

## Hauptmenü im XP-Stil

# Nouvelle Cuisine

Microsoft hat mit dem .NET Framework wirklich gute Arbeit geleistet. Allerdings sind viele Steuerelemente eher von zweifelhafter Güte. Vor allem die Menükomponente ist ein Quell des Unmuts, da sie praktisch auf dem Stand von Windows 3.1 stehen geblieben ist. dotnetpro zeigt, wie Sie Menüs im XP-Stil aufpeppen können.

Die Entwicklung einer kompletten Menü-Komponente wäre sehr aufwändig, da dies eine komplette Neuimplementierung der Klasse *MainMenu* erfordern würde – inklusive eines vollständigen Menü-Designers. Da das Aussehen eines Menüs jedoch von dessen Einträgen abhängt, genügt es, eine eigene *MenuItem*-Komponente mit dem gewünschten Aussehen zu implementieren.

Der Nachteil dieser Methode ist allerdings, dass bei allen Vorkommen der Klasse *System.Windows.Forms.MenuItem* die Klassenbezeichnung – hier in *XPControls.XPMenuItem* – geändert werden muss. Das ist an zwei Stellen notwendig. Einmal dort, wo die Menüpunkte deklariert werden, und weiterhin dort, wo sie erzeugt werden, also innerhalb der Region *Vom Windows Form-Designer generierter Code*. Die Umbenennung sollte erst erfolgen, nachdem Sie die Menüpunkte festgelegt haben, denn wenn Sie das Menü erweitern, geschieht dies wie-

### Listing 1

#### Die Deklarationen von *XPMenuItem*.

```
Color _colorGlyphBackLeft = Color.FromArgb(255,251,247);
Color _colorGlyphBackRight = Color.FromArgb(214,211,206);
Color _colorTextBackground = Color.FromArgb(255,251,247);
Color _colorTextEnabled = Color.Black;
Color _colorTextDisabled = Color.Gray;
Color _colorSelected = Color.FromArgb(180,190,214);
Color _colorSelectedFrame = Color.FromArgb(8,36,107);
Color _colorCheckedItem = Color.FromArgb(214, 215, 222);
Color _colorCheckedSelected = Color.FromArgb(132,146,181);
Color _colorSeparator = Color.FromArgb(165, 166, 165);
Bitmap _glyph = null;
int _lineIndent = 5;
int _textIndent = 5;
int _glyphAreaWidth = 25;
int _menuItemHeight = 22;
```

der mithilfe der *MenuItem*-Klasse, und Sie müssen die Änderungen erneut für die neuen Menüpunkte vornehmen. Der Designer zeigt die Menüpunkte zur Entwurfszeit zwar noch nicht korrekt an, die neuen Eigenschaften lassen sich jedoch im Eigenschaftfenster ändern. Dieser Weg ist zwar nicht allzu aufwändig und relativ schnell programmiert, dennoch ist der Quellcode für einen kompletten Abdruck zu umfangreich. Sie finden ihn vollständig auf der Heft-CD.

### Eigenschaften für *MenuItem*

Als Basisklasse für die neue Klasse *XPMenuItem* dient *MenuItem* aus *System.Windows.Forms*. Da diese aber faktisch gar nichts kann, müssen Sie die zusätzlichen Eigenschaften und Möglichkeiten erst einmal selbst programmieren. Dazu gehören unter anderem die folgenden:

- Aufteilung des Menüpunkts in Symbolbereich und Textbereich,
- Trennlinie mit einer eigenen Höhe,
- Highlight-Effekte bei darüber schwebender Maus sowie bei einem akti-

- vierten *MenuItem* (Checked-Item),
- frei definierbare Farben,
- automatische Transparenz für eine zugewiesene Grafik,
- Ausblenden nicht benötigter Eigenschaften.

Die Deklaration der benötigten neuen Felder für die Klasse ist nicht weiter kom-

### Auf einen Blick

#### Autor



**Frank Eller** ist .NET-Programmierer der ersten Stunde. Er ist Redner auf Konferenzen, Autor für Addison-Wesley und schreibt unter anderem für die dotnetpro und MSDN Online.

dotnetpro.code  
A0205Menu



Sprachen C#

Technik Menüs

Voraussetzungen Visual Studio .NET 2003, .NET Framework 1.1

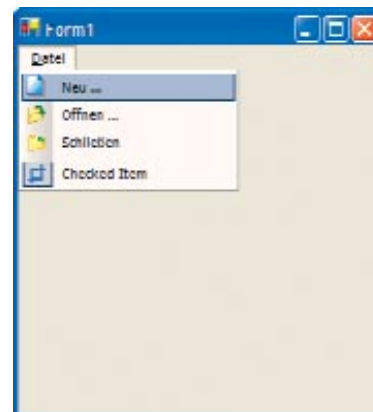


Abbildung 1 Das *MenuItem* im neuen Look in der Testapplikation.

## Listing 2

### Eigenschaften *Glyph* und *OwnerDraw* im Designer ausblenden.

```
[Category("Appearance")]
[Browsable(true)]
[EditorBrowsable(EditorBrowsableState.Always)]
[DefaultValue(null)]
public Bitmap Glyph {
    get { return this.glyph; }
    set {
        this.glyph = value;
        //Hintergrundfarbe transparent machen
        //Transparente Farbe ist oben links
        Pixel 1,1
        if (this.glyph != null) {
            Color transColor =
                this.glyph.GetPixel(1, 1);
            this.glyph.MakeTransparent(transColor);
        }
    }
}

[Browsable(false)]
[EditorBrowsable(EditorBrowsableState.Never)]
public new bool OwnerDraw {
    get { return true; }
    set { base.OwnerDraw = true; }
}
```

pliziert. Neben den Farben sind noch die Randabstände für den Separator sowie den Text festzulegen, außerdem die Grafik des Symbols, deren Breite sowie die

Höhe eines Menüpunkts. Diese Höhe kann abhängig von der Schriftart variabel sein, weshalb sie stets neu berechnet wird. Die Deklarationen zeigt Listing 1.

Die Deklaration der Eigenschaften erfolgt in der gleichen Weise. Das Verhalten zur Entwurfszeit wird über Attribute gesteuert. Das Attribut *CategoryAttribute* steht dabei für die Kategorie des Eigenschaftensfensters, in der die Eigenschaft angezeigt werden soll.

Das Attribut *BrowsableAttribute* gibt an, ob die Eigenschaft im Eigenschaftensfenster überhaupt angezeigt wird. Exemplarisch hier eine Implementierung:

```
[Category("Appearance")]
[Browsable(true)]
public Color ColorGlyphBackLeft {
    get {
        return this.colorGlyphBackLeft;
    }
    set {
        this.colorGlyphBackLeft = value;
    }
}
```

Die übrigen Eigenschaften funktionieren bis auf wenige Ausnahmen analog. Die Eigenschaften *Glyph* und *OwnerDraw* erfordern allerdings ein wenig mehr Aufwand. *Glyph* steht für die anzuzeigende Grafik, deren Hintergrund automatisch transparent gemacht werden soll. *OwnerDraw* ist eine geerbte Eigenschaft, die immer *true* sein muss, da anderenfalls die selbst implementierten

Methoden zum Zeichnen nicht aufgerufen werden. Deshalb muss diese Eigenschaft sowohl aus dem Eigenschaftensfenster als auch aus der IntelliSense-Hilfe verschwinden. Ein Entfernen ist unmöglich, da sie in der Basisklasse definiert ist und geerbt wird. Bleibt nur, dem Entwickler vorzugaukeln, die Eigenschaft sei gar nicht vorhanden, indem sie sowohl im Eigenschaftensfenster als auch in IntelliSense ausgeblendet wird. Die Implementierungen sehen Sie in Listing 2.

Welches Pixel im Fall der *Glyph*-Eigenschaft für die transparente Farbe steht, bleibt dem Entwickler überlassen. Die Eigenschaft *OwnerDraw* ist so zu implementieren, dass sie immer *true* ist, auch wenn ihr ein anderer Wert zugewiesen wird. Über *BrowsableAttribute* sowie *EditorBrowsableAttribute* wird sie sowohl im Eigenschaftensfenster als auch für die IntelliSense-Funktion versteckt.

### Größe und Hintergrund

Beim Zeichnen eines Menüpunkts passieren zwei Dinge. Zunächst wird die Methode *OnMeasureItem()* für jeden Menüpunkt aufgerufen, um seine Größe festzulegen. Die Breite ist dabei abhängig vom eingetragenen Text, die Höhe von der Schriftart. Danach wird *OnDrawItem()* aufgerufen und der Menüpunkt gezeichnet. Um den Aufruf kümmert sich .NET, Sie müssen ihn lediglich imple-

## Listing 3

### Die Methoden *OnMeasureItem()* und *OnDrawItem()*.

```
protected override void OnMeasureItem(MeasureItemEventArgs e) {
    // Methode der Basisklasse immer zuerst aufrufen
    base.OnMeasureItem(e);
    if (Text == "-") {
        e.ItemHeight = 3;
        return;
    }
    // Achtung: "&" wird nicht dargestellt und darf nicht
    // mitgerechnet werden
    string myText = Text.Replace("&", "");
    int textwidth = (int)(e.Graphics.MeasureString(myText,
        SystemInformation.MenuFont).Width);
    // Höhe des Menüpunkts berechnen
    e.ItemHeight = SystemInformation.MenuHeight;
    if (Parent == Parent.GetMainMenu()) {
        e.ItemWidth = textwidth + 6;
        e.ItemHeight = 16;
    } else {
        e.ItemWidth = Math.Max(160, textwidth + this._glyphAreaWidth);
        e.ItemHeight = this._menuItemHeight;
    }
}

protected override void OnDrawItem(DrawItemEventArgs e) {
    // Methode der Basisklasse immer zuerst aufrufen
    Graphics g = e.Graphics;
    Rectangle bounds = e.Bounds;
    // Status des Menüpunkts ermitteln
    bool selected = (e.State & DrawItemState.Selected) ==
        DrawItemState.Selected;
    bool topLevel = (Parent == Parent.GetMainMenu());
    bool hasGlyph = this._glyph != null;
    bool hotLight = (e.State & DrawItemState.HotLight) ==
        DrawItemState.HotLight;
    // Hintergrund zeichnen
    DrawBackground(g, bounds, selected, hotLight, topLevel, hasGlyph);
    // Text schreiben
    if (Text == "-") {
        DrawSeparator(g, bounds);
    } else {
        DrawText(g, bounds, topLevel);
    }
}
```

### Listing 4

#### Die Methoden DrawBackground() und DrawSeparator().

```
private void DrawBackground(Graphics g, Rectangle bounds, bool selected, bool hotLight,
    bool topLevel, bool hasGlyph) {
    // Unterscheidung Hauptpunkt oder Unterpunkt
    if (topLevel) {
        DrawTopItem(g, bounds, selected, hotLight);
    } else {
        DrawNormalItem(g, bounds, selected, hotLight);
        if (hasGlyph && !this.Checked) {
            DrawGlyph(g, bounds, selected);
        } else if (this.Checked) {
            DrawItemChecked(g, bounds, hasGlyph, selected);
        }
    }
}

private void DrawSeparator(Graphics g, Rectangle bounds) {
    Pen pen = new Pen(this._colorSeparator);
    int y = bounds.Y + (bounds.Height / 2);
    g.DrawLine(pen, bounds.X + this._glyphAreaWidth + this._lineIndent + 5, y, bounds.X
        + bounds.Width, y);
}
```

mentieren. *OnDrawItem()* ist recht umfangreich, weshalb sie auf mehrere Methoden verteilt wurde. Listing 3 zeigt *OnMeasureItem()* und *OnDrawItem()*. Schon an der Methode *OnDrawItem()* ist zu se-

hen, dass mehrere Zustände zu unterscheiden sind. Ein *MenuItem* kann untergeordnet sein oder an oberster Stelle stehen (*topLevel*), es kann ausgewählt sein (*selected*), eine Grafik enthalten oder nicht

(*hasGlyph*) sowie hervorgehoben sein (*hotLight*), wenn die Maus darüber steht. All diese Zustände muss die Komponente beim Zeichnen berücksichtigen, was die entsprechenden Methoden naturgemäß ein wenig aufbläht.

Zuerst ist der Hintergrund an der Reihe. Dabei wird unterschieden zwischen einem Menüpunkt, der an oberster Stelle steht, und allen anderen. Steht er nicht an oberster Stelle, sind noch weitere Eigenschaften von Bedeutung, nämlich ob es sich um einen markierten Menüpunkt mit einem Häkchen handelt und ob er eine Grafik besitzt. Die einfachste Version ist die eines Separators, der lediglich eine Trennlinie vorstellt. Das sehen Sie in Listing 4.

Das Zeichnen des Hintergrunds zeigt Listing 5. Die Methode *DrawNormalItem()* ist ein wenig komplex, da sie auch den Bereich für die Grafik zeichnen muss. In Office 2003 befindet sich dort ein Farbverlauf, der mithilfe eines *LinearGradientBrush* nachgebildet wird.

Die Klasse *ControlPaint*, die ebenfalls zum Einsatz kommt, enthält einige interessante Methoden, gerade zum Zeichnen von Steuerelementen. Allerdings ist auch hier die Funktionalität ein wenig eingeschränkt und die Methoden reichen bei

### Listing 5

#### Die Methoden DrawTopItem() und DrawNormalItem().

```
private void DrawTopItem(Graphics g, Rectangle bounds, bool selected,
    bool hotLight) {
    if (hotLight || selected) {
        // Maus über dem Element oder angeklickt
        if (hotLight) {
            g.FillRectangle(new SolidBrush(this._colorSelected), bounds);
            g.DrawRectangle(new Pen(this._colorSelectedFrame), bounds.X,
                bounds.Y, bounds.Width - 1, bounds.Height - 1);
        } else {
            // MenuItem ist angeklickt
            g.FillRectangle(new SolidBrush(this._colorTextBackground),
                bounds);
            ControlPaint.DrawBorder3D(g, bounds.Left, bounds.Top,
                bounds.Width, bounds.Height, Border3DStyle.Flat,
                Border3DSide.Top | Border3DSide.Left | Border3DSide.Right);
            return;
        }
    } else {
        // Keine Maus, nicht angeklickt
        g.FillRectangle(SystemBrushes.Control, bounds);
    }
}

private void DrawNormalItem(Graphics g, Rectangle bounds, bool selected,
    bool hotLight) {
    // Maus über MenuItem oder angeklickt
    if ((hotLight || selected) && this.Enabled) {
        g.FillRectangle(new SolidBrush(this._colorSelected), bounds);
        g.DrawRectangle(new Pen(this._colorSelectedFrame), bounds.X,
            bounds.Y, bounds.Width - 1, bounds.Height - 1);
    } else {
        // Menüpunkt, normal, nicht ausgewählt
        // Hintergrund für Glyph
        Rectangle glyphBounds = bounds;
        glyphBounds.Width = this._glyphAreaWidth;
        //Farbverlauf zeichnen
        LinearGradientBrush gradBrush = new LinearGradientBrush(glyphBounds,
            this._colorGlyphBackLeft, this._colorGlyphBackRight,
            LinearGradientMode.Horizontal);
        Blend gradBlend = new Blend(3);
        gradBlend.Positions[0] = 0.0f;
        gradBlend.Positions[2] = 1.0f;
        gradBlend.Factors[0] = 0.0f;
        gradBlend.Factors[2] = 1.0f;
        gradBlend.Positions[1] = 0.5f;
        gradBlend.Factors[1] = 0.5f;
        gradBrush.Blend = gradBlend;
        g.FillRectangle(gradBrush, glyphBounds);
        // Hintergrund für Text zeichnen
        bounds.X += this._glyphAreaWidth;
        bounds.Width -= this._glyphAreaWidth;
        g.FillRectangle(new SolidBrush(this._colorTextBackground), bounds);
    }
}
```

## Listing 6

### Bitmap und Text zeichnen.

```
private void DrawGlyph(Graphics g, Rectangle bounds, bool selected) {
    // Zeichnen, wenn ausgewählt, mit Schatten
    int x = bounds.Left + ((this._glyphAreaWidth - this._glyph.Width) / 2);
    int y = bounds.Top + ((this._menuItemHeight - this._glyph.Height) / 2);
    if (this.Enabled) {
        if (selected) {
            // Mit Schatten
            ControlPaint.DrawImageDisabled(g, this._glyph, x, y, Color.Black);
            g.DrawImage(this._glyph, x - 2, y - 2);
        } else {
            // Ohne Schatten
            g.DrawImage(this._glyph, x, y);
        }
    } else
        // Disabled
        ControlPaint.DrawImageDisabled(g, this._glyph, x, y,
            this._colorTextDisabled);
}

private void DrawText(Graphics g, Rectangle bounds, bool topLevel) {
    // Text mit Hotkey schreiben
    StringFormat stringFormat = new StringFormat();
    stringFormat.HotkeyPrefix = HotkeyPrefix.Show;
    // "&" nicht berücksichtigen
    string myText = this.Text.Replace("&", "");
    int textWidth = (int)(g.MeasureString(myText,
        SystemInformation.MenuFont).Width);
    int x = 0;
    Brush brush = this.Enabled ? new SolidBrush(this._colorTextEnabled)
        : new SolidBrush(this._colorTextDisabled);
    if (topLevel)
        x = bounds.Left + (bounds.Width - textWidth) / 2;
    else
        x = bounds.Left + this._glyphAreaWidth + this._textIndent;
    int y = topLevel ? bounds.Top + 3 : (bounds.Top + ((bounds.Height
        - SystemInformation.MenuFont.Height) / 2));
    g.DrawString(this.Text, SystemInformation.MenuFont, brush, x, y,
        stringFormat);
}
```

Weitem nicht für alles aus, was man an Steuerelement-Design umsetzen will. Der Hintergrund eines Menüpunkts jedoch stellt keine besondere Herausforderung dar und ist schnell implementiert.

### Markierte Menüpunkte zeichnen

Der nächste Schritt besteht im Zeichnen eines markierten *MenuItem*. Jedes *MenuItem* besitzt die Eigenschaft *CheckedItem*, die angibt, ob beim Anklicken ein Häkchen vor den Text gesetzt werden soll. Die Eigenschaft *RadioCheck* bestimmt das Aussehen dieses Häkchens. Ist sie *true*, wird statt eines Häkchens ein Punkt gezeichnet, eben wie bei einem *RadioButton*.

In diesem Fall kommt eine weitere Möglichkeit hinzu, denn der Menüpunkt könnte ja eine Grafik enthalten. Dann soll natürlich kein Häkchen, sondern die Grafik gezeichnet werden. Um trotzdem zu zeigen, dass der Menüpunkt markiert ist, werden Häkchen und Grafik hervorgehoben gezeichnet, allerdings nur im Symbolbereich. Dadurch ergeben sich mehrere Konstellationen, die auch in Office2003 zu beobachten sind:

- Das *MenuItem* besitzt keine Grafik. Dann wird entweder ein Häkchen oder ein Punkt gezeichnet und diese werden hinterlegt.
- Das *MenuItem* besitzt eine Grafik. Dann wird diese gezeichnet und hinterlegt.
- Das *MenuItem* ist markiert und die

Maus steht darüber. In dem Fall muss der gesamte Menüpunkt hinterlegt werden, wobei die Farbe im Symbolbereich dunkler sein muss.

Für das Häkchen oder den Auswahlknopf verwendet Windows übrigens keine Icon-Dateien, sondern eine besondere Schriftart, die auf jedem System vorhanden sein sollte. Es handelt sich um die Schriftart Marlett, die auch die Symbole in den Titelleisten der Fenster enthält. Wenn Sie diese einmal aus Ihrem System löschen, werden Sie feststellen, dass die Minimieren-, Maximieren- und Schließen-Kästchen plötzlich völlig anders aussehen. Geben Sie den Buchstaben B für das Häkchen und I für den Auswahlknopf ein. Die Methode *DrawItemChecked()*, die einen solchen markierten Menüpunkt zeichnet, finden Sie auf der Heft-CD.

### Der letzte Schritt: die Bitmaps und der Text

Nun fehlen nur noch der Text und das Zeichnen der Bitmap, das die Methode *DrawGlyph()* übernimmt. Auch hier gibt es eine besondere Anforderung, denn wenn die Maus über dem Menüpunkt schwebt, soll die Grafik etwas „angehoben“ werden und einen Schatten werfen. Dazu wird die Grafik versetzt gezeichnet. Über die Methode *DrawImageDisabled()* der Klasse *ControlPaint* lässt sich auch

ein Schatten erzeugen. Schließlich noch einen Text als letztes Element zu zeichnen, ist eine triviale Angelegenheit. Beide Methoden finden Sie in Listing 6.

### Fazit

Die hier gezeigte Implementierung ist sicherlich nicht optimal. Wie bereits angesprochen, wäre dazu die Klasse *MainMenu* zu überschreiben, doch deren Implementierung ist, vorsichtig ausgedrückt, etwas aufwändiger. Das Resultat des neuen *XPMMenuItem* kann sich jedoch sehen lassen. Es funktioniert nicht nur in einem Hauptmenü, sondern auch in Kontextmenüs. Mehr über die Entwicklung von Komponenten finden Sie im Übrigen in den Büchern [1], [2] und [3]. Die Implementierung einer zu diesem Menü passenden Toolbar beschreibt [4]. Bleibt nur noch, Ihnen viel Spaß mit einem ansprechenden Menü zu wünschen.

||||||

[1] Frank Eller, Michael Kofler, Visual C#, Addison-Wesley, ISBN 3-8273-2073-9

[2] Andreas Maslo, Jörg Freiberger, .NET Framework Developers Guide, Markt+Technik, ISBN 3-8272-6142-2

[3] Arne Schäpers, Rudolf Huttary, Dieter Bremes, C# Kompendium, Markt+Technik, ISBN 3-8272-6015-9

[4] Eine eigene Toolbar im Office-Stil bauen: [www.frankeller.de/default.aspx?show=bab7fa0c-37a8-4e5b-9201-49316fef27fb](http://www.frankeller.de/default.aspx?show=bab7fa0c-37a8-4e5b-9201-49316fef27fb)