Ewart Nijburg en Patrick Vorgers zijn beiden technisch architect

Integratie van Visual Studio tools in de Windows Shell

OWNER-DRAWN CONTEXTMENU SHELL-EXTENSIES IN C#

Visual Studio 2005 biedt de ontwikkelaar naast de uitgebreide IDE ook nog een hele batterij aan tools aan. Tools zoals installutil.exe en regasm.exe zijn onmisbaar bij het installeren van windows-services en COM-objecten. Ze zijn echter niet gemakkelijk te benaderen en worden via de command prompt opgestart. Als gebruiker van deze tools zou het prettig zijn wanneer deze tools met een druk op de knop op het juiste moment beschikbaar zouden zijn. Hoe kunnen Windows Shell-extensies ons helpen bij het eenvoudig beschikbaar stellen van deze tools?

Als .NET-ontwikkelaar heb je buiten de Visual Studio IDE om meestal nog verschillende tools nodig om je geschreven software goed te kunnen laten werken; of dat nu tools zijn die door Microsoft standaard worden meegeleverd met Visual Studio of die zelf zijn ontwikkeld. Ze hebben in ieder geval met elkaar gemeen dat je ze op een bepaald moment nodig hebt tijdens het ontwikkelproces. Ze helpen je bijvoorbeeld met de deployment van de software of genereren in sommige gevallen zelfs stukken code voor je. Veel van deze tools zijn weggestopt in een of andere directory waar ze moeilijk zijn terug te vinden. In Visual Studio heb je wel de mogelijkheid om in de IDE in het *Tools*-menu externe tools toe te voegen (*External Tools* in het *Tools*-menu), maar dan gaat men er van uit dat je altijd in de Visual Studio-ontwikkelomgeving de externe tools wilt aanroepen. Nu is dit meestal niet het geval en wordt er vanuit de Windows Explorer gewerkt. Het enige wat dan rest is het opstarten van een command prompt en naar de directory te gaan waar de externe tool nodig is en daar de tool op te starten. We gaan er dan voor het gemak maar van uit dat de tool in het pad staat, want anders moet je ook nog het volledige pad opgeven. Al met al is dit erg omslachtig. Zou het niet veel handig zijn om gewoon door middel van het contextmenu de gewenste tool te kunnen opstarten? Om dit te realiseren moeten we gebruikmaken van een contextmenu Shell-extensie.

Wat zijn Shell-extensies?

Iedereen kent de kracht van de Windows Explorer bij het tonen van contextgevoelige informatie van een bestand of folder. Wanneer bijvoorbeeld een folder met mp3-bestanden wordt getoond, dan wordt standaard een aantal bestandseigenschappen van de mp3's weergegeven zoals artiest en album. Op het eerste gezicht lijkt het alsof de Windows Explorer deze functionaliteit zelf implementeert. Deze functionaliteit wordt echter gerealiseerd door gebruik te maken van de Windows Shell. De Windows Explorer visualiseert alleen maar de informatie die door de Windows Shell wordt aangeboden. De Windows Shell kan uitgebreid worden met shell-extensies. Deze shell-extensies haken in op acties die de shell uitvoert. Voordat dit gebeurt, zal de shell een extensie de mogelijkheid geven de actie aan te passen. De acties van de shell zijn onder te verdelen in acties die op bestandsniveau werken en algemene acties. Tabel 1 geeft een overzicht van een aantal bekende shell-acties waarop shell-extensies kunnen inhaken. Het tonen van een pictogram bij een bestandstype is niet iets dat door middel van een shell-extensie hoeft te worden gerealiseerd. Je krijgt dit het eenvoudigst voor elkaar door de bestandsextensie door middel van een registrykey in de registry te registeren[1]. Het registeren van bijvoorbeeld een bestandseigenschappagina of het gebruik van owner-drawn contextmenu's kan alleen maar door gebruik te maken van shell-extensies.

Contextmenu shell-extensies in C#

De werking van een contextmenu vanuit de user-interface is zeer eenvoudig. Het realiseren ervan in C# is dat allerminst. Met de introductie van contextmenu's in Windows heeft Microsoft een interface IContextMenu geïntroduceerd. Met de introductie van nieuwere versies van Windows heeft Microsoft nieuwe interfaces geïntroduceerd, namelijk de interface IContextMenu2 en later IContextMenu3, om zodoende ondersteuning te houden voor bestaande software. De Windows Explorer in Windows maakt bijvoorbeeld gebruik van deze interfaces om contextgevoelige informatie op te vragen van een element.
Om in C# een shell-extensie te kunnen ontwikkelen, moeten de volgende problemen worden opgelost:

1. Hoe zorgen we er voor dat Windows Explorer een assembly aanroept?
2. Hoe komen we aan en gebruiken we de IContextMenu-interfaces?
3. Hoe realiseren we owner-drawn functionaliteit in een contextmenu?

Windows Explorer en assemblies

De technologie achter shell-extensies is van voor het managed code-tijdperk. Hoewel .NET een onderdeel is van het Windows-platform heeft de Windows Explorer geen weet van deze technologie. In .NET-termen wordt een shell-extensie gezien als unmanaged code. Aangezien de Windows Explorer gebruik gaat maken van de .NET-shell-extensie en daarbij pointers worden doorgegeven van unmanaged naar managed code en terug, zullen we gebruik moeten maken van marshalling. Marshalling zorgt er voor dat unmanaged en managed code door elkaar heen kunnen worden gebruikt vanuit een .NET-assembly. De Windows Explorer weet alleen hoe hij COM-objecten moet laden. Het is dus belangrijk dat onze extensie als COM-object wordt gezien. In dit soort gevallen kunnen we in .NET gebruikmaken van de .NET COM Interop-laag. Deze laag zorgt ervoor dat assemblies zich kunnen gedragen als COM-objecten. Hierdoor lijkt het voor de Windows Explorer

Actie	Soort	Omschrijving
Contextmenu	Bestand	Het opvragen van het contextmenu van een bestand is misschien wel de meest gebruikte shell-extensie. De actie geeft de shell-extensie de mogelijkheid items aan het contextmenu toe te voegen op basis van het geselecteerde bestand.
Icoon	Bestand	Deze actie wordt uitgevoerd wanneer het pictogram van een bestand wordt getoond. De actie geeft de shell-extensie de mogelijkheid om het standaardpictogram van een bestand te vervangen.
Bestandseigenschappen	Bestand	Deze actie wordt uitgevoerd als de eigenschappen van een bestand worden opgevraagd. De actie maakt het de shell-extensie mogelijk pagina's te vervangen en of toe te voegen.
Thumbnail image	Bestand	Deze actie wordt uitgevoerd wanneer de thumbnail image van een bestand nodig is. De actie geeft de shell-extensie de mogelijkheid om een thumbnail image aan te leveren.
Infotip	Bestand	Deze actie wordt uitgevoerd wanneer de gebruiker met de muis over een bestand zweeft en de pop-up-tekst wordt opgevraagd.
Zoeken	Algemeen	Deze actie wordt uitgevoerd om een zoekmachine op te starten. Het geeft de shell-extensie de mogelijkheid een aangepaste zoek-engine te implementeren
Kolom	Algemeen	Deze actie wordt bijvoorbeeld door de Windows Explorer aangeroepen om gedetailleerde gegevens van een folder te tonen. In het geval van een folder waar mp3's in staan, geeft het de shell-extensie die mp3 afhandelt, de mogelijkheid kolommen toe te voegen met de naam van de artiest en de naam van het album.

Tabel 1. Windows Shell-acties

net alsof hij te maken heeft met een standaard COM-object. Een shell-extensie in .NET bestaat daarom uit een assembly in de vorm van een COM-object. De assembly moet door de Windows Shell in de registry worden geregistreerd[2] om te kunnen worden geladen. Shell-extensies worden door de Windows Shell als plug-ins gezien. In het geval van een contextmenu shell-extensie moet deze op zijn minst de interfaces IShellExtInit en IContextMenu implementeren. De interface IShellExtInit wordt gebruikt voor het doorgeven van algemene gegevens aan de plug-in vanuit de Explorer. De interface IContextMenu daarentegen is specifiek voor de implementatie van specifieke contextmenu functionaliteit.

IContextMenu-interfaces

De IContextMenu-interface wordt aangeroepen wanneer een gebruiker op een bestand of map het contextmenu activeert. De Explorer zoekt op basis van de context (map of bestand) naar de bijbehorende contextmenu-handler in de registry. Als deze wordt gevonden laadt en initialiseert (IShellExtInit) de Explorer het bijbehorende COM-object (plug-in). Vervolgens roept de Explorer de interface IContextMenu aan om de plug-in in de gelegenheid te stellen menu-items toe te voegen aan het contextmenu. Het toevoegen van menu-items wordt gerealiseerd in de QueryContextMenu-methode die als eerste wordt aangeroepen door de Explorer. De eerste parameter van deze methode is een handle naar het contextmenu. Deze handle kan in samenhang met de overige parameters gebruikt worden om menu-items toe te voegen. De volgende methode die aangeroepen kan worden, is GetCommandString. De Explorer roept deze methode aan om contextgevoelige helpinformatie te tonen in de statusbalk als de gebruiker met de muiscursor over het menu-item beweegt. De laatste methode die aangeroepen kan worden, is de InvokeCommand-routine. De Explorer roept deze methode aan wanneer de gebruiker het menu-item selecteert. Voor de implementatie en definitie van de IContextMenu-interfaces zijn we op ons zelf aangewezen. Er bestaat geen basisklasse waar van afgeleid kan worden. Dit komt doordat de Windows Shell eigenlijk helemaal niets af weet van .NET-applicaties. Er zit niets anders op dan de definities van de benodigde interfaces zelf op te voeren; zie codevoorbeeld 1.

```
[ComImport(), InterfaceType(ComInterfaceType.InterfaceIsIUnknown),
GuidAttribute("bcfce0a0-ec17-11d0-8d10-00a0c90f2719")]
public interface IContextMenu3
{
    // IContextMenu
    [PreserveSig()]
    int QueryContextMenu (uint hMenu, uint iMenu, int idCmdFirst,
                          int idCmdLast, uint uFlags);
    [PreserveSig()]
    void InvokeCommand(IntPtr pici);
    [PreserveSig()]
    void GetCommandString(int idCmd, uint uflags, int reserved,
                          Stringuilder commandString, int cch);
    // IContextMenu2
}
```

Codevoorbeeld 1.

Niet lang na de introductie van IContextMenu kwam Microsoft met een uitbreiding op deze interface, de IContextMenu2-interface; zie codevoorbeeld 2. Deze extra interface was bedoeld om owner-drawn functionaliteit toe te voegen aan de extensie. Aan de IContextMenu2- interface ontbrak één ding, namelijk de mogelijkheid om toetsenbordaanslagen te ontvangen in de shellextensie. Hiervoor werd de IContextMenu3-interface toegevoegd. In principe is de owner-drawn functionaliteit van de laatste twee interfaces gelijk, maar veelal is het implementeren van IContextMenu2 niet voldoende. Om een volledige owner-drawn shell-extensie te maken, moeten we dus de IContextMenu3-interface implementeren.

Owner-drawn functionaliteit in contextmenu's

Een standaard contextmenu-extensie kan een plaatje van 12x12 pixels tonen aan de linkerzijde van de menutekst. Programma's zoals Winzip en WinRar maken hier gebruik van. Deze techniek heeft als beperking dat er gebruik moet worden gemaakt van de standaard lay-out en een monotone bitmap. Met behulp van de IContextMenu3-interface is het mogelijk volledige toegang te krijgen tot het tekenoppervlak van het menu-item. De interface IContextMenu3 voegt de methode HandleMenuMsg2 toe aan de vier reeds bestaande methoden van de IContextMenu en de IContextMenu2-interface. De methode HandleMenuMsg2 vervangt de HandlerMenuMsg- methode uit de IContextMenu2-interface[3]; zie codevoorbeeld 3.

Interfaces ondersteunen inheritance. Volgens interfacegebruik zou de IContextMenu3-interface afgeleid kunnen worden van IContextMenu2, die op zijn beurt weer zou kunnen worden afgeleid van IContextMenu. Jammer genoeg werkt het in .NET niet op deze manier. Het gebruik van een shell-extensie met interfaces die zijn afgeleid van andere interfaces, resulteert in een foutmelding; zie afbeelding 1. De reden voor deze foutmelding is dat COM Interop vereist

```
[ComImport(), InterfaceType(ComInterfaceType.InterfaceIsIUnknown),
GuidAttribute("000214F4-0000-0000-c000-000000000046")]
public interface IContextMenu2 : IContextMenu
{
    [PreserveSig()]
    int HandleMenuMsg(uint uMsg, uint wParam, uint lParam);
}
```

Codevoorbeeld 2.

```
[ComImport(), InterfaceType(ComInterfaceType.InterfaceIsIUnknown),
GuidAttribute("bcfce0a0-ec17-11d0-8d10-00a0c90f2719")]
public interface IContextMenu3 : IContextMenu2
{
    [PreserveSig()]
     int HandleMenuMsg2(uint uMsg, IntPtr wParam, IntPtr lParam,
     IntPtr plResult);
}
```

Codevoorbeeld 3.

```
// COM Interop requires that all members of the inherited interfaces
// are implemented as well, this means inheritance is not an option
[ComImport(), InterfaceType(ComInterfaceType.InterfaceIsIUnknown),
 GuidAttribute("bcfce0a0-ec17-11d0-8d10-00a0c90f2719")]
public interface IContextMenu3
{
    // IContextMenu
    [PreserveSig()]
     int QueryContextMenu(uint hMenu, uint iMenu, int idCmdFirst,
     int idCmdLast, uint uFlags);
     [PreserveSig()]
     void InvokeCommand(IntPtr pici);
     [PreserveSig()]
     void GetCommandString(int idCmd, uint uflags, int reserved,
     String Builder commandString, int cch);
     // IContextMenu2
     [PreserveSig()]
     int HandleMenuMsg(uint uMsg, uint wParam, uint lParam);
     // IContextMenu3
     [PreserveSig()]
     int HandleMenuMsg2(uint uMsg, IntPtr wParam, IntPtr lParam,
     IntPtr plResult);
}
```

Codevoorbeeld 4.

dat alle methoden expliciet worden geïmplementeerd op interfaceniveau. Interface-inheritance is dus geen optie voor dit soort uitbreidingen van de shell. Er zit dus niets anders op dan alle members van de interfaces in een keer te implementeren. De volledige IContextMenu3-interface ziet er dan uit als in codevoorbeeld 4.

De methode HandleMenuMsg2 van de IContextMenu3-interface wordt gebruikt om de owner-drawn functionaliteit te realiseren. Er zijn twee Windows-messages belangrijk voor owner-drawn menu-items. Dit zijn de messages WM_DRAWITEM en WM_MEASUREITEM. De uMsg-parameter van de methode HandleMenuMsg2 bevat de message-identificatie. Deze gaan we gebruiken om de gewenste messages te filteren. Beide Windows-messages bevatten een pointer naar een structuur. Hoe benader je deze structuren vanuit een managed object? Het antwoord is te vinden in de PtrToStructure .NET-methode.

```
public static void PtrToStructure (
    IntPtr ptr,
    Object structure
)
```

Deze methode geeft toegang tot een pointerstructuur vanuit managed code. In het geval van de structuren van beide Windows-messages komt het gebruik neer op:

```
DRAWITEMSTRUCT dis = (DRAWITEMSTRUCT)Marshal.PtrToStructure(lParam,
typeof(DRAWITEMSTRUCT));
MEASUREITEMSTRUCT mis = (MEASUREITEMSTRUCT)Marshal.
PtrToStructure(lParam, typeof(MEASUREITEMSTRUCT));
```

Nu we de gewenste data tot onze beschikking hebben, kunnen we verder gaan met het implementeren van de overige functionaliteit.

Afbeelding 1. Foutmelding interface-inheritance

Tekenen

In de DRAWITEMSTRUCT-structuur bevindt zich een handle naar een devicecontext (hDC). Deze handle geeft toegang tot het tekenoppervlak van het menu. In de namespace System.Drawing. Graphics is de Graphics-klasse te vinden. Deze klasse kan worden gebruik om gemakkelijk op een devicecontext te kunnen tekenen.

```
Graphics gr = Graphics.FromHdc(dis.hDC);
```

Nu we toegang hebben tot het tekenoppervlak van het menu kunnen we de owner-drawn functionaliteit implementeren. Het uitgangspunt is het tekenen van een realistisch menu-item. Een die niet afwijkt van de normale menu-items, maar wel de mogelijkheid biedt plaatjes te tekenen. Hoe ziet een normaal menu-item er uit? Het menu gebruikt vier systeemkleuren en één systeemfont. De kleuren zijn SystemColors.Menu en SystemColors.MenuText voor alle niet-geselecteerde menu-items, SystemColors.Highlight en SystemColors.HighlightText voor het geselecteerde menu-item en SystemInformation.MenuFont als het gebruikte font. In het geval het menu-item geselecteerd is, gebruiken we de laatste twee kleuren voor achtergrond en tekst en anders de eerste twee. Via een andere member van de DRAWITEMSTRUCT, de dis.itemState-variabele, kunnen we bepalen of het getekende menu-item is geselecteerd. Deze variabele kunnen we typecasten naar een DrawItemState, omdat deze enum dezelfde lay-out heeft als de Windows-structuur.

```
DrawItemState state = (DrawItemState)dis.itemState;
```

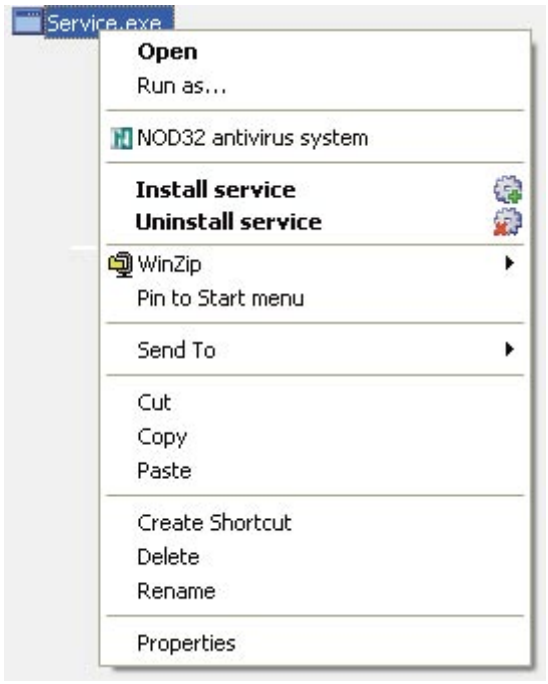
De laatste belangrijke variabele in DRAWITEMSTRUCT is dis.rcItem. Deze variabele bevat de coördinaten van het vierkant waarbinnen mag worden getekend. Het .NET Framework is bekend met het owner-drawn principe en definieert een delegate en een eventargs-klasse voor DrawItem.

Rekenen

We moeten de shell vertellen dat hij voor ons menu-item meer ruimte moet reserveren, wanneer we een ander font gebruiken of als datgene wat getekend wordt, hoger en of breder is dan het menufont. Dit is de functie van de WM_MEASUREITEM-message. In de MEASUREITEMSTRUCT kunnen we de gewenste hoogte en breedte retourneren aan de shell. Deze zorgt op zijn beurt voor het aanmaken van een menu-item met de gewenste afmetingen. De structuurvelden mis.ItemHeight en mis.itemWidth bevatten de standaardwaarden voor het gebruikte font. Zolang we ons aan de standaardgrootte houden, hoeven we niets te doen met het measure-item. Als we de hoogte en/of breedte hebben aangepast, moeten we de mis-structuur weer beschikbaar stellen aan de shell. Dit doen we door gebruik te maken van de tegenhanger van PtrToStructure, namelijk de StructureToPtr.

```
// Marshal structure back to pointer for shell to use
Marshal.StructureToPtr(mis, lParam, false);
```

Op deze manier kan de shell de door ons gezette waarden gebruiken voor het initialiseren van het menu-item.



Afbeelding 2. Voorbeeld contextmenu

Zelf een contextmenu shell-extensie implementeren

Het implementeren van een shell-extensie voor contextmenu's is eigenlijk iedere keer hetzelfde. Voor dit artikel hebben we een simpel framework gemaakt, waarmee eenvoudig contextmenu shell-extensies kunnen worden gemaakt. Het framework neemt de complexiteit van een contextmenu shell-extensie uit handen. Het stelt de owner-drawn functionaliteit beschikbaar via de DrawItemEventHandler-delegate met DrawItemEventArgs-event-argument en de MeasureItemEventHandler-delegate met MeasureItemEventArgs-event-argument.
Het framework bestaat uit een basisklasse genaamd ContextMenuShellExtension.

```
public class ContextMenuShellExtension : IShellExtInit, IContextMenu,
IContextMenu3
```

Door te overerven van deze klasse worden de volgende methoden beschikbaar voor overerving.

```
protected override bool AcceptContext(string filePath);
```

Deze methode wordt gebruikt om al dan niet de geselecteerde directory of file te accepteren en onze menu-items aan het pop-up-menu toe te voegen.

```
protected override void InitializeMenu();
```

Deze methode wordt gebruikt om de menustructuur op te bouwen. Een menu-item wordt toegevoegd door gebruik van de constructie zoals in codevoorbeeld 5 is te zien.

Het voorbeeld dat bij dit artikel zit, maakt gebruik van het beschreven framework en implementeert menu-items die het mogelijk maken een service te installeren en te deïnstalleren; zie afbeelding 2. Deze menu-items zijn bijvoorbeeld gemakkelijk uit te breiden met andere Visual Studio-tools. Hierdoor wordt het mogelijk de Visual Studio-tools vanuit de Windows Explorer eenvoudig via het contextmenu te benaderen.

Onbekend terrein?

Het maken van owner-drawn contextmenu-items is helemaal niet zo complex als je eenmaal weet hoe het moet worden gerealiseerd.

```
ContextMenuItem m = new ContextMenuItem();
m.Text = name;
// Temporary storage of data required at a later stage
m.Properties["dynamicdata"] = "Dynamic Value"
// Event handler for menu item click, declared by user
m.OnExecuteItem += ExecuteCommandMenuItem;
// Event handler for menu item draw, declared by user
m.OnDrawItem += DrawMenuItem;
// The user is going to draw the menu item
m.Owner-draw = true;
// Add the item to the internal menu structure
Items.Add(m);
```

Codevoorbeeld 5.

Het feit dat je ze nauwelijks tegenkomt, geeft wel aan dat het nog steeds onbekend terrein is. De interface IContextMenu3 biedt een krachtige manier om de shell en dus ook de contextmenu's van uitgebreide grafische elementen te voorzien. Wees echter voorzichtig met het gebruik van owner-drawn functionaliteit in contextmenu's. Gebruikers zijn nu eenmaal gewend aan uitgelijnde menu's en gelijksoortige fonts.

Ewart Nijburg is, vanuit zijn eigen onderneming, als technisch architect werkzaam bij diverse ondernemingen (www.troolean.nl).

Patrick Vorgers is als technisch architect werkzaam bij de Management en Consultancy- afdeling van Ordina Software Integration & Development (www.ordina.nl). Hun specialisaties zijn softwarearchitecturen, high availability en software-performance-engineering. Voor vragen en opmerkingen kun je ze bereiken op enijburg@troolean.nl. en patrick.vorgers@ordina.nl

Referenties

Shell Basics - http://msdn.microsoft.com/ library/en-us/shellcc/platform/shell/programmersguide/shell_basics/shell_basics.asp
Creating Shell Extension Handlers - http://msdn.microsoft.com/library/en-us/shellcc/platform/shell/programmersguide/shell_int/shell_int_extending/extensionhandlers/shell_ext.asp
Shell-extensies – http://www.theserverside.net/tt/articles/showarticle.tss?id=Shellextensies
Shell Basics: Extending the Shell - http://windowssdk.msdn.microsoft.com/en-us/library/ms649612.aspx
Marshalling – http://msdn.microsoft.com/net/framework/programming/interop/
IContextMenu - http://windowssdk.msdn.microsoft.com/en-us/library/ms646648.aspx
IContextMenu2 - http://windowssdk.msdn.microsoft.com/en-us/library/ms646645.aspx
IContextMenu3 - http://windowssdk.msdn.microsoft.com/en-us/library/ms646648.aspx

Noten

1 Zie de 'Shell Basics' op de MSDN site voor het registeren van bestandsextensies in de registry
2 Zie de 'Creating Shell Extension Handlers' op de MSDN-site voor het registreren van shell-extensies
3 Zie msdn.microsoft,com voor een uitgebreide beschrijving van de parameters