Peter van Ooijen is zelfstandig .NET-consultant en ontwikkelaar

Aan de slag met .NET Framework 3.0

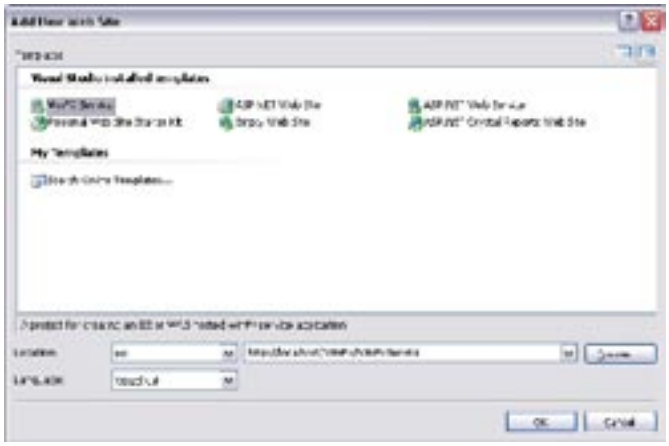
EEN APPLICATIE BOUWEN IN XAML VOOR DE TABLET PC

De afgelopen tijden zijn we als ontwikkelaars overspoeld met de nieuwste technologieën en nog veel fraaiere, of soms juist saaiere, bijbehorende namen. Al deze zaken bouwen voort op het .NET Framework, waarvan versie 2.0 zich heeft bewezen als een productieve basis voor het maken van applicaties. Sinds kort komt alles ook in naam bij elkaar en slokt .NET Framework 3.0 alles op wat nieuw is.

Voor het maken van applicaties voor Windows Vista is .NET 3.0 het platform bij uitstek, maar Windows Vista heeft niet het alleenrecht. De ontwikkeltools om .NET 3.0-applicaties te maken en de runtime-library om .NET 3.0-applicaties te draaien werken ook op Windows XP (SP2) en Windows Server 2003 (SP1). Dit maakt de overgang naar Windows Vista een stuk gemakkelijker voor ons ontwikkelaars. Je ontwikkelt tegen een uitgebreid Framework aan en de gebruiker kan de ontwikkelde applicaties zowel op Windows XP als op Windows Vista draaien. Op dat laatste platform zal het er natuurlijk wel wat mooier uitzien.
In dit artikel ga ik niet met een specifiek onderdeel van .NET Framework 3.0 aan de slag, maar proberen we een echte applicatie te maken. Hierin zal de volgende mix van oud en nieuw voorbij komen:

- Tablet pc: hier bouw ik verder op mijn artikel 'Software ontwikkelen voor de Tablet pc' in .NET Magazine #10
- Windows Vista: het nieuwe OS van huis uit voorzien van Tablet-functionaliteit
- Windows Presentation Foundation: het presentatiegedeelte van .NET Framework 3.0
- IIS als webserver
- Windows Communications Foundation: het communicatiegedeelte van .NET Framework 3.0

Bij het schrijven van de code zal ik geregeld terugkijken. We zullen zien dat veel zaken al langer bestonden in een iets andere, maar wel zeer vergelijkbare vorm. Zo lijkt het maken van een WCF-service op het maken van een klassieke ASP.NET-webservice, het opmaken van een WPF Windows-form vertoont grote overeenkomsten met het opmaken van een webpagina en als we bij de zogeheten commands in deze forms zijn aanbeland, kan zowel bij ASP.NET- als bij Delphi-ontwikkelaars een lichtje van herkenning gaan branden.



Afbeelding 1. Het maken van een WinFX service-template

De applicatie richt zich op Windows Vista, maar de vraag is of je Windows Vista ook moet gebruiken als ontwikkelplatform. Ik heb gewerkt op bètaversie 2 van Windows Vista met de CTP van WinFX (zo heette .NET Framework 3.0 toen nog). Het ontwikkelen van het WPF-gedeelte ging lekker, als je enkele bètahobbels voor lief wil nemen. Maar bij het ontwikkelen van de service liep ik tegen zo veel, voornamelijk security-gerelateerde problemen aan, dat ik daarmee verder ben gegaan onder Windows XP. Op zich geen ramp. Het resultaat kan gewoon op Windows Vista worden geïnstalleerd en ik zie de vrije keuze van ontwikkelplatform als één van de vele voordelen van het loskoppelen van het framework van het OS.

Aan de slag

Ik ga er van uit dat je al eerder applicaties hebt gemaakt voor .NET. Om nu ook voor .NET Framework 3.0 te kunnen ontwikkelen, heb je het volgende nodig:

- Visual Studio 2005
- De WinFX run-time componenten
- De WinFX SDK
- Visual Studio additions 'Orcas'

De laatste drie zijn, compleet met instructies, te downloaden via www.microsoft.com/downloads. Dankzij de VS-additions kunnen we nu vanuit Visual Studio 2005 een .NET Framework 3.0-applicatie maken. Onze applicatie wordt een tekenblok, waarmee je op een Tablet pc mooie plaatjes kan inkten. Deze inktfunctionaliteit is een geïntegreerd onderdeel van .NET Framework 3.0 geworden. Je hebt geen Windows XP Tablet pc edition of een aparte library nodig om te kunnen tekenen. Een pen tekent het handigst, maar als die niet op je pc zit, gebruik je de muis. De applicatie gaat tekeningen ook opslaan en opvragen. Deze functionaliteit wordt uitbesteed aan een door IIS gehoste service. De service wordt een algemene service die ook gebruikt kan worden om andere bestandstypen beheren. Ik maak een nieuwe solution en ga hier de projecten aan toevoegen.

De service

We beginnen met de service. Hij wordt gehost door IIS en is dus een nieuwe website. In de lijst van website-templates staat de WinFX-service als nieuw type.
Deze template maakt drie bestanden aan. De eerste, *Service.svc*, is te vergelijken met het asmx-bestand van een ASP.NET-webservice.

```
<%  @ServiceHost Language=""C""# Debug="true"
  Service="DataStoreService" CodeBehind="~/App_Code/
Service.cs" %>
```

Hij koppelt een request die binnenkomt via de webserver aan het contract en de implementatie van de service. Beide zijn te vinden in de codebehind, een bestand in app_code, één van de voorgedefinieerde mappen in ASP.NET 2.0. Dit gegenereerde service.cs-bestand bevat een voorbeeldimplementatie van een service. Je kunt het vergelijken met de 'Hello World' webmethod van een ASP.NET-webservice. De daarin voorkomende types gaan we stuk voor stuk aanpassen aan onze eigen behoeftes. Dit moet met enige zorg gebeuren. Als je een naam op één plaats verandert, heeft dat ook gevolgen op andere plaatsen. Behalve in de net genoemde Service.svc komt een aantal van de type-namen ook weer terug in de web.config. Over dat laatste straks meer.
Een service scheidt contract van implementatie. Naar buiten toe toont een service alleen de contracten. Hoe deze geïmplementeerd zijn, is van geen enkel belang voor de consument van de service. Er zijn twee soorten contracten. Het servicecontract is een opsomming van de functionaliteit die de service te bieden heeft, en het datacontract beschrijft de data die de service en de consument met elkaar uitwisselen. Het servicecontract heeft het datacontract nodig om de parameters van zijn operaties te typeren. De contracten zijn in de code gemarkeerd met attributen, in de gegenereerde code zie je een *ServiceContract* en een *DataContract*. De service biedt een opslag voor bestanden. Een bestand heeft een naam en een inhoud, de naam is een string en in een array van bytes kun je alles kwijt wat je maar wil. Ons datacontract zal er uitzien zoals in codevoorbeeld 1. De class wordt gepubliceerd als contract door het *DataContract*-attribuut. De publieke properties Name en Data worden door het *DataMember*-attribuut als leden van dat contract gepubliceerd. We hebben nu vastgelegd hoe de data er uit zien die de consumer met de service gaat uitwisselen. De volgende stap is het beschrijven wat de service doet. De service heeft drie operaties: het teruggeven van een lijst van bestanden met een bepaalde extensie, het ontvangen van een bestand, en het leveren van een bestand. Het servicecontract ziet er uit zoals in codevoorbeeld 2.
De interface is gemarkeerd met het *ServiceContract*-attribuut, de members van de interface zijn gemarkeerd met het *OperationContract*-attribuut. Zij beschrijven de operaties die de service te bieden heeft. In de beschrijving van de operaties wordt het zojuist gedefinieerde datacontract gebruikt. Nu we de service hebben beschreven, rest ons een implementatie. De gegenereerde code bevat ook een klasse die het servicecontract

```
[DataContract]
public class DataStoreContract
{
  string name;
  byte[] data;

  [DataMember]
  public string Name
  {
    get { return name;}
    set { name = value;}
  }
  [DataMember]
  public byte[] Data
  {
    get { return data;}
    set { data = value;}
  }
}
```

Codevoorbeeld 1.

```
[ServiceContract()]
public interface IDataStoreService
{
  [OperationContract]
  string[] GetFileList(string extension);
  [OperationContract]
  DataStoreContract GetFile(string fileName);
  [OperationContract]
  bool StoreFile(DataStoreContract dataStoreContractValue);
}
```

Codevoorbeeld 2.

```
public class DataStoreService : IDataStoreService
{
  private string dataStore
  { .. }

  public string[] GetFileList(string extension)
  { .. }

  public DataStoreContract GetFile(string filename)
  { ..}

  public bool StoreFile(DataStoreContract dataStoreContractValue)
  { .. }
}
```

Codevoorbeeld 3.

```
public DataStoreContract GetFile(string filename)
{
  DataStoreContract result = new DataStoreContract();
  byte[] data = {};
  FileStream fs = null;
  try
  {
    string fullFileName = dataStore + filename;
    FileInfo fi = new FileInfo(fullFileName);
    if (fi.Exists)
    {
      result.Data = new byte[fi.Length];
      fs = new FileStream(fullFileName, FileMode.Open,FileAccess.Read);
      fs.Read(result.Data, 0, (int)fi.Length);
    }
  }
  finally
  {
    if (fs != null)
      fs.Close();
  }
  return result;
}

public bool StoreFile(DataStoreContract dataStoreContractValue)
{
  string fileName = dataStore + dataStoreContractValue.Name;
  FileStream fs = null;
  bool succes = false;
  try
  {
    fs = new FileStream(fileName, FileMode.Create, FileAccess.Write);
    fs.Write(dataStoreContractValue.Data, 0,
      dataStoreContractValue. Data.Length);
    succes = true;
  }
  finally
  {
    if (fs != null)
      fs.Close();
  }
  return succes;
}

public string[] GetFileList(string extension)
{
  DirectoryInfo di = new DirectoryInfo(dataStore);
  ArrayList names = new ArrayList();
  if (di.Exists)
  {
    FileInfo[] fi = di.GetFiles("*" + extension);
    foreach (FileInfo info in fi)
    {
      names.Add(info.Name);
    }
  }
  string[] result = names.ToArray(typeof (string)) as string[];
  return result;
}
```

Codevoorbeeld 4.

implementeert. De class *DataStoreService* implementeert de interface IDataStoreService; zie codevoorbeeld 3.
De code voor de members is rechttoe rechtaan. De array van bytes in het datacontract wordt naar schijf geschreven en weer teruggelezen. De name in het datacontract wordt de bestandsnaam; zie codevoorbeeld 4.
In het wegschrijven zit een kleine valkuil. Deze code zal uitgevoerd worden door het ASP.NET-workerproces en dat heeft default geen rechten om te mogen schrijven. Om dit netjes en ook veilig op orde te brengen, moet je op de webserver een eigen map aanmaken en het ASP.NET-machineaccount daar schrijfrechten geven. Waar die map staat is in de code ingepakt in de helper dataStore; zie codevoorbeeld 5.
Deze leest de configuratie uit de web.config. De WinFX-template had de web.config al aangemaakt. Hierin worden de zogeheten endpoints van de service geconfigureerd; zie codevoorbeeld 6. Een endpoint is hoe de buitenwereld de service ziet en er contact mee maakt.
Het endpoint heeft een binding en is in deze service geconfigureerd als *wsHttpBinding*. Het is een webservice die door de consument over http kan worden benaderd. Het endpoint geeft toegang tot het contract *IdataStoreService*. Als je dit vergelijkt met een asmx-webservice, dan valt je als eerste op dat de manier waarop de service benaderbaar is naar de configuratie is verhuisd. Een asmx-webservice benader je altijd over http. Het is hier mogelijk om dat in de configuratie te veranderen in tcp of in iets als een messagequeue. Ook kun je in de configuratie het contract en zo ook de implementatie van de service veranderen. De service zou voor deze functionaliteit ook als klassieke asmx-webservice gebouwd kunnen worden, maar in deze vorm biedt die een startpunt voor een uitgebreid scala aan mogelijkheden. Voor nu is deze vorm voldoende om er mee verder te kunnen in dit artikel.

De Windows-client

Voor de client gaan we een onderdeel van .NET Framework 3.0 gebruiken dat tot nu toe door het leven ging onder de weinig poëtische naam Windows Presentation Foundation en daarvoor Avalon heette. Eén van de zegeningen van .NET Framework 3.0 is naar mijn smaak dat het een einde maakt aan de eindeloze naamsoep, waarbij menige documentatie onleesbaar was geworden, omdat elke keer dat een naam gebruikt werd ook alle voorgaande namen genoemd werden. Nu is er één grote bibliotheek waarin je alles zou moeten kunnen vinden dat je nodig hebt en er is maar één korte en duidelijke naam. Leve .NET. Aan de solution voeg ik een nieuw project toe: een WinFX Windows Application.
Dit levert een Windows-applicatie op die in plaats van een WinForms form1 een WPF Window1.xaml form heeft. Dat ziet er op het eerste gezicht in de designer bijna net zo uit als een klassiek winform en ook hier kun je er in de designer vanuit de toolbox textboxen, buttons en allerlei andere controls op slepen. Maar dat is niet de manier om een WPF-applicatie te ontwikkelen. Een WPF-applicatie is anders. Een klassiek Windows-form heeft een gefixeerde lay-out. Met een beetje kunst- en vliegwerk zijn controls nog op te rekken door een form te vergroten of te verkleinen, maar in ieder geval liggen de onderlinge positie van de controls en de gebruikte lettertypes en groottes vast. Eenieder die wel eens geprobeerd heeft een strakke en flexibele lay-out te maken van een Windows-form weet hoe vreselijk lastig dat is.
Webapplicaties hebben deze beperkingen niet. Een vloeiende lay-out, waarin de ordening en grootte van de controls zich aanpassen aan de beschikbare ruimte is iets dat elke webbrowser kan. Met css-stylesheets kan bovendien de opmaak van een pagina in de configuratie worden bepaald. Een WPF-applicatie volgt ditzelfde model, in de XAML-tab van het form bepaal je de lay-out in een mark-up-taal die heel sterk aan HTML en ASP.NET doet denken. Als je ervaring hebt met het maken van webapplicaties zul jij je onmiddellijk thuis voelen. De tekenblokapplicatie ga ik dan ook helemaal vanuit XAML opbouwen.
In grote lijnen moet de tekenapplicatie er als volgt uit gaan zien: links een kolom met knoppen voor de instellingen van de pen, in het midden het tekenvel zelf en rechts een galerij van plaatjes om uit te kunnen kopiëren. En bovenin een toolbar voor het algemene menu. De XAML van het nieuwe form bestaat uit een paar Window-tags met als attributen de default-grootte van het window en een aantal namespaces waarin de XAML-elementen worden gedefinieerd. Tussen de *Windows*-tags staan een paar *Grid*-tags. Onze applicatie is te beschrijven als een grid van drie kolommen bij twee rijen. Dat kun je rechtstreeks in XAML inkloppen; zie codevoorbeeld 7.
Er worden drie kolommen gedefinieerd, de linker is 100 pixels breed, de andere twee kolommen delen de verder beschikbaar ruimte. Er worden twee rijen gedefinieerd, de bovenste is 30 pixels hoog, de onderste krijgt alle ruimte die er overblijft. Nu kun je alle controls die op het form moeten komen in deze zelfde syntax inkloppen. Het tekenvel wordt een InkCanvas, omdat

```
private string dataStore
{
   get
   {
      return System.Web.Configuration.WebConfigurationManager.AppSettings
         ["datastore"];
   }
}
```

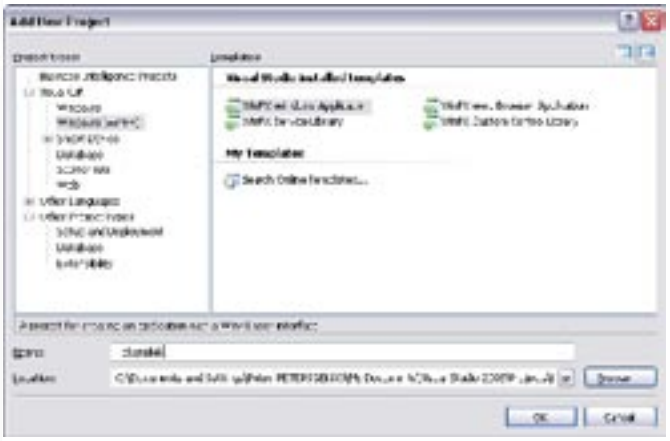
Codevoorbeeld 5.

```
<system.serviceModel>
   <services>
      <service name="DataStoreService">
         <endpoint contract="IDataStoreService" binding="wsHttpBinding"/>
      </service>
   </services>
   <behaviors>
</system.serviceModel>
```

Codevoorbeeld 6.

```
<Window x:Class="TekenBlok.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="TekenBlok" Height="596" Width="710"
   >
   <Grid>
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="100"/>
         <ColumnDefinition/>
         <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
         <RowDefinition Height="30"/>
         <RowDefinition/>
      </Grid.RowDefinitions>
   </Grid>
</Window>
```

Codevoorbeeld 7.



Afbeelding 2. Het toevoegen van een WinFX Windows-applicatie

het de Tablet pc-functionaliteit biedt. In de attributen geef je op dat het canvas in rij nummer 1 en kolom nummer 1 van het grid terecht moet komen.

```
<InkCanvas Grid.Row="1" Grid.Column="1" Name="papier"
Background="White
Smoke"></InkCanvas>
```

Van een control geef je alleen die attributen op die je echt nodig hebt. Uiteindelijk wordt vanuit al deze XAML binaire code gegenereerd en krijgen de niet-gespecificeerde attributen een default waarde. Maar daar hoef je je mark-up niet mee lastig te vallen. Als je met de drag- en dropdesigner gaat werken, zal je zien dat je mark-up vol komt te staan met allerlei attributen waar je helemaal niet om hebt gevraagd, zoals attributen die je controls een absolute positie en grootte geven. Iets wat je juist wilt vermijden.

De service gebruiken

In de rechterkolom van het grid wil ik een paar controls om plaatjes naar en van de service te sturen. Eerst verschijnen er achtereenvolgens een button om een lijst op te vragen, een listbox om de lijst te tonen en een button om het geselecteerde item daadwerkelijk op te halen. Dan komt een button om het huidige plaatje op te slaan met daarbij een textbox om de naam in te voeren. Al deze controls wil ik in één verticale lijst hebben. De ideale XAML-container daarvoor is een stackpanel; zie codevoorbeeld 8.
Ook hier geven de Grid.Row- en de Grid.Column-attributen aan welke gridkolom de container van het panel wordt. Het Orientation-attribuut geeft aan dat de controls verticaal geordend moeten worden. Opnieuw heb ik de mark-up tot het essentiële beperkt. Alleen de controls die vanuit de code benaderd gaan worden, krijgen een expliciete naam. Ook platte tekst komt tussen tags, hier staat een leeg paar TextBlock-tags als scheidingsruimte.
De buttons gaan code uitvoeren. In het click-attribuut koppel ik het click-event aan een eventhandler met de waarde van het attribuut als naam. De code-behind-file, de derde view op het form, bevat de eventhandlers. XAML-eventhandlers hebben een iets andere signature dan de eventhandlers van een Winform. Beide hebben als eerste parameter de sender, dat is de control die het event heeft afgevuurd. Later zullen we daar nog wat mee gaan doen. De tweede parameter van een XAML-event is van het type RoutedEventArgs. In een Windows-form voert een eventhandler wat code uit en daarna is het klaar. In een XAML-applicatie kan een event bubbelen naar andere controls in de hiërarchie van controls. Wie in ASP.NET met command bubbling heeft gewerkt, zal hier iets herkennen. Ik heb hier helaas geen ruimte om verder in te gaan op het routeren van events, maar wil het even genoemd hebben. Het is een punt waar je later op verder kunt bouwen en veel moois mee kan doen. De eventhandlers voor de buttons gaan gebruikmaken van de service. Voor we de code kunnen schrijven, hebben we eerst een reference naar de service nodig. In het project staat een map Service References, met een rechtermuisklik kun je er een reference aan toevoegen.
Vervolgens verschijnt een dialoog die om een URI naar een service vraagt. De dialoog biedt aan naar webservices te zoeken. Het probleem is dat er alleen naar klassieke ASMX-services wordt gezocht, de WCF-service van onze solution wordt niet gezien. We voeren daarom de URI met de hand in. De URI is die van de svc uit het serviceproject, in dit geval dus http://localhost/WinFx/FileStoreService/Service.svc?wsdl. De wsdl-parameter geeft aan dat we de WebServiceDescriptionLanguage nodig hebben om een proxy te genereren. Na deze actie heeft de applicatie een proxy om met de service te communiceren. De hele configuratie van de service wordt door de wizard opgenomen in de app.config; zie codevoorbeeld 9. Het abc van een service komt hier bij elkaar.

```
<StackPanel Name="Galery" Grid.Row="1"
         Grid.Column="2" Orientation="Vertical">
  <Button HorizontalAlignment="Center"
        Click="Beschikbaar">Lees lijst</Button>
  <ListBox Name="plaatjesBeschikbaar">
  </ListBox>
  <Button HorizontalAlignment="Center"
        Click="Lees">Lees plaatje</Button>
  <TextBlock></TextBlock>
  <Button Click="Bewaar"
        HorizontalAlignment="Center" >Bewaar</Button>
  <TextBox Name="bewaarNaam">Naam hier</TextBox>
</StackPanel>
```

Codevoorbeeld 8.

```
<client>
  <endpoint address="http://localhost/WinFx/FileStoreService/Service.svc"
  binding="wsHttpBinding"
  bindingConfiguration="WSHttpBinding_IDataStoreService"
  contract="TekenBlok.DataStoreService.IDataStoreService"
  name="WSHttpBinding_IDataStoreService" />
</client>
```

Codevoorbeeld 9.

Een service heeft een adres (te vinden op localhost), een binding (de communicatie loopt over het http-protocol) en een contract (de IDataStoreService) biedt de functionaliteit nodig. Door een nieuw proxy-object aan te maken, kunnen messages met de service worden uitgewisseld. Het aanmaken van de proxy doe ik in een helper-property. Het opvragen van de beschikbare plaatjes ziet eruit zoals in codevoorbeeld 10.
De beschikbaar methode is de eventhandler voor de button, let op het XAML-specifieke type van de tweede parameter. Het ophalen van een plaatje van de service gaat vergelijkbaar; zie codevoorbeeld 11.
De code maakt on the fly een nieuw InkCanvas aan, de binnenkomende array van bytes wordt door een memorystream naar inkt omgezet en op dit canvas getekend. Vervolgens wordt het canvas on the fly toegevoegd aan de children van het Galery-stackpanel. Je ziet hier dat de inhoud van de panel XAML-tags in je C#-code een volwaardig programmeerbare container is, waarvan je de tree van child-controls kunt uitbouwen.

Commands

Tot nu toe hebben we code aan een button gekoppeld door een click-attribuut in te vullen en een eventhandler te schrijven. Met zogeheten commands biedt WPF een nog krachtiger en efficiëntere manier om code aan een event te koppelen. Mensen met een Delphi-achtergrond zullen hier de Delphi-actionlist herkennen; tot nu toe de meeste gemiste Delphi-component in .NET. Een command is een actie (een stukje code) waar de code

Afbeelding 3. Het toevoegen van een reference

zelf losgekoppeld is van de gebeurtenis die de code initieert. Heel duidelijke actions zijn de bekende clipboard-operaties knip, kopieer en plak. Deze kunnen worden opgestart vanuit een button, een menukeuze of een specifieke toetsaanslag zoals ctrl-insert. Een action heeft ook een context, een kopieeractie heeft geen zin als er niets geselecteerd is, een plakactie heeft geen zin als er niets op het clipboard staat. Om dat naar de gebruiker duidelijk te maken zouden knoppen en menu-items in een dergelijk geval ook niet beschikbaar moeten zijn. Het .NET Framework 3.0 bevat een hele trits aan standaard commands, waaronder de clipboard-operaties. Deze ga ik gebruiken in een toolbar met een knip, een plak- en een kopieerknop, zie codevoorbeeld 12.

In het *Command*-attribuut ken ik de commands toe aan de buttons. Het mooie is nu dat de commands zelf hun targets weten te vinden. Zowel in de textboxes als op het inkcanvas kunnen we nu knippen, kopiëren en plakken en de beschikbaarheid van de knoppen wordt automatisch geregeld als er iets te knippen

```
private DataStoreServiceProxy storeService
{
   get
   {
      return new DataStoreServiceProxy();
   }
}

void Beschikbaar(object sender, RoutedEventArgs e)
{
   plaatjesBeschikbaar.Items.Clear();
   System.Collections.ObjectModel.ObservableCollection<string>
      fileNames = storeService.GetFileList(inkExt);
   foreach (string fileName in fileNames)
   {
      plaatjesBeschikbaar.Items.Add(fileName);
   }
}
```

Codevoorbeeld 10. Het opvragen van beschikbare plaatjes

```
void Lees(object sender, RoutedEventArgs e)
{
   if (plaatjesBeschikbaar.SelectedItems.Count > 0)
   {
      string fileName = plaatjesBeschikbaar.SelectedItems[0].ToString();
      DataStoreService.DataStoreContract data =
         storeService. GetFile(fileName);
      if (data.DataProperty != null)
      {
         System.IO.MemoryStream ms = new MemoryStream(data.DataProperty);
         StrokeCollection strokes = new StrokeCollection(ms);
         InkCanvas plaatje = new InkCanvas();
         plaatje.Strokes.Add(strokes);
         plaatje.EditingMode = InkCanvasEditingMode.Select;
         Galery.Children.Add(plaatje);
      }
   }
}
```

Codevoorbeeld 11. Het ophalen van een plaatje van de service

```
<ToolBar Grid.ColumnSpan="3" Grid.Row="0">
   <Button Command="ApplicationCommands.Cut">Knip</Button>
   <Button Command="ApplicationCommands.Copy">Kopieer</Button>
   <Button Command="ApplicationCommands.Paste">Plak</Button>
</ToolBar>
```

Codevoorbeeld 12.

```
public static RoutedUICommand PenDrawCommand = new RoutedUICommand();
public static RoutedUICommand PenEraseCommand = new RoutedUICommand();
public static RoutedUICommand PenSelectCommand =
new RoutedUICommand();
```

Codevoorbeeld 13.

```
private void PenDrawExecuted(object target, ExecutedRoutedEventArgs e)
{
   papier.EditingMode = InkCanvasEditingMode.Ink;
}

private void PenDrawCanExecute(object sender, CanExecuteRoutedEventArgs e)
{
   e.CanExecute = (papier != null) &&
       (papier.EditingMode != InkCanvasEditingMode.Ink);
}
```

Codevoorbeeld 14.

```
<Window x:Class="TekenBlok.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:custom="clr-namespace:TekenBlok"
   Title="TekenBlok" Height="596" Width="710" >
   <Window.CommandBindings>
   <CommandBinding Command="{x:Static custom:Window1.PenDrawCommand}"
             Executed="PenDrawExecuted"
             CanExecute="PenDrawCanExecute"/>
   <CommandBinding Command="{x:Static custom:Window1.PenEraseCommand}"
             Executed="PenEraseExecuted"
             CanExecute="PenEraseCanExecute"/>
   <CommandBinding Command="{x:Static custom:Window1.PenSelectCommand}"
             Executed="PenSelectExecuted"
             CanExecute="PenSelectCanExecute"/>
   </Window.CommandBindings>
...
```

Codevoorbeeld 15.

```
<Button Command="{x:Static custom:Window1.PenDrawCommand}">Pen</Button>
<Button Command="{x:Static custom:Window1.PenEraseCommand}">Gum</Button>
<Button Command="{x:Static custom:Window1.PenSelec
tCommand}">Selecteren</Button>
```

Codevoorbeeld 16.

```
<StackPanel Grid.Row="1" Grid.Column="0"
     Orientation="Vertical" Margin="5">
   <TextBlock FontSize="12">Pen</TextBlock>
   <Button Command="{x:Static custom:Window1.PenDrawCommand}">Pen</Button>
   <Button Command="{x:Static custom:Window1.PenEraseCommand}">Gum</Button>
   <Button Command="{x:Static custom:Window1.PenSelectCommand}">Selecteren
</Button>
   <TextBlock></TextBlock>
   <TextBlock>Pen dikte</TextBlock>
   <Slider Name="Weight" Minimum="1" Maximum="30"
         ValueChanged="penSettingChanged"></Slider>
   <TextBlock>Rood</TextBlock>
   <Slider Name="Red" Maximum="255"
         ValueChanged="penSettingChanged"></Slider>
   <TextBlock>Groen</TextBlock>
   <Slider Name="Green" Maximum="255"
         ValueChanged="penSettingChanged"> </Slider>
   <TextBlock>Blauw</TextBlock>
   <Slider Name="Blue" Maximum="255"
         ValueChanged="penSettingChanged"></Slider>
   <TextBlock></TextBlock>
   <InkCanvas Name="StrokePreview" EditingMode="None"></InkCanvas>
</StackPanel>
```

Codevoorbeeld 17.

```
private void penSettingChanged(object sender, RoutedEventArgs e)
{
   if (papier != null)
   {
      Slider mySlider = sender as Slider;
      if (mySlider != null)
      {
         byte sliderPos = (byte)mySlider.Value;
         Color nu = papier.DefaultDrawingAttributes.Color;
         switch (mySlider.Name)
         {
            case "Red": nu.R = sliderPos;
               break;
            case "Green": nu.G = sliderPos;
               break;
            case "Blue": nu.B = sliderPos;
               break;
            case "Weight": papier.DefaultDrawingAttributes.Width = sliderPos;
               break;
         }
         papier.DefaultDrawingAttributes.Color = nu;
      }
      StrokePreview.Strokes.Clear();
      StylusPointCollection spts = new StylusPointCollection(2);
      spts.Add(new StylusPoint(0, 0));
      spts.Add(new StylusPoint(100, 100));
      Stroke str = new Stroke(spts, papier.DefaultDrawingAttributes);
      StrokePreview.Strokes.Add(str);
   }
}
```

Codevoorbeeld 18.

Afbeelding 4.

of plakken valt. Om een beter idee te krijgen hoe commands werken, gaan we er zelf een paar maken. Met de pen kun je tekenen, gummen of selecteren. Om het tekencanvas in één van deze drie states te brengen, declareer ik voor elk een command in de codebehind; zie codevoorbeeld 13.
Een command wordt gedeclareerd als een public static member van de form. Deze commands interacteren met de UI en zijn daarom van het type *RoutedUICommand*. In de intialisatie van de form worden ze aangemaakt. Een command heeft twee methoden nodig om zijn werk te kunnen doen. Eén methode voert een actie uit en een logische functie bepaalt of het zinvol is de actie uit te voeren. Neem het tekenen van de pen. De actie zet de editingmode van het inkcanvas op Ink, de logische functie geeft aan of de edit-mode van dat canvas al op Ink staat. In code ziet dat er uit zoals in codevoorbeeld 14.
Voor het PenEraseCommand en het PenSelectCommand gebruiken we volstrekt vergelijkbare methodes. Ze zijn te vinden in de download die bij dit artikel hoort. In de XAML van het form worden deze acties opgenomen in de CommandBindings-tags. Deze sectie komt als eerste, nog voor het grid; zie codevoorbeeld 15.
Let op de extra Xmlns:custom-namespace die aan het form moet worden toegevoegd. De assembly TekenBlok, onze applicatie, is nodig om de XAML-referenties van de custom commands op te kunnen lossen. Deze extra namespace heeft één nadeel: de forms-designer heeft alle gerefereerde namespaces nodig, de namespace kan de custom assembly niet vinden en zal het verder vertikken. Op zich vind ik dit geen ramp maar, hoop wel dat dit in de volgende versie is opgelost. De applicatie build prima, je moet hem nu alleen runnen om te zien hoe ze er in de praktijk uit ziet. Nu de custom commands beschikbaar zijn, kunnen we ze gebruiken; zie codevoorbeeld 16.
De syntax voor de waarde van het command-attribuut ziet er een beetje cryptisch uit. Je geeft aan dat er een statisch command wordt toegekend dat te vinden is als member van de class Window1. Maar het resultaat is dat je met de buttons de penstatus kunt aansturen en dat de status van de buttons zich weer richt naar de status van de pen. Met commands kun je ontzettend veel doen en er valt nog veel meer over te vertellen. Alleen al de rijke hoeveelheid beschikbare commands in het framework is overdonderend. Ik hoop je hier een zetje in de goede richting te hebben gegeven om er meer mee te gaan doen.

Een paar toetsers en bellen

Als het goed is heb je zo langzamerhand een idee gekregen hoe je met XAML in korte duidelijke statements een rijke UI kunt maken. Als afsluiting een paar toeters en bellen om wat meer met de pen te doen. De dikte en kleur waarmee de pen tekent, kun je instellen. De applicatie doet dit met een viertal sliders. Als visuele feedback naar de gebruiker is bij elke wijziging in een preview een stroke met deze settings te zien. De XAML voor het panel vind je in codevoorbeeld 17.
Na de drie penknoppen komen de sliders. Het minimum en maximum van de pen-property die ze instellen wordt in de attributen gezet. Alle sliders gebruiken dezelfde eventhandler penSettingChanged. De InkCanvas StrokePreview laat het resultaat zien, de gebruiker kan er niet op tekenen, maar vanuit de code gaat dat wel. De penSettingChanged-eventhandler stelt de properties in. Hij gebruikt de sender-parameter om te zien welk control het event heeft afgevuurd; zie codevoorbeeld 18.
Deze eventhandler kan vaker afgaan dan je denkt. Het is daarom belangrijk eerst te controleren of het inktcanvas *papier* al is geïnitialiseerd. Zo ja, dan wordt de sender van het event met **as** 'zacht' getypecast naar een slider. Lukt dat, dan lijken we goed te zitten en kan aan de hand van de naam van de sender worden bekeken welke slider het event heeft gevuurd. Vervolgens worden kleur of dikte van de pen ingesteld. Tot slot wordt vanuit twee punten een penstroke gemaakt en op het previewcanvas getekend.

Het resultaat

Om het resultaat naar een echte Windows Vista Tablet pc te kunnen uitrollen, maak ik een setup-project aan. Dit levert een kompleet msi-bestand op. Default neemt die daarin ook de hele WinFX-runtime mee. Het resultaat zal een redelijk groot bestand zijn. In de nabije toekomst mag je het .NET Framework 3.0 op elke machine verwachten en ben je van de overhead af. Na installatie heb je een leuke applicatie die in zijn app.config kan opgeven waar de service voor het beheer van de plaatjes staat.

Tot slot

In dit verhaal zijn we in sneltreinvaart door een gedeelte van het .NET Framework 3.0 gelopen. We hebben nergens echt heel diep op in kunnen gaan, maar ik hoop dat het jullie aanknopingspunten heeft gegeven om ook met .NET Framework 3.0 aan de slag te gaan en er mooie Windows Vista-applicaties mee te bouwen.

Peter van Ooijen is werkzaam als zelfstandig .NET-consultant en ontwikkelaar bij Gekko Software (www.Gekko-Software.nl). Naast de dagelijkse ASP.NET-kost is de Tablet pc een geliefd troeteldier. Ook op zijn weblog http://codebetter.com/blogs/peter.van.ooijen komt de Tablet pc geregeld voorbij. Voor vragen en opmerkingen is hij te bereiken via Peter.van.Ooijen@Gekko-Software.nl

Referenties:

http://msdn.microsoft.com/winfx/
http://msdn.microsoft.com/tabletpc
www.devx.com/tabletpc
www.netfx3.com/