Fons Sonnemans is zelfstandig software-development consultant en trainer.

Visueel programmeren met .NET

HET SCHRIJVEN VAN EEN HERBRUIKBARE WINDOWS FORMS DUALLIST-COMPONENT

Een slimme ontwikkelaar is lui en hanteert daarom de 'herhaal jezelf niet'-werkwijze. Zo stel ik enkele properties in een designer in zodat ik code niet opnieuw hoef te schrijven. Dit is gelukkig vaker mogelijk dan je in eerste instantie denkt. Hiervoor moet je wel eerst voor je probleem een generieke component ontwikkelen. Vervolgens kan je deze component op meer plaatsen toepassen, waardoor jouw productiviteit en kwaliteit verbetert. Hoe gaat het schrijven van een herbruikbare Windows Forms Duallist-component in zijn werk?

Een veel terugkomende functionaliteit in een Windows Forms-applicatie is het verplaatsen van een geselecteerd item van een listbox naar een andere listbox; zie afbeelding 1. Elke ontwikkelaar heeft dit waarschijnlijk al vele keren moeten bouwen. Slechts enkele zullen dit gedaan hebben op een generieke manier.

Per button kan je in de Click-eventhandler met zes regels code de verplaatsactie realiseren. Een voorbeeld van de benodigde code vind je in codevoorbeeld 1.

De code van codevoorbeeld 1 voldoet overigens niet aan de gebruikelijke kwaliteitseisen. De twee methodes bevatten 'dubbele code', een van de meest gemaakte programmeerfouten. Dit los je op door een derde methode aan te maken en deze vanuit de twee bestaande methoden aan te roepen met de 'source' en de 'destination' listbox als argument. Door de argumenten om te draaien is de werking aan te passen. Deze derde methode kan je dan het beste 'static' maken en in een extra helper-class plaatsen; zie codevoorbeeld 2. Dit laatste is vergelijkbaar met een Visual Basic-module. Hiermee is deze oplossing herbruikbaar in het project. Door deze helper-class in een aparte classlibrary-assembly te plaatsen en deze assembly vanuit de Windows-applicatie te refereren, is deze oplossing in meer projecten te gebruiken.

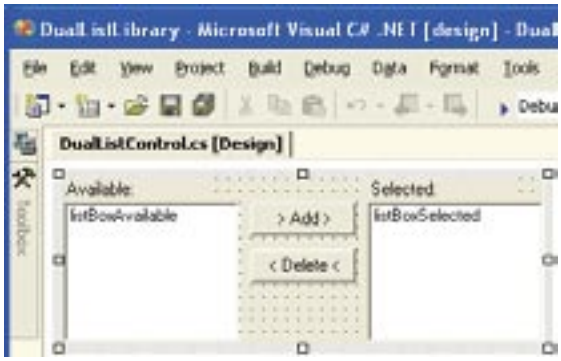
```
Private void buttonAdd_Click(object sender, System.EventArgs e) {
   int index = listBoxAvailable.SelectedIndex;
   if (index > -1) {
      listBoxSelected.SelectedIndex =
        listBoxSelected.Items.Add(listBoxAvailable.SelectedItem);
      listBoxAvailable.Items.RemoveAt(index);
      listBoxAvailable.SelectedIndex =
      (index > listBoxAvailable.Items.Count - 1) ? index - 1 : index;
   }
}

private void buttonDelete_Click(object sender, System.EventArgs e) {
   int index = listBoxSelected.SelectedIndex;
   if (index > -1) {
      listBoxAvailable.SelectedIndex =
        listBoxAvailable.Items.Add(listBoxSelected.SelectedItem);
      listBoxSelected.Items.RemoveAt(index);
      listBoxSelected.SelectedIndex =
        (index > listBoxSelected.Items.Count - 1) ? index - 1 : index;
   }
}
```

Codevoorbeeld 1. Click-eventhandlers in Form1

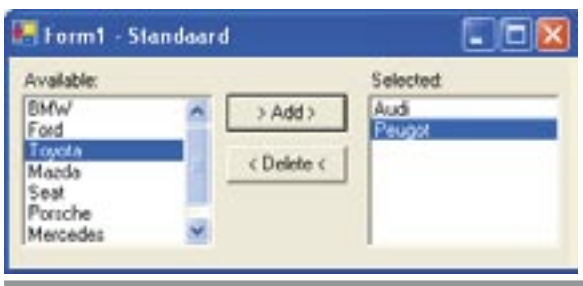
Deze oplossing heeft nog niet direct iets te maken met het onderwerp van dit artikel: visueel programmeren. Na het plaatsen van de controls moet de ontwikkelaar nog steeds de twee click-eventhandlers coderen. Vandaar dat ik voor dit probleem twee oplossingen ontwikkeld heb waardoor de ontwikkelaar in het formulier geen code meer hoeft te schrijven. In het eerste alternatief gebruik ik een user control. Deze oplossing heeft echter zijn beperkingen. Vandaar dat ik een oplossing ontwikkeld heb dat deze beperkingen niet heeft.

Oplossing 1 – DualListControl

Een veelgebruikte techniek voor visueel programmeren is het ontwikkelen van een user control. Op dit user control zijn de twee listboxes en buttons geplaatst; zie afbeelding 2.



Afbeelding 1. Verplaatsen van geselecteerde items



Afbeelding 2. Designer van DualListControl

Het DualListControl is een subklasse van de class *System.Windows.Forms.UserControl* bevat verder de twee Click-eventhandler-methoden. Om in de designer via het Properties Window de items van de twee listboxes te kunnen plaatsen, moet je twee extra read-only properties in de class opnemen. Deze properties (ItemsAvailable en ItemsSelected) hebben twee attributen; zie codevoorbeeld 3. Het *DesignerSerializationVisibility*-attribuut zorgt ervoor dat de inhoud bewaard wordt in de InitializeComponent-methode van het formulier. Het Editor-attribuut zorgt ervoor dat de 'String Collection Editor'-dialoog wordt getoond als je de properties wijzigt.

Het gebruik van het DualListControl is zeer eenvoudig. In de Toolbox zie je het control terug onder het tabje 'My User Controls'. Van daar sleep je het control op je verder lege formulier; zie afbeelding 3. Daarna kan je de property ItemsAvailable vullen met bijvoorbeeld een aantal automerken.

Dit DualListControl werkt prima, maar kent toch nog een aantal beperkingen. Zo kun je niet de Text-property van de buttons wijzigen. Dit is natuurlijk wel op te lossen door twee extra properties aan de DualListControl class toe te voegen; zie codevoorbeeld 4. Maar wanneer een groot aantal properties ingesteld moet kunnen worden, zoals Enabled, SelectedIndex en Sorted, leidt dit tot veel extra code die ook nog allemaal gedocumenteerd en getest moet worden.

Een veel grotere beperking is de starheid van het uiterlijk (de layout) van het control. Deze is alleen door de ontwikkelaar van het control aan te passen. Hierdoor is het control niet echt flexibel in te zetten. Het ontwikkelen van een user control bestaande uit verscheidene opgenomen controls is daarom niet mijn favoriete techniek.

Oplossing 2 – DualListComponent

Een veel minder bekende techniek is het ontwikkelen van een component. Een component is een object waarvan je in de Visual Studio.NET Designer de properties in kunt stellen. Dit wordt design time support genoemd. Een component schrijf je met een normale class die de *System.ComponentModel.IComponent* interface implementeert of, nog eenvoudiger, een subclass is van de class *System.ComponentModel.Component*.

Als uitgangspunt van de DualListComponent neem ik niet de controls zoals bij oplossing 1, maar neem ik het mechanisme van het verplaatsen van het geselecteerde item tussen de twee listboxes dat geactiveerd wordt op moment dat je op de button klikt. Dit mechanisme heeft dan ook drie kenmerken: Source, Destination en ActionButton. Deze kenmerken zijn in het UML class-diagram (zie afbeelding 4) weergegeven met drie public variabelen. De werkelijke implementatie is uitgevoerd met private variabelen en public properties get/set. Om design time support te ondersteunen werkt een component niet met publieke variabelen.

In de ActionButton property set-methode abonneer ik het object op het Click-event van de button. Hiervoor wordt de += operator gebruikt

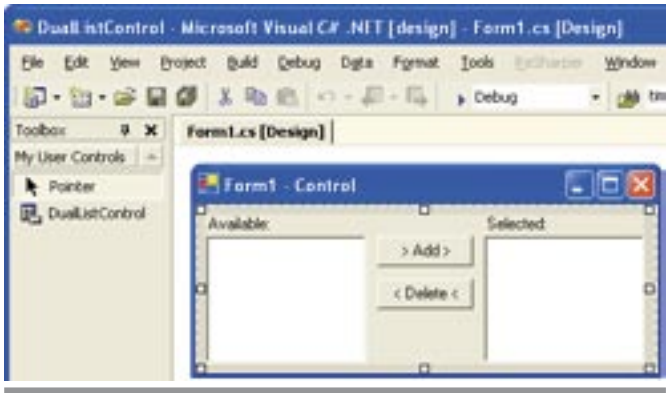
```
public sealed class ListBoxHelper
{
  private ListBoxHelper() {}  // constructor

  public static void SwitchItem(ListBox source, ListBox destination)
  {
    int index = source.SelectedIndex;
    if (index > -1)
    {
      destination.SelectedIndex =
        destination.Items.Add(source.SelectedItem);
      source.Items.RemoveAt(index);
      source.SelectedIndex =
        (index > source.Items.Count - 1) ? index - 1 : index;
    }
  }
}

public class Form1 : System.Windows.Forms.Form
{
  // overige code
  private void buttonAdd_Click(object sender, System.EventArgs e)
  {
    ListBoxHelper.SwitchItem(listBoxAvailable, listBoxSelected);
  }

  private void buttonDelete_Click(object sender, System.EventArgs e)
  {
    ListBoxHelper.SwitchItem(listBoxSelected, listBoxAvailable);
  }
}
```

Codevoorbeeld 2. EventHelper + Click-eventhandlers

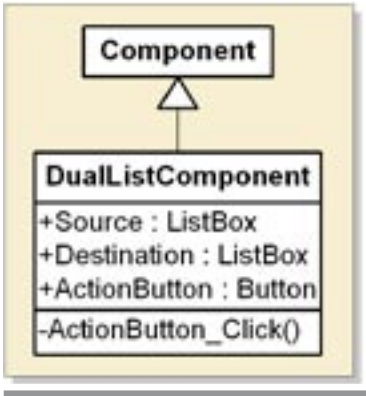
in combinatie met een EventHandler-delegate naar de ActionButton_Click-methode. In deze methode wordt de uiteindelijke verplaatsing uitgevoerd door de eerder ontwikkelde static SwitchItem-methode van de ListBoxHelper-class aan te roepen; zie codevoorbeeld 5.

Het gebruik van dit component is zeer eenvoudig. Ontwerp het formulier met de twee listboxes en buttons zonder de Click-eventhandlers. Daarna moet je twee DualListComponent-objecten als een soort onzichtbare control aan het formulier toevoegen. Daarvoor moet eerst het component toegevoegd worden aan de Toolbox. Dit gaat het eenvoudigst door de assembly van het component te compileren. Vervolgens sleep je de assembly vanuit een normale Windows Verkenner en laat je deze los op de Toolbox. Alle controls en componenten uit de assembly worden automatisch opgenomen in de Toolbox. Vanuit de Toolbox voeg je tweemaal het component toe aan het formulier. Deze componenten worden onder het formulier als icoon weergegeven; zie afbeelding 5.

Na het selecteren van dualListComponent1 kan je in het properties-window de Source-, Destination- en ActionButton-properties



Afbeelding 3. Toevoegen DualListControl aan Form1



Afbeelding 4. Het UML class-diagram

```
public class DualListControl :
  System.Windows.Forms.usercontrol
{
  private System.Windows.Forms.Button buttonDelete;
  private System.Windows.Forms.Button buttonAdd;
  private System.Windows.Forms.ListBox listBoxSelected;
  private System.Windows.Forms.ListBox listBoxAvailable;

  // Overige code:
  // - DualListControl() methode (constructor)
  // - Dispose() methode
  // - InitializeComponent() methode

  private void buttonAdd_Click(object sender, System.EventArgs e)
  {
    ListBoxHelper.SwitchItem(listBoxAvailable, listBoxSelected);
  }

  private void buttonDelete_Click(object sender, System.EventArgs e)
  {
    ListBoxHelper.SwitchItem(listBoxSelected, listBoxAvailable);
  }

  [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
  [Editor("System.Windows.Forms.Design.ListControlStringCollectionEditor,
  System.Design", typeof(System.Drawing.Design.UITypeEditor))]
  public ListBox.ObjectCollection ItemsAvailable
  {
    get
    {
      return listBoxAvailable.Items;
    }
  }

  [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
  [Editor("System.Windows.Forms.Design.ListControlStringCollectionEditor,
  System.Design", typeof(System.Drawing.Design.UITypeEditor))]
  public ListBox.ObjectCollection ItemsSelected
  {
    get
    {
      return listBoxSelected.Items;
    }
  }
}
```

Codevoorbeeld 3. DualListControl

```
[DefaultValue("> Add >")]
public string TextButtonAdd
{
get { return buttonAdd.Text; }
set { buttonAdd.Text = value; }
}

[DefaultValue("< Delete <")]
public string TextButtonDelete
{
get { return buttonDelete.Text; }
set { buttonDelete.Text = value; }
}
```

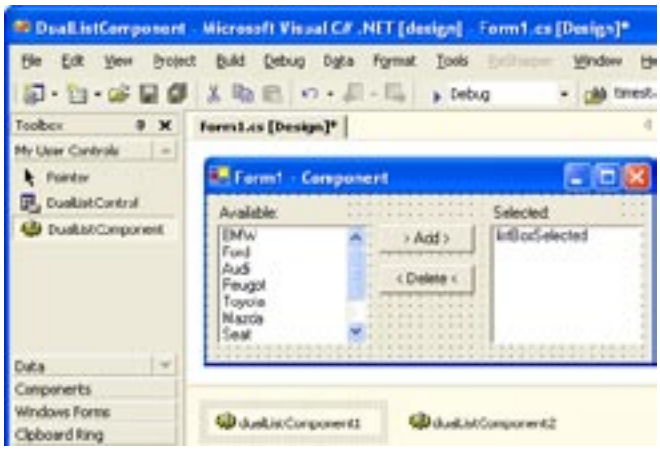
Codevoorbeeld 4. Extra properties DualListControl

```
public class DualListComponent : System.ComponentModel.Component
{
  private Button _actionButton;
  private ListBox _source;
  private ListBox _destination;

  public Button ActionButton
  {
    get { return this._actionButton; }
    set {
      this._actionButton = value;
      if (_actionButton != null)
      {
        _actionButton.Click += new EventHandler(ActionButton_Click);
      }
    }
  }

  public ListBox Source
  {
    get { return this._source; }
    set { this._source = value; }
  }

  public ListBox Destination
  {
    get { return this._destination; }
    set { this._destination = value; }
  }

  private void ActionButton_Click(object sender, EventArgs e)
  {
    ListBoxHelper.SwitchItem(Source, Destination);
  }
}
```

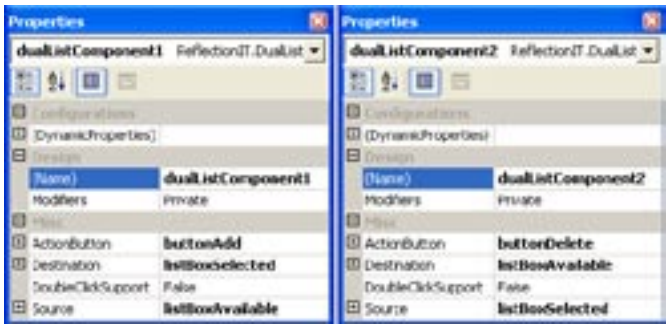
Codevoorbeeld 5. DualListComponent

```
private bool _doubleClickSupport;
private IButtonControl _acceptButton;

[DefaultValue(False)]
public bool DoubleClickSupport {
  get { return this._doubleClickSupport; }
  set {
    this._doubleClickSupport = value;
    if (value && Source != null) {
      Source.DoubleClick += new EventHandler(Source_DoubleClick);
      Source.Enter += new EventHandler(Source_Enter);
      Source.Leave += new EventHandler(Source_Leave);
    }
  }
}

private void Source_DoubleClick(object sender, EventArgs e) {
  ListBoxHelper.SwitchItem(Source, Destination);
}

private void Source_Enter(object sender, EventArgs e) {
  Form currentForm  = Source.FindForm();
  _acceptButton = currentForm.AcceptButton;
  currentForm.AcceptButton = ActionButton;
}

private void Source_Leave(object sender, EventArgs e) {
  Source.FindForm().AcceptButton = _acceptButton;
}
```

Codevoorbeeld 6. DualListComponent met DoubleClickSupport



Afbeelding 5. Twee DualListComponenten op Form1



Afbeelding 6. Instellen properties

instellen. Hierbij kun je de controls eenvoudig selecteren uit de keuzelijst. Dit herhaal je vervolgens voor dualListComponent2 waarbij je de Delete-button selecteert en de properties Source en Destination omdraait; zie afbeelding 6.

Je bent nu klaar. Je kunt het resultaat testen door het project op te starten. De applicatie doet nog steeds hetzelfde. Het formulier bevat echter geen code meer, waardoor de kans op fouten afneemt en de productiviteit toeneemt.

DoubleClick-ondersteuning

Het DualListComponent is overigens eenvoudig uit te breiden met opties zoals DoubleClick, Drag&Drop en MultiSelect. Deze opties worden vaak achterwege gelaten, omdat ze te complex zijn om ze iedere keer te ontwikkelen; vandaar dat je ze in een component moet onderbrengen.
Bij DoubleClick-ondersteuning moet je niet alleen het DoubleClick-event van de 'source' listbox afhandelen, maar ook de AcceptButton-property van het formulier wijzigen. Hiervoor moet je zes extra eventhandlers en een extra variabele in het formulier opnemen. Dit is veel werk en daardoor foutgevoelig. Bij het gebruik van het DualListComponent kan je volstaan met het instellen van de DoubleClickSupport-property. Een voorbeeld van de benodigde code vind je in voorbeeldcode 6.

Extra interessant

Visueel programmeren is met componenten zeer krachtig en eenvoudig. De beperkingen van de User Control-oplossing - extra properties en uiterlijke starheid - treden bij een component niet op. Dit maakt de componententechniek extra interessant. Vreemd genoeg zie ik deze techniek in de praktijk nog niet vaak toegepast. Hopelijk heeft dit artikel je geïnspireerd om je eigen componenten te schrijven en je applicaties meer en meer visueel te ontwikkelen. Dit zal de productiviteit en de betrouwbaarheid zeker ten goede komen.

Fons Sonnemans Fons is zelfstandig software-development consultant en trainer (www.reflectionit.nl). Sinds begin 2001 is hij bezig met .NET. Fons richt zich met name op het automatiseren van de automatisering. Zijn e-mailadres is fons.sonnemans@reflectionit.nl.