einzelnen Elemente des Stringarays.
Nach Einlesen wird noch der Node mit C: selektiert.


```csharp
private void ReadSubDirs(TreeNode node)
{
    DirectoryInfo[] arrDirInfo;  // Array für Subdirectories
    DirectoryInfo dirinfo = new DirectoryInfo(node.FullPath);
    try
    {
        arrDirInfo = dirinfo.GetDirectories();
    }
    catch
    {
        return;
    }
    foreach (DirectoryInfo DI in arrDirInfo)
    {
        node.Nodes.Add(DI.Name);
    }
}
```


```csharp
public Form1()
{
    InitializeComponent();
    treeView1.Sorted = true;
    string[] drives = Directory.GetLogicalDrives();
    TreeNode node;
    foreach (string drv in drives)
    {
        node = treeView1.Nodes.Add(drv);
        ReadSubDirs(node);
        if (drv == "C:\")  // C: selektieren
            treeView1.SelectedNode = node;
    }
}
```
Außerdem wird die Sortieren- Eigenschaft von TreeView aktiviert. Nach Programmstart sieht es dann so aus, die „plus“ Kästchen öffnen dann Unterverzeichnisse der ersten Baumebene. Jetzt kann man im Click- Ereignis von TreeView alle Dateien des selektierten Pfades in ListBox anzeigen lassen mit

```csharp
private void treeView1_Click(object sender, EventArgs e)
{
    TreeNode node;
    node = treeView1.SelectedNode;
    listBox1.Path = node.FullPath;
    listBox1.Update();
}
```

 Dann sieht man rechts die Dateien der selektierten Pfade. Nur ganz zuverlässig funktioniert es nicht, denn manchmal muss man zweimal klicken. Der Ereignis scheint nicht der richtige zu sein. Besser ist der Ereignis:

```csharp
private void treeView1_NodeMouseClick(object sender, TreeNodeMouseClickEventArgs e)
{
    listBox1.Path = e.Node.FullPath;
    listBox1.Update();
}
```
dann wird in e der angeklickte Knoten übergeben. Dann funktioniert es sicher. Jetzt fehlen noch zwei Dinge, dann ist der Miniexplorer fertig. Bei jedem Öffnen eines Knotens sollten die Unterverzeichnisse dieses Knotens eingelesen werden, so dass man sich durch den gesamten Baum bewegen kann. Das geschieht in dem folgenden Ereignis so:

```csharp
private void treeView1_BeforeExpand(object sender, TreeViewCancelEventArgs e)
{
    foreach (TreeNode node in e.Node.Nodes)
    {
        ReadSubDirs(node);
    }
}
```

Bei mir sieht die imageList1 so aus:

Jetzt bekommen schon mal alle Einträge das Icon mit Nummer 0. Will man bei geöffnetem Directory das grüne Icon anzeigen, dann muss man allen Knoten, die man mit Add hinzufügt, mit

```csharp
e.Node.SelectedImageIndex = 1;
```

dieses mitteilen. Ich hab diese Zeile im Click-Ereignis ergänzt.
Kurs 13 Drucken
Diesen Kurs habe ich mehr oder weniger direkt von Andreas Kühnel [1], Beispiel 18 übernommen. Was muss man tun:

1. Hinzufügen der Print-Komponenten

![PrintDialog](image)

2. using System.Drawing.Printing;
3. Im Konstruktor:printDialog1.PrinterSettings = new PrinterSettings();
4. Deklaration folgender Variablen:

   ```csharp
   private int startSeite;       // Anzahl der Druckseiten
   private int anzahlSeiten;     // aktuelle Seitenzahl
   private int seitenNummer;     // zu druckender Text
   private string strPrintText;  // der Name der geöffneten Datei
   private string strDateiName;
   ```

5. In einer Druck- Funktion (hinter einem Button Druck oder so):

   ```csharp
   if (printDialog1.ShowDialog() == DialogResult.OK)
   {
       printDocument1.DocumentName = "Dokument";
       // Startwerte abhängig vom zu druckenden Text initialisieren
       switch (printDialog1.PrinterSettings.PrintRange)
       {
           case PrintRange.AllPages:
               strPrintText = textBox1.Text;
               startSeite = 1;
               anzahlSeiten = printDialog1.PrinterSettings.MaximumPage;
               break;
           case PrintRange.SomePages:
               strPrintText = textBox1.Text;
               startSeite = printDialog1.PrinterSettings.FromPage;
               anzahlSeiten = printDialog1.PrinterSettings.ToPage-startSeite+1;
               break;
       }
       // Drucken starten
       seitenNummer = 1;
       printDocument1.Print();
   }
   ```


   ```csharp
   StringFormat stringFormat = new StringFormat();
   RectangleF rectFPapier, rectFText;
   int intChars, intLines;

   // Ermitteln des Rectangles, das den gesamten Druckbereich
   // beschreibt (inklusive Kopf- und Fusszeile)
   rectFPapier = e.MarginBounds;

   // Ermitteln des Rectangles, das den Bereich für den
   // Text beschreibt (ausschließlich Kopf- und Fusszeile)
   rectFText = RectangleF.Inflate(rectFPapier, 0, -2 *
   textBox1.Font.Inflate(rectFText.Height, e.Graphics));

   // eine gerade Anzahl von Druckzeilen ermitteln
   int anzahlZeilen = (int)Math.Floor(rectFText.Height /
   textBox1.Font.GetHeight(e.Graphics));
   ```
// die Höhe die textbeinhaltenden Rechtecks festlegen, damit die
// letzte Druckzeile nicht abgeschnitten wird
rectFText.Height = anzahlZeilen * textBox1.Font.GetHeight(e.Graphics);

// das StringFormat-Objekt festlegen, um den Text in einem Rechteck
// anzuzeigen - Text bis zum nächstliegenden Wort verkürzen
stringFormat.Trimming = StringTrimming.Word;

// legt die Druckstartseite fest, wenn es sich nicht um die
// erste Dokumentenseite handelt
while ((seitenNummer < startSeite) && (strPrintText.Length > 0))
{
    e.Graphics.MeasureString(strPrintText, textBox1.Font,
        rectFText.Size, stringFormat, out intChars, out intLines);
    strPrintText = strPrintText.Substring(intChars);
    seitenNummer++;
}

// Druckjob beenden, wenn es keinen Text zum Drucken mehr gibt
if (strPrintText.Length == 0)
{
    e.Cancel = true;
    return;
}

// den Text an das Graphics-Objekt übergeben
e.Graphics.DrawString(strPrintText, textBox1.Font, Brushes.Black, rectFText, stringFormat);

// Text für die nächste Seite
// intChars - Anzahl der Zeichen in der Zeichenfolge
// intLines - Anzahl der Zeilen in der Zeichenfolge
e.Graphics.MeasureString(strPrintText, textBox1.Font, rectFText.Size,
    stringFormat, out intChars, out intLines);
strPrintText = strPrintText.Substring(intChars);

// StringFormat restaurieren
stringFormat = new StringFormat();

// Dateiname in der Kopfzeile anzeigen
stringFormat.Alignment = StringAlignment.Center;
e.Graphics.DrawString(this.strDateiName, textBox1.Font, Brushes.Black, rectFPapier, stringFormat);

// Seitennummer in der Fusszeile anzeigen
stringFormat.LineAlignment = StringAlignment.Far;
e.Graphics.DrawString("Page " + seitenNummer, textBox1.Font, Brushes.Black, rectFPapier, stringFormat);

// ermitteln, ob weitere Seiten zu drucken sind
seitenNummer++;
e.HasMorePages = (strPrintText.Length > 0) && (seitenNummer < startSeite + anzahlSeiten);

// Neuinitialisierung
if (!e.HasMorePages)
{
    strPrintText = textBox1.Text;
    startSeite = 1;
    anzahlSeiten = printDialog1.PrinterSettings.MaximumPage;
    seitenNummer = 1;
}
Zuvor müssen noch bei den Komponenten `printDialog1`, `printPreviewDialog1` und `pageSetupDialog1` die Eigenschaft `Document` auf `printDocument1` gestellt werden.

**Kurs 14 DLL**


Dazu zuerst einmal der Hinweis auf die unterschiedliche Bezeichnung der Variablentypen in C/C++ und C#. Hier ein Auszug aus Frank Eller [3]: C#-**Typen**

<table>
<thead>
<tr>
<th>Datentyp</th>
<th>Größe</th>
<th>Wertebereich</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>3 Bit</td>
<td>true, false</td>
<td>System.Boolean</td>
</tr>
<tr>
<td>byte</td>
<td>8 Bit</td>
<td>0 bis 255</td>
<td>System.Byte</td>
</tr>
<tr>
<td>sbyte</td>
<td>8 Bit</td>
<td>-128 bis +127</td>
<td>System.SByte</td>
</tr>
<tr>
<td>char</td>
<td>16 Bit</td>
<td>ein Unicode-Zeichen</td>
<td>System.Char</td>
</tr>
<tr>
<td>decimal</td>
<td>128 Bit</td>
<td>±1.0 × 1018 to ±7.9 × 1018</td>
<td>System.Decimal</td>
</tr>
<tr>
<td>double</td>
<td>64 Bit</td>
<td>±5.0 × 10304 to ±1.7 × 10308</td>
<td>System.Double</td>
</tr>
<tr>
<td>float</td>
<td>32 Bit</td>
<td>±1.5 × 1038 to ±3.4 × 1038</td>
<td>System.Single</td>
</tr>
<tr>
<td>int</td>
<td>32 Bit</td>
<td>-2,147,483,648 bis +2,147,483,647</td>
<td>System.Int32</td>
</tr>
<tr>
<td>uint</td>
<td>32 Bit</td>
<td>0 bis 4,294,967,295</td>
<td>System.UInt32</td>
</tr>
<tr>
<td>ulong</td>
<td>64 Bit</td>
<td>0 bis 18,446,744,073,709,551,615</td>
<td>System.UInt64</td>
</tr>
<tr>
<td>short</td>
<td>16 Bit</td>
<td>-32768 bis +32767</td>
<td>System.Int16</td>
</tr>
<tr>
<td>ushort</td>
<td>16 Bit</td>
<td>0 bis 65535</td>
<td>System.UInt16</td>
</tr>
</tbody>
</table>

*Tabelle 4.1: Die integralen Datentypen von .NET*

Im Gegensatz dazu sehen Sie hier die ganzzahligen Datentypen in C/C++:

File: Kurs MS10C# Bayerlein4.6.docx
Aus Kaiser C++ mit Borland C++ Builder

3.3 Ganzzahldatentypen

Varianten, deren Datentyp ein Ganzzahldatentyp ist, können ganzzahlige Werte darstellen. Je nach Datentyp können dies ausschließlich positive Werte oder positive und negative Werte sein. Der Bereich der darstellbaren Werte hängt dabei davor ab, wie viele Bytes der Compiler für eine Variable des Datentyps reserviert und wie er diese interpretiert.

In C++ gibt es die folgenden Ganzzahldatentypen:

<table>
<thead>
<tr>
<th>Datentyp</th>
<th>Wertebereich im C++Builder</th>
<th>Datenformat</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed char</td>
<td>-128 .. 127</td>
<td>8 bit mit Vorzeichen</td>
</tr>
<tr>
<td>char</td>
<td>(Voreinstellung)</td>
<td></td>
</tr>
<tr>
<td>unsigned char</td>
<td>0 .. 255</td>
<td>8 bit ohne Vorzeichen</td>
</tr>
<tr>
<td>short int</td>
<td>-32768 .. 32767</td>
<td>16 bit mit Vorzeichen</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Datentyp</th>
<th>Wertebereich im C++Builder</th>
<th>Datenformat</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signed short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signed short int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned short int</td>
<td>0 .. 65535</td>
<td>16 bit ohne Vorzeichen</td>
</tr>
<tr>
<td>wchar_t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>-2,147,483,648 .. 2,147,483,647</td>
<td>32 bit mit Vorzeichen</td>
</tr>
<tr>
<td>signed int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signed long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signed long int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned int</td>
<td>0 .. 4,294,967,295</td>
<td>32 bit ohne Vorzeichen</td>
</tr>
<tr>
<td>unsigned long int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long long</td>
<td>-92233720368 .. 92233720368</td>
<td>64 bit mit Vorzeichen</td>
</tr>
<tr>
<td>(siehe Abschnitt 3.3.8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned long long</td>
<td>0 .. 18446744073709551615</td>
<td>64 bit ohne Vorzeichen</td>
</tr>
<tr>
<td>bool</td>
<td>true, false</td>
<td></td>
</tr>
</tbody>
</table>

Die Funktionsaufrufe der DLL sind in der Regel mit den C++ Datentypen deklariert, da muss man dann aufpassen. Z.B. ist der char in C ist 8 bit, der char in C# ist 16 Bit.
Die Header Dateien, die meist mit den DLL geliefert werden, sind leider für C# nicht direkt nutzbar. Am sichersten ist die Erzeugung einer Lib-Datei aus der DLL, aus der man dann genau entnehmen kann, welche Funktionen wirklich in der DLL verborgen sind.

using System.Runtime.InteropServices;

 Dann außerhalb einer Funktion im Deklarationsteil:
[DllImport("ProjecttestSqr.DLL")]
public extern static double sqr(double x);
Diese Datei muss das Programm finden können, deshalb am Einfachsten kopieren nach Projekt...\bin\debug. Sollte nun der Fall auftreten, dass C# diese Funktion sqr nicht finden kann, obwohl Sie sicher sind, dass diese Funktion drin sein muss, dann hilft nur die Möglichkeit, diese DLL zu untersuchen. Ich habe in Foren den Hinweis auf ein Freeware Tool namens DependencyWalker gefunden, mit dem eine DLL analysiert werden kann. Man lädt die DLL und er listet alle möglichen exportierten Funktionen auf. Beispiel: Hab mit dem „alten C++ Builder eine Dll erzeugt mit folgendem Code:

\[
\text{double } \_\_\text{declspec(dllexport) } \text{sqr(double x)} \\
\{ \\
\quad \text{return } (x \times x); \\
\}
\]

Also eine einfache Quadratur der übergebenen Zahl x. Daraus hab ich die DLL namens ProjectTestSqr.dll erzeugt. In C# führte dann der normale Aufruf wie oben beschrieben zu der Fehlermeldung

Erst eine Suche mit DependencyWalker brachte des Rätsels Lösung: Screenshot:

Der Name der Funktion ist dort mit „@sqr$qd“ angegeben. Damit führte dann folgender Aufruf zum Erfolg:

\[
[DllImport("ProjectTestSqr.dll",EntryPoint="@sqr$qd")] \\
\text{public extern static double } \text{sqr(double x);} \\
\text{private void button1_Click(object sender, EventArgs e) \\
\{ \\
\quad \text{double } y = \text{Convert.ToIntDouble(textBox1.Text);} \\
\quad \text{textBox1.Text = sqr(y).ToString("F5");} \\
\}]
\]

So zeigte sich, dass die DLL für das USB- ADC- Interfaceboard von Prof. Dr. Orlowski (in FH erhältliche) namens „USBCardFH2.dll“ jetzt alle Namen mit einem Unterstrich davor wieder findet: Screenshot DependencyWalker dieser Dll:
Die Anbindungen der DLL der Vellemann – USB-Karte mit der DLL K8055D lieferte folgendes Bild, alle Namen tauchen wie in der Doku auf:

Natürlich ist hier nicht entnehmbar, wie die Parameterübergabe definiert ist, dass muss der Doku oder der Header – Datei der Dll entnommen werden.
**Zusammenfassung AD- DA mit dll für ME2600, USB Orlowski:**

**ME2600**

Vorher einbinden von **ME2600.dll** und using System.Runtime.InteropServices;

DA und AD haben 12 Bit Auflösung. Die übergebenen Werte bilden -10 bis +10 Volt ab in iValue 0 bis 4095.

**Initialisieren**

```csharp
[DllImport("ME2600.DLL")]
public extern static int me2600AOSetMode(int iBoardNumber, int iChannel, int iRange, int iMode);
```

**Beispiel**

```csharp
ex=me2600AOSetMode(0, 0, me2600Def.AO2600_MINUS_10,
me2600Def.AO2600_TRANSPARENT);
```

**Auf DA Schreiben**

```csharp
[DllImport("ME2600.DLL")]
public extern static int me2600AOSingle (int iBoardNumber, int iChannel,
short iValue);
```

**Beispiel**

```csharp
short outval = (short)numericUpDown1.Value;
int ex=me2600AOSingle(0, 0, outval);
```

**Von DA Lesen**

```csharp
[DllImport("ME2600.DLL")]
public extern static int me2600AISingle(int iBoardNumber, int iChannel, int
iMode, ref short iValue);
```

**Beispiel**

```csharp
short val=0;
int ex=me2600AISingle(0, (int)numericUpDownADChan.Value,
me2600Def.AI2600_MINUS_10 + me2600Def.AI2600_SINGLE_ENDED, ref val);
```

**USB- Orlowski – Karte – alte USB 1.1 - Version**

Vorher einbinden von der **USBCardFH2.dll** und using System.Runtime.InteropServices;

**Initialisieren**

```csharp
[DllImport("USBCardFH2.dll")]
public extern static bool _Init();
[DllImport("USBCardFH2.dll")]
public extern static bool _SetKanal2Bereich(byte a);
[DllImport("USBCardFH2.dll")]
public extern static bool _SetMode(byte a);
```

**Beispiel**

```csharp
bool OK = _Init();
_setKanal2Bereich(1);//bip1_4V=1, bipolar -4 bis +4Volt
_setMode(2);          //StartRealtime=2
```
**Auf DA Schreiben**

```csharp
[DllImport("USBCardFH2.dll")]
public extern static bool _SetAnalogOut1(ushort a);
```

**Beispiel**

```csharp
ushort Ua=3000;
/SetAnalogOut1(Ua); // Einstellen einer Spannung in 3V am D/A-Ausgang
```

**Von DA Lesen**

```csharp
[DllImport("USBCardFH2.dll")]
public extern static bool _RealtimeData(ref float x);
```

**Beispiel**

```csharp
float [] buf = new float [8];
double fak = 10.0 / 4096.0;
/RealtimeData(ref buf[0]); // Abholen der Messwerte von Kanal 2;
label2.Text = "Ua = " + (buf[1] * fak).ToString("F3") + " V";
```

---

**Neue DLL / Klassenbibliothek mit C# erzeugen**

Um eine eigenen neue DLL zu erzeugen, öffne man ein neues Projekt mit der Eigenschaft Klassenbibliothek.

Ich habe dort die gleiche DLL mit der Quadratfunktion erzeugt wie mit dem Borland CBuilder:

**Syntax:**

```csharp
namespace ClassLibrary1
{
    public class Class1
    {
        public static double sqrCSHarp(double x)
        {
            return (x * x);
        }
    }
}
```

Wichtig ist vor der Methodendefinition das Schlüsselwort `static`, erst dann wird die Funktion `sqrCSHarp` exportiert und benutzbar.

Nach „Build Project“ ist eine DLL verfügbar. Mit default – Namen sieht es jetzt so aus:
Will man diese DLL in einem anderen Projekt benutzen, so füge man sie als Verweis hinzu: (rechte Maus auf Verweise, dann Verweis hinzufügen, dann kommt linkes Bild.)

anschließend ist dann die Funktion `sqrCSarp` verfügbar und kann benutzt werden:

Beispiel:
```csharp
double y = Convert.ToDouble(textBox1.Text);
textBox1.Text = ClassLibrary1.Class1.sqrCSsharp(y).ToString("F5");
```

## Kurs 15 Zeitmessung und 1ms Timer

Die Zeitmessung kann sehr genau mit der Klasse `Stopwatch` durchgeführt werden. Mit

```csharp
Stopwatch stopwatch1 = new Stopwatch();
```


```csharp
long dtstart = stopwatch1.ElapsedTicks;
```
dann kommt der Befehl, von dem man die Laufzeit messen will...
```csharp
long dtend = stopwatch1.ElapsedTicks;
```
dann
```csharp
long dt = (dtend - dtstart);
```

Anzeige in Millisekunden:
```csharp
double dtsec = dt / (double)Stopwatch.Frequency * 1000;
Label1.Text = "Laufzeit= " + dtsec.ToString("F2");
```
Im Projekt „Test StandardTimer mit Zeitmessung Bay“ wird nun die Periodendauer des System-Timers gemessen. Dabei läuft stopwatch endlos vor sich hin und im TimerTick wird er lediglich periodisch abgefragt, die Abstände der Ticks berechnet und in einem Speicher TimeAvr aufakkumuliert, um den Durchschnittswert zu bestimmen. Mit einem zweiten Timer wird dann zweimal in der Sekunde das Ergebnis angezeigt.

Man stellt fest, dass der System - Timer keine 1ms – Auflösung besitzt. Die Periode kann nur in groben Schritten verstell werden. Bei meinem Desktop ist dies etwa 15 ms, die Intervallzeit kann nur ein Vielfaches dieser 15 ms betragen.

Ich frage mich, wie es möglich ist, dass die C#-Entwickler wirklich einfach nicht in der Lage sind, diesen Timer auf 1 ms genau hinzubekommen. Seit Jahrzehnten (schon bei Borland) bietet man einen Timer an, in dem das Intervall auf 1 ms verstell werden kann, tatsächlich ist er in Wirklichkeit nur 10 – 15 ms genau.

Ich bin froh, dass es meinem Dritthersteller Mario Prantl (mario.prantl@hh-system.com) genauso geht wie mir. Er hatte für Borland einen 1 ms Timer entwickelt, ist inzwischen auch auf C# umgestiegen und bietet auch dafür seinen 1 ms Multimediatimer an. Die FH hat eine Campuslizenz für 150 € gekauft. Industrienernutzer müssen sich eine eigene Lizenz besorgen.


```csharp
   instanziert man ihn mit
```

```csharp
   TMultimediaTimer MTimer1;
   und initialisiert ihn mit
   MTimer1 = new TMultimediaTimer(null);

Bei den Email – Objekten ging dies mit
```csharp
   pop3.OnWork += new TWorkEvent(this.Work);
```
mit dem das Ereignis Work dann verfügbar war.

Hier wird dies mit „Delegate“, einem neuen Konzept von C# realisiert. Das Konzept ist schwierig zu verstehen (an dieser Stelle nicht nötig) die Durchführung sieht furchtbar einfach aus. Mit

```csharp
   MTimer1.OnTimer = EventVomMMTimer;
```

wird die Ereignisfunktion EventVomMMTimer erzeugt und kann dann wie die beim Standard – Timer bekannte _Tick() – Ereignisfunktion benutzt werden:

```csharp
   void EventVomMMTimer(TMultimediaTimer sender)
   {
      .......
   }
```

Im Projekt erkennt man, dass jetzt der Multimediatimer eine Auflösung von 1 ms besitzt. Eine genaue Überprüfung dieser Periodendauer mit gemessener Zeit und einem eventuell vorhandenen Jitter (Schwankung der Periodendauer) ist möglich, wenn man mit diesem Timer über eine AD- Karte, die schnell genug ist, in jedem Event einen Impuls auf einem DA- Ausgang erzeugt und mit einem Digitalscope misst. Dies wird in dem Projekt MMTimer mit
**DA Impuls auf ME2600 realisiert.** Dieses Programm läuft natürlich nur mit einer ME2600 – Meilhauskarte im PC. Aber mit einer anderen Karte (keiner USB, USB ist nicht schnell genug) ginge es natürlich auch. Das Verhalten ist identisch dem Multimediatimer unter Borland CBuilder. Es gibt eine von der Last von Windows abhängigen Schwankung, und statt 1 ms sind es 1024 usec Periodendauer auf einem „BlueChip – PC.“

**Kurs 16 RS232**


Will man 2 PCs miteinander verbinden, dann benötigt man ein „Nullmodemkabel“, bei dem PIN 2 und 3 über Kreuz verbunden sind:

```
PIN 5       SG
PIN 3       TD
PIN 2       RD
```

Damit die Geräte sich aber synchronisieren können, müssen die „Sendefrequenzen“ gleich sein. Diese werden mit der Eigenschaft BaudRate eingestellt, Standardwert sind 9600 Baud, das sind 9600 Bits pro Sekunde. Ebenso müssen Parity (None,Odd, Even, Mark, Space) und Anzahl der StopBits (One, Two, OnePointFive) und die DataBits (7 oder 8) übereinstimmen.


Hat ein PC keinen COM-Port (das ist diese seriele RS2322) mehr, dann kann man sich einen USB-Seriell – Adapter kaufen (z.B. DIGITUS USB 2.0 SERIEL ADAPTER, bei Conrad 10 €).

Bevor man startet, kann man mit

```
foreach (string s in SerialPort.GetPortNames())
    cbPort.Items.Add(s);
```

z.B. in eine ComboBox cbPort alle auf dem PC verfügbaren COM-Ports ermitteln. Die tatsächlich angeschlossene muss dann in der Eigenschaft „PortName“ eingestellt werden, also z.B.

```
serialPort1.PortName = “COM1”
```

Dann kann dieser Port mit .Open() geöffnet werden.
Steht ein Text in einem string tx, so kann man nun einfach Daten senden mit

```csharp
serialPort1.Write(tx);
```

Benutzt man WriteLine, so wird dem Text ein LineFeed (Hex0a) anhängt, allerdings kein CarriageReturn (Hex0d). In einer Textbox wird dann kein Zeilenvorschub gemacht. Also am Besten nur Write() benutzen und ggf. einen Zeilenvorschub mit „\x0d“ und „\x0a“ anhängen.

Empfangene Daten lesen kann man mit .ReadByte() oder .ReadChar(). Damit keine Daten verloren gehen, wenn der Eingangsbuffer voll ist, sollte man dieses in einem Timer periodisch z.B. alle 100 ms durchführen. Erwartet man nur Text, so benutzt man ReadChar(), bei binären Daten (z.B. aus unserem Scope) muss man mit ReadByte() lesen. Also in einem Timer_Tick – Event (inp ist eine globale string- Variable):

```csharp
if (serialPort1.BytesToRead > 0)
{
    for (int i = 0; i < serialPort1.BytesToRead; i++)
    {
        char ch=(char) serialPort1.ReadByte();
        inp=inp+ch;
    }
    textBox1.AppendText(inp);
    inp = "";
}
```

Voreingestellt ist eine ASCII – Codierung, das heißt, deutsche Sonderzeichen gehen verloren, im Empfangenen Text steht dort ein “?”? Will an diese Sonderzeichen auch lesen können, dann muss z.B. nach Initialisierung der Befehl

```csharp
serialPort1.Encoding = Encoding.Default;
```

ausgeführt werden. In der bool - Eigenschaft .isOpen kann man abfragen, ob ein Öffnen geklappt hat, denn andere Prozesse können ja auch einen COM- Port belegen. Will man Eigenschaften wie BaudRate verändern, muss zuvor der Prot geschlossen werden mit .Close().

Noch ein Hinweis zur Textbox. Schreibt man die empfangenen Zeichen in die Textbox mit AppendText(), so wird dort nur eine neue Zeile gezeichnet, wenn die Sonderzeichen „\x0d“ und „\x0a“ hintereinander auftauchen oder nur LineFeed („\x0a“) auftaucht. Löst man dann aber den Text mit .Lines in ein String- array auf, dann wird in diesem Stringarray auch schon bei einem CarriageReturn - Zeichen („\x0d“ ) eine neue Zeile erzeugt. Will man also auf einzelne Zeilen zugreifen, kann man sich verzählen. Beispiel:

```csharp
textBox1.Text = "Hello World!"
```

Anschließend nach diesem Header werden die Werte der Abtastpunkte in Binärform übertragen. Das erste Byte ist ein vorzeichenbehaftetes High-Byte (Type sbyte), das zweite das vorzeichenlose low-Byte (type byte). Der Spannungswert des Punktes ergibt sich zu
\[ Y_{\text{zero}} + Y_{\text{resolution}} \times (\text{High} \times 256 + \text{low}) \] und der Zeitwert errechnet sich zu \[ t = X_{\text{zero}} + X_{\text{resolution}} \times \text{num} \], wobei num die Nummer des Abtastpunktes ist. Das Auslesen der seriellen Schnittstelle muss hier mit ReadByte() durchgeführt werden.
Manche Multimeter benötigen als Sparversion für eine Pegelwandlung auf dem DTR-Signal (PIN 4) eine positive Spannung, sonst geht gar nichts. Dies kann man erreichen, wenn man

```csharp
serialPort1.DtrEnable = true;
```

)einmal ausführt

Auszug aus Scope – Bedienanleitung siehe nächste Seite.
**weve_nr**  Die Quelle der Oszilloskopwellenform:
001 - 004 für CH1 - CH4
011 - 084 für m1.1 - m8.4
091 - 504 für m9.1 - m50.4 (nur bei erweitertem Speicher)

**admin** 16 Parameter, getrennt mit einem ",":

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TYP</th>
<th>BEISPIEL</th>
<th>ANMERKUNGEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace_name</td>
<td>Zeichenfolge</td>
<td>m4.1</td>
<td></td>
</tr>
<tr>
<td>Y_unit</td>
<td>Zeichenfolge</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>X_unit</td>
<td>Zeichenfolge</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>Y_zero</td>
<td>Zahl</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>X_zero</td>
<td>Zahl</td>
<td>-8.625E-6</td>
<td></td>
</tr>
<tr>
<td>Y_resolution</td>
<td>Zahl</td>
<td>78.13E 3</td>
<td></td>
</tr>
<tr>
<td>X_resolution</td>
<td>Zahl</td>
<td>1E-6</td>
<td></td>
</tr>
<tr>
<td>Y_range</td>
<td>Zahl</td>
<td>65535</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>Zeichenfolge</td>
<td>0000-00-00</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Zeichenfolge</td>
<td>00:00:00</td>
<td></td>
</tr>
</tbody>
</table>
| dT-corr         | Zahl         | 375E-3       | dT = dT-corr * X-
|                 |              |              | Auflösung                |
| min/max         | Zahl         | 0            | 1 = min/max-Spur          |
| reserved        | Zeichenfolge | 0            |                           |
| reserved        | Zeichenfolge | 0            |                           |
| reserved        | Zeichenfolge | 0            |                           |
| reserved        | Zeichenfolge | 0            |                           |

**count**  Acquisitionslänge:
512, 6192,16384 oder 32768
(Erweiterter Speicher: 2048 und mehr)

**sample**  2 Bytes (Höchstwertiges Byte zuerst), die den 16-Bit Probenwert
darstellen (Bit 16 = -32768 ... Bit 1 = 1)
Bereich: -32k (down) ... +32k (up)

**checksum**  1 Byte Prüfsumme über sämtliche Proben-Bytes

---

**Kurs 17 bis 20 entfernt**

**Kurs 21 DataGridView**

In diesem Kurs soll die recht komplexe Komponente DataGridView eingeführt werden.
Dazu starten wir ein neues Projekt im Ordner Kurs 20# und nennen das Projekt TestGridView.
Dann eine Komponente DataGridView auf das Formular ziehen mit Dock Top.
Wir öffnen den Spalteneditor im Eigenschaftenfenster, dort unter „Columns“ die Auflistung
öffnen:
Dann fünf Spalten hinzufügen:

Das Formular sieht dann so aus:

Es soll diese Komponente mit fest programmiert
Information gefüllt werden. Dazu den nächsten
Programmabschnitt in ihr Programm außerhalb des
Konstruktors einfügen:

```csharp
string[] row0 = { "11/22/1968", "29", "Revolution 9", "Beatles", "The Beatles [white Album]" };
string[] row2 = { "1969", "6", "Fools Rush In", "Frank Sinatra", "Nice 'N' Easy" };
string[] row3 = { "11/11/1971", "1", "One of These Days", "Pink Floyd", "Meddle" };
string[] row6 = { "6/18/2003", "13", "Scatterbrain. (As Dead As Leaves.)", "Radiohead", "Hail to Thief" };

Damit werden 7 string- Arrays vorbelegt mit je 5 Elementen.
Hinter einer Taste mit Namen buttonLoad dann folgenden Text kopieren:
```
dataGridView1.Rows.Add(row0);
dataGridView1.Rows.Add(row1);
dataGridView1.Rows.Add(row2);
dataGridView1.Rows.Add(row3);
dataGridView1.Rows.Add(row4);
dataGridView1.Rows.Add(row5);
dataGridView1.Rows.Add(row6);
dataGridView1.Columns[1].HeaderText = "Name1";
```

Nach Start und Klick auf diesen button sieht dann die Form so aus:

Man beachte, wie die Spaltenüberschrift der zweiten Spalte mit dem Befehl
dataGridView1.Columns[1].HeaderText = "Name1";
verändert werden kann, also Zählung von Null.
Fügt man folgende Zeile hinzu:

```csharp
dataGridView1.Rows[3].HeaderCell.Value = "Zeile4";
```

Dann wird so auch eine Zeilenbeschriftung ganz rechts möglich. Man kann die Größe automatisch einstellen lassen mit z.B.

Oder man gibt in der Eigenschaft „RowHeaderWidth“ die Größe in Pixel an. Das geht aber nur so mit „EnableResizing:

Jetzt sollen die Daten aus der Komponente ausgelesen werden. Wie kommt man an die Inhalte dran: dazu bitte eine label3 und eine textBox1 hinzufügen und einen Eventhandler hinter einem CellClick erzeugen. Dann folgenden Code dort eingeben:

Außerhalb der function:

```csharp
int c=0, r=0;
```

Innerhalb:

```csharp
c = e.ColumnIndex;
r = e.RowIndex;
label3.Text="clicked Row= "+r.ToString() +" Col= "+c.ToString();
if ((c==0)&&(r==0)&&(c<dataGridView1.ColumnCount)&&(r<dataGridView1.RowCount))
textBox1.Text = dataGridView1[c,r].Value.ToString();
```

In dem Parameter e kann man den Spalten- und Zeilenindex abfragen, diese werden nach c und r geschrieben und in label3 ausgegeben. Das Auslesen geschieht wie bei einem zweidimensionalen Array mit ...[c,r] per Eigenschaft Value.

Ein Beispiel erkennt man im nächsten Screenshot, auch da hat die Zelle oben links die Zeile Null und die Spalte Null:

Man kann jetzt z. B. in jeder Zelle einen eigenen Font einstellen. Beispielprogramm dazu hinter einer Taste buttonFont, vorher einen FontDialog auf die Form ziehen:

```csharp
if (fontDialog1.ShowDialog() == DialogResult.OK)
dataGridView1[c, r].Style.Font = fontDialog1.Font;
```

Z.B. Zelle 2,2 mit Comic Sans MS, Schriftgröße 12:

File: Kurs MS10C# Bayerlein4.6.docx
Man muss darauf achten, dass keine Zelle angesprochen wird, die es nicht gibt, denn sonst bricht das Programm mit Fehlermeldung ab. Da bietet sich eine try – catch – Abfrage an.

Soll der User Zeilen hinzufügen können, dann muss folgende Eigenschaft auf true gesetzt werden:

dataGridView1.AllowUserToAddRows = true;

dann gibt es eine Leerzeile am Ende der Tabelle. Wenn dort etwas hineingeschrieben wird, sieht es so aus:

![Image](image.png)

Nur dann gibt es bei einem Klick in diese neue Zeile sofort eine Fehlermeldung, da der Inhalt sofort in die Textbox geschrieben werden soll. Dieser existiert aber noch nicht. Die Eigenschaft RowCount ist schon um eins erhöht worden, aber für den Inhalt ist noch kein Speicherbereich zugewiesen worden. Das kann man abfangen mit folgendem if:

```csharp
if (dataGridView1[c, r].Value!=null) textBox1.Text = dataGridView1[c, r].Value.ToString();
```

Erst wenn die Zeile verlassen wird, wird der Inhalt aktualisiert und der Stern verschwindet.

**Kurs 22 C# DirectX**


**Kurs 23 MySql und C#**

Auf der Seite http://www.fh-luebeck.de/Inhalt/03_Hochschulangehoerige_Ch031/04_FB_Elektrotechnik/04_Labore_und_Institute/63_Labor_ESA_RT/Visual-C_/MYSQL/index.html findet man alles, was man wissen muss, um mit C# eine mySQL- Datenbank anzusprechen. Dies wurde in einer Diplomarbeit von Frau Yannan Shen erarbeitet, die dort auch zur Verfügung gestellt wird. Ein Beispielprogramm vervollständigt das Angebot. Dort ist ein

**Kurs 24 Meilhaus ME 4660**

Inzwischen sind auch modernere Versionen der Meilhauskarte (ME 4660) und der Orlowskkarte (USB 2.0 – Version) entstanden und im Labor verfügbar. Ich habe die Anleitungen dazu im Internet veröffentlicht.

ME4660: (geht sowohl mit 32.Bit als auch mit 64 Bit- Windows!)


**Kurs 25 NI mit NIDAQmx**


![Image of .NET tool window](image.png)

Anschließend muss dann im Projektmappen-Explorer dann folgendes stehen:

File: Kurs MS10C# Bayerlein4.6.docx
Dann

```csharp
using NationalInstruments.DAQmx;
```

Und dann sind folgende Befehle möglich:

AD auslesen:

```csharp
double val = rschan.ReadSingleSample();
```

Ausgabe auf DA

```csharp
wschan.WriteSingleSample(true, val);
```

zuvor muss man die beiden Tasks rschan und wschan „kreieren“ mit

```csharp
AnalogSingleChannelReader rschan;
AnalogSingleChannelWriter wschan;
```

Und das geht so: Erst die Instanziierung der AD- Tasks: (DA geht ähnlich, siehe Datei .cs):

```csharp
Task taskAI = new Task("abc1");
dann

taskAI.AIChannels.CreateVoltageChannel(actdev+"/ai0", "aiChannel0", 
  AITerminalConfiguration.Rse, MinI, MaxI, AIVoltageUnits.Volts);
```

Dann

```csharp
rschan = new AnalogSingleChannelReader(taskAI.Stream);
```

dann sollte es nach taskAI.Start(); gehen.

Weitere Hinweise kann man auf den Internetseiten von mir zu diesem Punkt finden.

---

**Kurs 26 USB- Orlowski 2.0**

USB – Orlowski USB 2.0 Version: (geht nur auf 32. Bit Windows)


**Kurs 27 Interface mit TCP/IP**


Man benötigt Zugriff auf diesen Namespace:

```csharp
using System.Net.Sockets;
```

dann kann man einen TCP – Client instanziieren:

```csharp
TcpClient tcpClientWAGO = new TcpClient();
```

Hinter einem Button „Connect“ kann man dann mit folgendem Programm eine Verbindung aufbauen:
try
{
    if (!tcpClientWAGO.Connected)
    {
        tcpClientWAGO.Connect("192.168.0.2", 502); // tcpClient.Connect("193.175.124.132", 502);
        if (tcpClientWAGO.Connected) toolStripStatusLabel1.Text = "Wago is connected";
    }
    else MessageBox.Show("Verbindung besteht bereits", "TCP Connection Info", MessageBoxButtons.OK);
}
catch (SocketException)
{
    MessageBox.Show("Verb.-aufbau nicht möglich", "TCP Connection Error", MessageBoxButtons.OK);
}

Dabei wird eine Nachricht in ein toolStripStatusLabel geschrieben, ein StatusStrip muss
natürlich dann vorhanden sein.
Wenn eine Verbindung gelungen ist, kann man Lesen und Schreiben mit einem
NetworkStream:
NetworkStream netStreamWAGO;

Den muss man mit der Verbindung zusammenführen. Das Schreiben geht so:

netStreamWAGO = tcpClientWAGO.GetStream();
if (netStreamWAGO.CanWrite)
{
    byte[] sendBytes = new byte[12]; // lokales Array mit Daten
    netStreamWAGO.Write(sendBytes, 0, sendBytes.Length);
}

Das Lesen geht so:

netStreamWAGO = tcpClientWAGO.GetStream();
if (netStreamWAGO.CanRead)
{
    byte[] readBytes = new byte[tcpClientWAGO.ReceiveBufferSize];
    netStreamWAGO.Read(readBytes, 0, (int)tcpClientWAGO.ReceiveBufferSize);
}

Anschließend sind im Byte-Array die Daten.

Das Wago-Protokoll ist den Versuchsunterlagen zu entnehmen und wird am Versuchstag zur Verfügung gestellt.

WindfC#, mein großes Regelungstechnik-Toolprogramm.
Weitere Informationen kann man erhalten, wenn man sich das folgende Projekt, das auch als Source-Code in C# erhältlich sind - quasi als Freeware – anschaut:
http://www.fh-luebeck.de/Inhalt/03_Hochschulangehoerige_Ch031/04_FB_Elektrotechnik/04_Labore_und_Institute/63_Labor_ESA_RT/WindfCSharp/index.html
Einige Hinweise

Problematis mit Dock

Werden zwei Elemente mit Dock auf Top gesetzt, erhebt sich die Frage, welcher wirklich zuerst oben und welche dann darunter platziert wird. Beispiel ToolStrip und Panel:

So soll es sein. Manchmal aber passiert es, dass bei beiden Elementen, die auf Dockstyle.Top gesetzt werden, eine falsche Platzierung erfolgt, z.B. so:

Es hängt von der Reihenfolge ab, mit der diese Elemente der Form bzw. dem Container hinzugefügt worden sind. Wie kann man diese nachträglich verändern? So: Man muss die Datei „designer“, die normalerweise nicht angefasst wird, nun doch öffnen. In dem Teil, der oben rechts ausgeschnitten ist, kann man die Zuordnung erkennen. Das Element, das unter dem anderen steht, gewinnt!

Parameter in Funktionsaufrufen und Arrays

Dort gibt es einige Änderungen gegenüber C++:

Beispiele:

Deklaration von Arrays:

<table>
<thead>
<tr>
<th>C++</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>double xein[max2n]</td>
<td>double[] xein = new double[max2n]</td>
</tr>
</tbody>
</table>

Funktionsaufruf

<table>
<thead>
<tr>
<th>C++</th>
<th>C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>void xr_db(double&amp; xrr, double p[10], double&amp; q[10], int m)</td>
<td>void xr_db(ref double xrr, double[] p, ref double[] q, int m)</td>
</tr>
</tbody>
</table>
Xrr wird referenziert aufgerufen, damit können Werte zurückgegeben werden, es wird nur der Pointer übergeben. Das Array p wird in die Funktion hineingeben und von der Funktion nicht verändert. Das Array q wird wieder referenziert, also lediglich mit Pointer übergeben und kann von der Funktion verändert werden.

**Zeilennummern im Editor**

Manchmal helfen Zeilennummern im Texteditor. Sie aktiviert man mit Menü Debuggen → Optionen…
Dann im Text-Editor → Alle Sprachen
Dort Klick auf Zeilennummern:

Weitere Projekte werde ich auf den Internetseiten unter

www.fh-Luebeck.de, → Studierende → FB EI → Labore & Institute, → Labor Regelungstechnik → Visual-C# -Page oder Windfc#
dennächst in loser Folge veröffentlichen.

Stand November 2012
Prof. Dr. Bayerlein