MyNotepad. Ein kleiner Notepad-Klon mit VB.NET und Windows.Forms

== Getting started ==
Beim ersten Start sieht das Studio ungefähr so aus, wir wollen erst einmal ein neues Projekt anlegen. Dazu kann man links auf neues Projekt klicken, oder Datei->Neues Projekt…

[Blockierte Grafik: http://hertz-eichenrode.net/easy-coding/Bild1.png]
Screenshot 1: Visual Studio Startbildschirm

Wir wählen links Visual Basic und in der Mitte „Windows-Forms-Anwendung“ aus. Unten kann man noch einen Projektnamen, wie beispielsweise MyNotepad eingeben.

[Blockierte Grafik: http://hertz-eichenrode.net/easy-coding/Bild2.png]
Screenshot 2: Neues Projekt anlegen

Nach dem Bestätigen mit OK erhalten wir ein leeres Windows Fenster, das wir gestalten können.

== Form-Design ==

[Blockierte Grafik: http://hertz-eichenrode.net/easy-coding/Bild4.png]
Screenshot 3: Aufbau Visual Studio

1. Toolbox, hier sind die Windows-Controls zu finden
2. Hauptfenster, hier ist entweder der Code oder der Form-Designer zu sehen (mehrere Tabs, die mit Drag&Drop „herausnehmbar“ sind)
3. Projektmappen-Explorer, hier sind alle zu dem Projekt gehörende Objekte abgelegt (wird in diesem Tutorial nicht weiter benötigt)
4. Eigenschaften-Fenster: Hier sind die Eigenschaften und Ereignisse der Controls zu finden

[Blockierte Grafik: http://hertz-eichenrode.net/easy-coding/Bild3.png]
Screenshot 4: Toolbox

Auf der linken Seite von Visual Studio befindet sich die Toolbox, in der alle Windows Controls zu finden sind, die das Visual Studio einem zur Verfügung stellt. Je nach Bildschirmbreite lohnt es sich diese Toolbox zu anzupinnen. Wir benötigen folgende Controls um den Notepadklon zu realisieren (hinzufügen per Doppelklick oder Drag&Drop):

- MenuStrip
- TextBox
- OpenFileDialog
- SaveFileDialog
- StatusStrip

Ich ziehe mir alle Controls von denen ich am Anfang weiß, dass ich sie benötigen werde schon gleich auf die Form. Hier handelt es sich, mit Ausnahme der TextBox um nicht frei platzierbare Controls die unterhalb der Form gesammelt werden. Das erste, was unbedingt nach dem Hinzufügen eines neuen Controls zu tun ist, ist das Ändern des Namens. Bei größeren Anwendungen verliert man sonst schnell den Überblick. Der Name des Control ist eine Eigenschaft. Die Eigenschaften eines Controls kann man auf der rechten Seite von Visual Studio in dem Eigenschaften-Fenster sehen, wenn man das Control selektiert.

[Blockierte Grafik: http://hertz-eichenrode.net/easy-coding/Bild5.png]

Screenshot 5: Eigenschaften-Fenster

Hier ist es sinnvoll sich ein Namenskonzept zu überlegen, ich schreibe in kleinen Buchstaben die den Namen des Controltyps vorweg (verkürzt auf max. 4 Buchstaben), gefolgt von seinem Einsatzzweck in CamelCase-Notation. Die Vorteile davon werdet ihr beim Programmieren sehen, da AutoComplete so effektiver nutzbar ist. Hier nochmal die Controls und ihre Namen:

- MenuStrip: msMainMenu
- TextBox: tbText
- OpenFileDialog: ofdOpen
- SaveFileDialog: sfdSave
- StatusStrip: ssStatus

Damit das ganze etwas mehr nach einem Notepad aussieht, müssen wir erstmal ein paar weitere Eigenschaften der TextBox ändern. Unter der Rubrik gibt es eine Eigenschaft, die heißt Dock. Wenn man diese ausklappt, bekommt man ein Fenster, in dem man angeben kann, wo das Control andocken soll. Wir wählen das mittlere (Fill), um das ganze Formular auszufüllen. Weil der Standard der Textbox anders einzeilig ist, müssen wir die Eigenschaft Multiline unter der Rubrik Verhalten auf true ändern sowie WordWrap in derselben Rubrik auf False. Zuletzt müssen wir ScrollBars unter der Rubrik Darstellung auf Both. Das bewirkt, dass wir ein mehrzeiliges Edit haben, dass Scrollbar ist und nicht von alleine Wörter umbricht, wenn die Zeile voll ist.

[Blockierte Grafik: http://hertz-eichenrode.net/easy-coding/Bild6.png]
Screenshot 6: Dock-Eigenschaften der TextBox auf Fill ändern

Ein letzter Schritt ist noch zu erledigen, bevor wir mit dem Programmieren beginnen können, und zwar das Erstellen des Menüs. Dazu klicken wir einmal auf ssMainMenu in der Hauptansicht unter dem Formular. Dann sehen wir oben im Formular ein Menü in dem man Text eingeben kann. Das ist eigentlich recht einfach und selbsterklärend. Man kann nach rechts und nach unten immer neue Einträge erzeugen. Wir erzeugen in diesem Tutorial folgende Einträge:

Datei: Neu, Öffnen, Speichern, Speichern unter.., - Beenden
Bearbeiten: Rückgängig, - Ausschneiden, Kopieren, Einfügen
?: Info

Der einfache Bindestrich (oder auch Minus) erwirkt einen sogenannten Seperator, einen logischen Trennstrich der nicht anklickbar ist.

== Programmierung ==

So, um etwas programmieren zu können, brauchen wir einen Code-Editor. Den erhalten wir sobald wir irgendein Control auf der Form doppelt anklicken. Visual Studio erstellt automatisch ein Gerüst indem wir tippen können. Bevor wir irgednwo etwas eintippen, sollten wir aber noch kurz auf das eventbasierte Programmieren eingehen. Beim programmieren mit Windows-Forms, ist es so, dass alle Systemnachrichten für uns ausgewertet werden und sogenannte Events oder Ereignisse generieren. So hat jedes Control unterschiedliche Ereignisse. Zum Beispiel kann ein Knopf (Button) ein Klick-Event haben. Immer wenn der Benutzer den Knopf drückt, wird das Ereignis ausgelöst und unser Code, den wir in das Klick-Event hineinschreiben ausgeführt. Eine TextBox kann zum Beispiel ein TextChanged-Event haben, dass immer ausgelöst wird, wenn der Text sich ändert (durch den Benutzer). Wenn man doppelt auf ein Control im Visual Studio Designer klickt, wird immer das Event vom Visual Studio generiert, das für das Control am häufigsten verwendet wird. Eine Liste aller Events findet sich in dem Eigenschaften-Fenster, wenn man auf den Blitz drückt. Dann stehen da wo vorher die Eigenschaften standen, jetzt die Events. Klickt man links neben den Blitz, wechselt man wieder zu den Eigenschaften zurück. Rechts neben einem Event steht entweder nichts, oder der Name einer Prozedur. Wenn da nichts steht und man doppelt hineinklickt, landet man in im Code-Editor, in dem eine leere Prozedur (Subroutine) angelegt wurde. Eine Prozedur ist ein Code-Abschnitt, der zwar etwas macht, aber kein Ergebnis zurückliefert. Eine Funktion ist ein Code-Abschnitt die ein Ergebnis zurückliefern muss. In VB sieht das generell so aus:

Quellcode

```
1. Private Sub Name(ByVal parameter1 Object, ByVal parameter2 As Integer)
2. End Sub
3. Private Function Name(ByVal parameter1 Object, ByVal parameter2 As Integer) As Integer
4. End Function
```

Private ist der Sichtbarkeitsmodifier. Er bestimmt von wo die Prozedur aus sicherbar ist, braucht uns hier nicht weiter zu interessieren, möglich sind Private, Protected und Public. Danach wird angegeben ob es sich um eine Prozedur oder eine Funktion handelt: Sub bedeutet Subroutine, also Prozedur, oder eben Function.
Ansachließend folgt der Name und die Liste von Parametern in VB-Notation. Bei einer Funtkion wird zum Schluss noch der Typ des Rückgabewertes angegeben.

Alle Funktionen und Prozeduren befinden sich in einem Klassengerüst:

Quellcode

```
1. Public Class Form1
2. End Class
```

Wir fangen mit dem einfachsten Knopf an. Das dürfte Datei->beenden sein. Wenn wir also in unserer Design-Ansicht auf Datei und dann doppelt auf Beenden drücken, wird ein neuer Regsiter eröffnet in dem wir den Code der Anwendung sehen. Wir sehen dann folgendes:

Quellcode

```
1. Public Class Form1
2. Private Sub BeendenToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
   Handles BeendenToolStripMenuItem.Click
3. End Sub
4. End Class
```

Wir schreiben dort wo der Cursor steht einfach Me.Close hin. Me ist die Selbstreferenz (auf Form1), und Close die Funktion die wir von Form1 aufrufen möchten. Jetzt speichern wir alles noch irgendwohin und machen den ersten Testlauf. Dazu drücken wir auf den grünen Pfeil im Visual Studio oben in der Menüleiste (oder F5). Dann startet unser Programm, welches wir jetzt über Datei->Beenden wieder beenden können.
Jetzt fügen wir zwei Klassenvariablen ein, die wir später brauchen werden. Dazu schreiben wir direkt unter

Quellcode

```
1. Public Class Form1
2. Dim bChanged As Boolean = False
3. Dim strFilename As String = ""
4. ...
```

In der Variable bChanged merken wir uns ob der Benutzer Änderungen vorgenommen hat, am Anfang initialisieren wir sie mit False.
In der Stringvariable strFilename speichern wir den Pfad zur aktuell geöffneten Datei, sofern eine Datei geöffnet bzw. die aktuelle bereits gespeichert wurde.

Wir brauchen noch eine Prozedur um den aktuellen Text zu speichern, sowie eine Funktion zum Öffnen einer bestehenden Datei.

Öffnen:

Quellcode

```
1. Protected Function loadFile(ByVal filename As String) As String
2. Dim sr As System.IO.StreamReader = New System.IO.StreamReader(filename)
3. Dim result As String = ""
4. Try
5. result = sr.ReadToEnd
6. sr.Close()
7. Return result
8. Catch ex As Exception
9. MsgBox("Fehler! Kann angegebene Datei nicht lesen!" & Chr(13) & Chr(10) & ex.ToString, MsgBoxStyle.Critical, "Fehler")
10. End Try
11. Return result
12. End Function
```

Alles anzeigen

Diese Funktion hat einen Parameter: den Pfad zu der Datei. Ihr Rückgabewert ist ein String (also Text), nämlich der den wir ausgelesen haben. Wir initialisieren uns ein StreamReader-Objekt, welches zum Auslesen von Dateien gedacht ist. Dafür übergeben wir den Dateinamen im Konstruktor (New System.IO.StreamReader(filename)), danach speichern wir den Rückgabewert von ReadToEnd in der Variablen result. Drum herum haben wir einen Try-Catch-Block, der uns vor Fehlern schützt. Sollten wir die Datei nicht lesen können (z.B. fehlende Berechtigungen oder bereits anderweitig geöffnet), erzeugen wir eine Fehlermeldung und setzten strFilename zurück, da wir die Datei ja nicht öffnen konnten.

Speichern:

Quellcode

```
1. Protected Function saveToFile(ByVal text As String, ByVal filename As String) As Boolean
2. Dim sw As System.IO.StreamWriter = New System.IO.StreamWriter(filename, False)
3. Try
4. sw.Write(text)
5. sw.Flush()
6. sw.Close()
7. Return True
8. Catch ex As Exception
9. MsgBox("Fehler! Kann angegebene Datei nicht schreiben!" & Chr(13) & Chr(10) & ex.ToString, MsgBoxStyle.Critical, "Fehler")
10. Return False
11. End Try
12. End Function
```

Alles anzeigen

Analog zum öffnen.

Ich habe bereits erklärt, wie man Events zu Controls erstellt und werde im Folgenden nur noch sagen, bei welchem Control wir in welchem Event was genau machen werden.

Speichern unter: Click

Quellcode

```
1. Private Sub SpeichernUnterToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SpeichernUnterToolStripMenuItem.Click
2. If sfdSave.ShowDialog() = DialogResult.OK Then
3. strFilename = sfdSave.FileName
4. End If
5. saveToFile(tbText.Text, strFilename)
6. bChanged = False
```

7. End Sub

Wir öffnen den SaveFileDialog sfdSave und wenn der vom Anwender erfolgreich benutzt wurde speichern wir den ausgewählten Dateinamen in der Variablen strFilename. Anschließend führen wir die vorhin gezeigte Funktion saveToFile aus. Als Parameter geben wir tb.Text, den Text in unseren TextBox-Control mit, und den Pfad zur Datei in der gespeichert werden soll. Anschließend merken wir uns das in dem Dokument keine ungespeicherten Änderungen sind.

Speichern: Click

Quellcode

1. Private Sub SpeichernToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles SpeichernToolStripMenuItem.Click
2. If strFilename.Length = 0 Then
3. SpeichernUnterToolStripMenuItem_Click(Nothing, Nothing)
4. Else
5. saveToFile(tbText.Text, strFilename)
6. End If
7. bChanged = False
8. End Sub

Wir überprüfen ob wir bereits eine Datei haben. Wenn nein (dann steht in strFilename nichts drin) und wir führen einfach den Speichern unter… Dialog aus. Wir können hier die Parameter Nothing benutzen da wir da nicht drauf zugreifen. Haben wir bereits eine Datei (weil schonmal gespeichert oder es eine geöffnete Datei ist), speichern wir mit unserer saveToFile-Routine. Auch hier merken wir uns anschließend, dass es keine ungespeicherten Änderungen gibt.

Neu: Klick

Quellcode

1. Private Sub NeuToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles NeuToolStripMenuItem.Click
2. If bChanged AndAlso MsgBox("Wirklich? Alle ungespeicherten Änderungen gehen verloren", MsgBoxStyle.YesNo, "Warnung") = MsgBoxResult.Yes Then
3. tbText.Text = ""
4. End If
5. bChanged = False
6. strFilename = ""
7. End Sub

Hier machen wir folgendes: Wenn es ungespeicherte Änderungen gibt, fragen wir den Anwender ob er das wirklich möchte. Bestätigt er dies mit Ja, setzen wir den Text der TextBox auf „“. Anschließend merken wir uns, dass es keine Änderungen mehr gibt (das Dokument ist ja jetzt wieder leer). Zusätzlich leeren wir den Dateinamen, da ja eine neue Datei entstehen soll.

Form1: Closing

Quellcode

1. Private Sub Form1_FormClosing(ByVal sender As System.Object, ByVal e As System.Windows.Forms.FormClosingEventArgs) Handles MyBase.FormClosing
2. If bChanged AndAlso MsgBox("Wirklich? Alle ungespeicherten Änderungen gehen verloren", MsgBoxStyle.YesNo, "Warnung") = MsgBoxResult.No Then
3. e.Cancel = True
4. End If
5. End Sub

Dieses Ereignis wird ausgeführt, bevor das Fenster geschlossen (und damit unsere Anwendung beendet wird). Das kann durch Datei->Beenden oder durch den Klick auf das rote X oben rechts . Wir prüfen wieder, ob es ungespeicherte

Änderungen gibt. Wenn ja, fragen wir ob er wirklich beenden möchte. Verneint er dies, canceln wir das gesamte Event mit e.cancel=true.

Rückgängig: Click

Quellcode

```
1. Private Sub RückgängigToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RückgängigToolStripMenuItem.Click
2. tbText.Undo()
3. End Sub
```

Ausschneiden: Click

Quellcode

```
1. Private Sub AusschneidenToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AusschneidenToolStripMenuItem.Click
2. tbText.Cut()
3. End Sub
```

Kopieren: Click

Quellcode

```
1. Private Sub KopierenToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles KopierenToolStripMenuItem.Click
2. tbText.Copy()
3. End Sub
```

Einfügen: Click

Quellcode

```
1. Private Sub EinfügenToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles EinfügenToolStripMenuItem.Click
2. tbText.Paste()
3. End Sub
```

Alles markieren: Click

Quellcode

```
1. Private Sub AllesMarkierenToolStripMenuItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AllesMarkierenToolStripMenuItem.Click
2. tbText.SelectAll()
3. End Sub
```

Diese ganzen Aktionen unter Bearbeiten haben wir so an die TextBox delegieren und bedürfen keiner Erklärung, denke ich.

tbText: TextChanged

Quellcode

```
1. Private Sub tbText_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
```

tbText.TextChanged
2. bChanged = True
3. End Sub

Sobald der Text sich ändert, müssen wir uns merken, dass der Benutzer Änderungen vorgenommen hat.

TextBox: KeyDown:

Quellcode

1. Private Sub tbText_KeyDown(ByVal sender As System.Object, ByVal e As System.Windows.Forms.KeyEventArgs) Handles tbText.KeyDown
2. Dim index As Integer = tbText.SelectionStart
3. tsslStatusText.Text = "Zeile " & tbText.GetLineFromCharIndex(index) & ", Spalte " & tbText.SelectionStart - tbText.GetFirstCharIndexOfCurrentLine + 1
4. End Sub

Das KeyDown wird nach dem Drücken der Taste ausgeführt. Hier führen reagieren wir, um in der StatusBar die Zeile und die Spalte anzeigen zu können.

TextBox: PreviewKeyDown

Quellcode

1. Private Sub tbText_PreviewKeyDown(ByVal sender As System.Object, ByVal e As System.Windows.Forms.PreviewKeyDownEventArgs) Handles tbText.PreviewKeyDown
2. tbText.ClearUndo()
3. End Sub

Dieses Event wird vor dem Tastendruck ausgelöst (natürlich nicht wirklich vorher, aber bevor die TextBox das mitkriegt). Dort löschen wir den UndoCache, da sonst ein Undo alles Rückgängig macht. So nur ein Zeichen (analog zu Notepad)

Das war ein kurzer Einstieg in das Arbeiten mit VS2010, für Fragen gibt es das entsprechende Unterforum. Falls ihr Fehler findet, könnt ihr diese natürlich direkt im Wiki verbessern. Das gesamte Visual-Studio-Projekt gibt es hier noch als .zip zum Download, falls jemand abkürzen möchte, oder an einer Stelle nicht weiterkommt.