Robin Verschuur is supervisor system development bij QIOS

Geavanceerde UI-programmering in .NET

HET ONTWIKKELEN VAN VISUELE ASPECTEN IN EEN .NET CONTROL

Programmeurs van gebruikersinterfaces in Windows komen vaak hetzelfde probleem tegen. De interface van de applicatie ziet er meestal niet zo geavanceerd uit als bijvoorbeeld Microsoft Word of Outlook. Deze hebben effecten zoals overlopende kleuren (gradients) en verschillend gekleurde borders en schaduwen. De toolbox van Visual Studio bevat enkel controls die juist deze grafisch geavanceerde eigenschappen niet implementeren. Men kan meestal niet veel meer dan de achtergrond- en voorgrondkleur en het type border kiezen. Het .NET-platform bevat echter een uitstekende mogelijkheid om zelf controls aan deze toolbox toe te voegen. Zodoende kunnen controls ontwikkeld worden die deze eigenschappen wel implementeren. Dit artikel bevat een beschrijving hoe programmeurs een paar van deze visuele aspecten kunnen ontwikkelen in een .NET-control.

Om een control in .NET te ontwikkelen met specifieke functionaliteit, zoals de weergave van een dikke border met een willekeurige kleur, moet gebruik gemaakt worden van de *System.Windows.Forms.Control* class. Wat zijn de werking en de mogelijkheden van deze control?

System.Windows.Forms.Control

Elke TextBox, Label, Panel, Form of DataGrid in het.NET Framework erft van *System.Windows.Forms.Control*. Deze control implementeert de functionaliteit die nodig is om een grafisch element op het scherm weer te geven. Het is een omhulsel voor de basis van Windows, het Window, en zorgt ervoor dat dit Window in de objectgeoriënteerde .NET-omgeving gebruikt kan worden. Het bevat de mogelijkheid om op het scherm te tekenen en om gebruikersacties zoals bewegingen van de muis of het ingeven van toetsencombinaties af te handelen.

Voor het afhandelen van gebruikersacties zijn events aan de control toegevoegd die de meeste operaties afdekken. Hierbij kan gedacht worden aan KeyDown, KeyUp, MouseMove, MouseDown, MouseUp of Click. Wanneer op een gebruikersactie gereageerd moet worden, wordt een EventHandler voor deze actie gemaakt. Het is ook mogelijk om een nieuwe implementatie aan de bijbehorende OnEventname-methode te geven, bijvoorbeeld OnClick. Doordat van control geërfd wordt, bevelen we aan de tweede methode te hanteren, dus om een nieuwe implementatie uit te voeren met de OnEventname-methode.

Voor het presenteren van een control op een Windows Form zijn er twee events aan de control toegevoegd: Paint en PaintBackground met de OnPaint- en OnPaintBackground-methode. Wanneer een control getekend dient te worden, worden deze twee methodes aangeroepen. Dit gebeurt bijvoorbeeld als de control voor het eerst gepresenteerd wordt, als er een ander Window overheen gesleept wordt of als de Form vanuit een geminimaliseerde status weer zichtbaar wordt gemaakt. Het is ook mogelijk om zelf aan te geven dat de control opnieuw getekend dient te worden. Dit gebeurt met de Refresh()- of Invalidate()-methodes.

Afbeelding 1. "I clicked on the Control"

Afbeelding 2. Niet het gewenste resultaat

Afbeelding 3. Netjes binnen de lijnen

Codevoorbeeld 1 bevat een implementatie van een dergelijke control. Het bevat een control met de naam MyControl en implementeert de OnClick-, OnPaintBackground- en OnPaint-methode. Wanneer je deze control op een Form zet, wordt een rode dikke border getekend. Als je er vervolgens op klikt, wordt de tekst "I clicked on the Control" zichtbaar; zie afbeelding 1.

Via deze methode is het mogelijk om controls te maken die zichzelf presenteren en gebruikersacties afhandelen. Dit gaat voor de meeste controls goed. Wanneer je een control maakt die andere controls bevat en daar dan een 20 pixels brede border omheen tekent, wordt een probleem geconstateerd.

Om dit te demonstreren is er een andere control gemaakt, genaamd *MyUserControl*. Als basis wordt van UserControl geërfd. Deze erft ook van Control, maar implementeert functionaliteit voor ScrollBars. Aan de OnPaintBackground-methode zijn geen wijzigen aangebracht. Als je nu een TextBox op MyUserControl plaatst, dan ziet het er allemaal goed uit. Maar als je MyUserControl kleiner maakt zodat de TextBox niet meer past, verschijnt er een ongewenst resultaat zoals je in afbeelding 2 kunt zien.

De TextBox wordt over de border heen getekend en de ScrollBar ligt aan de buitenkant van de border. Bij het gewenste resultaat ligt de ScrollBar binnen de border en overschrijdt de TextBox de border niet.

Controls met willekeurige borders

Om dit probleem op te lossen bevat een window het zogenaamde Non Client Area-gedeelte. Een Window kan opgedeeld worden in twee gedeeltes: de Client Area en de Non Client Area. De Client Area is het gedeelte waar controls zoals Labels en TextBoxen zich bevinden, de Non Client Area is het gedeelte waar de Borders, Caption en ScrollBars getekend worden. Een standaard window in Windows heeft een Caption, een Border en een grijs gedeelte. De Caption en Borders liggen in de Non Client Area. Het grijze gedeelte is de Client Area. Wanneer nu wordt gezorgd dat de borders op de Non Client Area getekend worden, zal de in de Client Area gelegen TextBox de borders niet overschrijden.

.NET Control ondersteunt de Non Client Area niet standaard. Om hiermee te kunnen werken moet de programmeur gebruikmaken van de WndProc-methode. De control-class van .NET biedt de mogelijkheid deze methode te overschrijven, zodat er een willekeurige implementatie gegeven kan worden aan bepaalde Windows-messages. De messages die betrekking hebben op de Non Client Area zijn WM_NCCALCSIZE en WM_NCPAINT. WM_NCCALCSIZE geeft aan welk gedeelte van de control de Non Client Area betreft. WM_NCPAINT biedt de mogelijkheid om de Non Client Area te tekenen.

In codevoorbeeld 2 is de implementatie van de OnPaintBackground naar de afhandeling van WM_NCPAINT in de WndProc-methode verplaatst, zodat de borders op de Non Client Area getekend worden. Daarnaast is aangeven welk gedeelte van de control de Non Client Area betreft. Dit is gedaan in de WM_NCCALCSIZE-afhandeling. Het resultaat is zichtbaar in codevoorbeeld 2. De definitie van de WinApi-structuren en -methodes is weggelaten. Deze zijn echter wel beschikbaar in de codevoorbeelden die zijn te downloaden. Zoals zichtbaar is in afbeelding 3 wordt nu de ScrollBar binnen de borders getekend en overschrijdt de TextBox de borders niet.

Op dit moment wordt een dikke rode border getekend om de control. Er kan echter alles op de Non Client Area getekend

```
namespace AdvancedGui
{
    using System;
    using System.Drawing;
    using System.Windows.Forms;

    public class MyControl : System.Windows.Forms.Control
    {
        public MyControl()
        {
            //Laat Windows weten dat we de control zelf tekenen
            this.SetStyle(ControlStyles.ResizeRedraw |
                ControlStyles.UserPaint |
                ControlStyles.DoubleBuffer, true);
        }

        protected override void OnClick(EventArgs e)
        {
            base.OnClick (e);
            //Vul de Tekst en Refresh de control
            this.Text = "I clicked on the Control";
            this.Refresh();
        }

        protected override void OnPaintBackground(PaintEventArgs e)
        {
            base.OnPaintBackground (e);

            int borderWidth = 20;
            //Teken rode Rectangle
            Pen myPen = new Pen(Color.Red, borderWidth);
            e.Graphics.DrawRectangle(myPen,
                borderWidth / 2, borderWidth / 2,
                this.ClientSize.Width - borderWidth,
                this.ClientSize.Height - borderWidth);
            //Ruim gebruikte resources op
            myPen.Dispose();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            base.OnPaint (e);
            SolidBrush myForeBrush = new SolidBrush(this.ForeColor);
            //Lijn de tekst in het midden uit.
            StringFormat myFormat =
                new StringFormat(StringFormat.GenericDefault);
            myFormat.Alignment |= StringAlignment.Center;
            myFormat.LineAlignment |= StringAlignment.Center;
            //Teken de tekst
            e.Graphics.DrawString(this.Text, this.Font, myForeBrush,
                this.ClientRectangle, myFormat);
            //Ruim gebruikte resources op
            myForeBrush.Dispose();
            myFormat.Dispose();
        }
    }
}
```

Codevoorbeeld 1.

worden: afbeeldingen, gradients en patronen. Ook kan de programmeur er voor kiezen om standaard resize-mogelijkheden aan te geven op de Non Client Area. Dit gebeurt door het WM_NCHITTEST-bericht te implementeren. Wanneer je dat doet, wordt als resultaat van dit bericht aangegeven op welk gedeelte van de control geklikt is, bijvoorbeeld rechtsonder. Er wordt dan automatisch een resize-cursor weergegeven, waarna de gebruiker de control groter of kleiner kan maken.

```
protected override void WndProc(ref Message m)
{
    //Teken de Non-Client-Area.
    if (m.Msg == WM_NCPAINT)
    {
        base.WndProc(ref m);
        //Haal de Windows DC van de control op
        IntPtr myHdc = GetWindowDC(m.HWnd);
        //Teken de Rectangle
        Graphics myGraphics = Graphics.FromHdc(myHdc);
        Pen myPen = new Pen(Color.Red, borderWidth);
        myGraphics.DrawRectangle(myPen, borderWidth / 2,
            borderWidth / 2, this.Width - borderWidth,
            this.Height - borderWidth);

        //Ruim gebruikte resources op
        myPen.Dispose();
        myGraphics.Dispose();
        ReleaseDC(m.HWnd, myHdc);
    }
    //Geef aan wat de Client-Area is.
    else if (m.Msg == WM_NCCALCSIZE)
    {
     //Hier kan nog onderscheid gemaakt worden tussen m.Wparam = 0 of 1
     //Dit is geïmplementeerd in de codevoorbeelden op Internet en
     //hier voor de eenvoud weggelaten.
        RECT clientRect = (RECT)Marshal.PtrToStructure(
            m.LParam, typeof(RECT));
        //Haal de borders van de clientRect af
        clientRect.Add(borderWidth, borderWidth,
            -borderWidth, -borderWidth);
        //Zet de nieuwe clientRect
        Marshal.StructureToPtr(clientRect, m.LParam, false);
        //Roep base aan, zodat de ScrollBars toegevoegd worden.
        base.WndProc(ref m);
        //Geef 0 als resultaat.
        m.Result = IntPtr.Zero;
    }
    else
    {
        base.WndProc(ref m);
    }
}
```

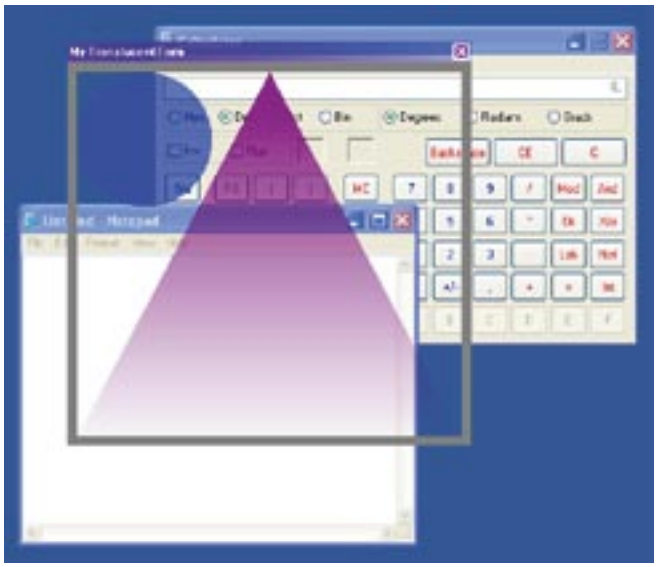
Codevoorbeeld 2.

Geanimeerde controls

Sinds Windows98 is het mogelijk om controls via animatie op het scherm te presenteren. Vooral menu's profiteren hiervan. De gedachtegang is dat het in de fysieke wereld niet mogelijk is om iets uit het niets zichtbaar te maken. Er gaat altijd eerst een beweging aan vooraf. Om dit te simuleren in de virtuele wereld kan animatie gebruikt worden. Op het moment dat een menu moet verschijnen, wordt het opengerold of met een fade-in zichtbaar gemaakt.

Om animatie in een .NET Control te implementeren moet de programmeur gebruikmaken van de mogelijkheden van de Windows API. In de User32-bibliotheek zit de functie AnimateWindow. Deze functie animeert controls op allerlei manieren. Een control kan uitgeklapt worden in elke richting of door middel van transparantie zichtbaar gemaakt worden. Er kan een tijd aan de functie worden meegegeven die bepaalt hoe lang de animatie moet duren.

Een goede plaats om dit te implementeren in een control is in de SetVisibleCore-methode. Op deze manier kan de control via de gebruikelijke Visible-property zichtbaar en onzichtbaar gemaakt worden. Codevoorbeeld 3 geeft een implementatie



Afbeelding 4. MyTranslucentForm

hiervan. Een eis voor het gebruik van AnimateWindow, wanneer een control op een ander window staat (een ChildWindow), is dat het WM_PRINT-bericht afgehandeld moet worden en dat de transparantiefunctionaliteit (blend) niet gebruikt kan worden. Het WM_PRINT-bericht wordt aangeroepen wanneer de control op een andere plaats dan het scherm getekend moet worden, bijvoorbeeld op een afbeelding die vervolgens voor de animatie gebruikt wordt. Het codevoorbeeld wat u kunt downloaden (zie de lijst met nuttige internetadressen) bevat een complete implementatie van MyAnimationControl. Wanneer de animatie gebruikt wordt voor een niet-ChildWindow, bijvoorbeeld een menu, kan de transparantiefunctionaliteit wel gebruikt worden.

Doorschijnende Forms

Een ander populair effect is een doorschijnende (translucent) Form. Hierbij blijven de onderliggende Forms zichtbaar onder de bovenliggende Form. Met dit effect worden ook de schaduwranden van een menu weergegeven. Een .NET Form bevat een eenvoudige methode om de Form egaal doorschijnend weer te geven. Dit gebeurt met de Opacity-property. Hier kan de programmeur een waarde van 0 t/m 1.0 aangeven wat de doorschijnendheid van de Form bepaalt. 1.0 is volledig zichtbaar, 0 is niet zichtbaar.

Windows bevat ook de mogelijkheid om slechts delen van een Form doorschijnend weer te geven, waarbij de programmeur per pixel de doorschijnendheid bepaalt. Een voorbeeld hiervan is het startscherm van Adobe Reader 6.0. Om dit in een .NET Form te implementeren wordt gebruik gemaakt van de Layered Window-functionaliteit. De User32-bibliotheek bevat sinds Windows 2000 de UpdateLayeredWindow-functie en de WS_EX_LAYERED Window Style. Deze maken het mogelijk een doorschijnende afbeelding op een Form te projecteren en de zichtbaarheid van elke pixel aan te geven. Afbeelding 4 geeft MyTranslucentForm weer. De Form in deze afbeelding is doorschijnend. De implementatie van deze Form is beschikbaar via de codevoorbeelden op internet.

Compatibiliteit

De voornoemde voorbeelden representeren enkele aspecten van geavanceerde UI-programmering in .NET. Er moet echter rekening worden gehouden met het feit dat deze voorbeelden slechts werken op Windows-systemen. De reden hiervoor is dat er gebruik wordt gemaakt van de Windows API en functies die pas sinds een specifieke Windows-versie beschikbaar zijn. Het Microsoft .NET Framework is echter beschikbaar voor veel

```
protected override void SetVisibleCore(bool value)
{
    if (value)
    {
        //Laat de control naar buiten rollen.
        AnimateWindow(this.Handle, 1000, AW_SLIDE |
            AW_HOR_POSITIVE | AW_VER_POSITIVE);
    }
    else
    {
        //Laat de control naar binnen rollen.
        AnimateWindow(this.Handle, 500, AW_HIDE | AW_SLIDE |
            AW_HOR_NEGATIVE | AW_VER_NEGATIVE);
    }
    base.SetVisibleCore (value);
}
```

Codevoorbeeld 3.

verschillende platformen. Wanneer een applicatie ook dient te werken op bijvoorbeeld Mono - het .NET-platform voor Linux - kan geen gebruik worden gemaakt van de Windows API. De gewenste functionaliteit moet dan op een andere manier worden geïmplementeerd.

Third party controls

Wanneer je grafische hoogwaardige controls in een applicatie wilt implementeren, is er de optie om deze op basis van de voorbeelden te ontwikkelen. Als de gewenste functionaliteit niet te specialistisch is, zijn er ook control-bibliotheken beschikbaar die grafisch hoogwaardige controls bevatten. Een voorbeeld hiervan is Qios.DevSuite.Components, beschikbaar via www.qiosdevsuite.com. Het voordeel van een dergelijke bibliotheek is meestal dat de prijs van een dergelijke bibliotheek niet opweegt tegen de ontwikkelkosten van grafisch hoogwaardige controls.

Al met al is het dus mogelijk om op het Microsoft .NET-platform applicaties met geavanceerde interfaces te ontwikkelen. Hierbij kan je de keuze maken om van Control te erven en de gewenste functionaliteit zelf te implementeren, of gebruik te maken van third party controls.

Robin Verschuur is supervisor system development bij QIOS BV (www.qios.nl). Zijn e-mailadres is rverschuur@qios.nl

Nuttige internetadressen

Codevoorbeelden: http:www.microsoft.nl/netmagazine of http://www.qiosdevsuite.com/dotnetmagazine8

Qios.DevSuite: http://www.qiosdevsuite.com

Windows overview: http://msdn.microsoft.com/library/en-us/winui/winui/windowsuserinterface/windowing/windows.asp

Layered Windows: http://msdn.microsoft.com/library/en-us/dnwui/html/layerwin.asp