Debugger Visualizers

BEKIJK JE DATA TIJDENS HET DEBUGGEN MET JE EIGEN VIEW

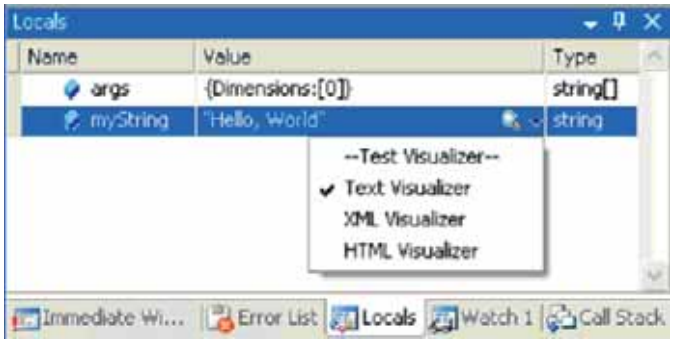
Vanaf Visual Studio 2005 kun je eenvoudig code maken om de inhoud van variabelen in de debugger met een eigen form te laten zien. Dit artikel beschrijft hoe je dergelijke visualizers gebruikt en hoe je ze maakt.

Tijdens het ontwikkelen en onderhouden van software besteden we tijd aan het opsporen van fouten. Hierbij is de debugger een essentieel onderdeel om de inhoud van de data te evalueren. De debugger van Visual Studio biedt zeer goede en geavanceerde mogelijkheden om de data van de applicatie tijdens het debuggen te bekijken en te veranderen. Toch zijn er nog genoeg situaties waarbij je een andere view op data wilt hebben dan Visual Studio al aanbiedt.
Visualizers zijn debugger-helper classes die datatypes kunnen tonen in een eigen form. Als je aan het debuggen bent en Visual Studio herkent dat er een passende visualizer aanwezig is voor een datatype, dan wordt dat met een vergrootglas en een dropdown button ( ). aangegeven in het autos, locals, watch of tooltip window van de debugger. De opengeklapte dropdown box bevat alle visualizers die je voor het betreffende datatype kunt gebruiken. Dit geldt niet alleen voor het datatype zelf, maar ook voor classes waarvan het datatype is afgeleid. Je kunt dus verschillende visualizers aan een datatype koppelen. Afbeelding 1 toont een eigen visualizer (-- Test Visualizer--), gekoppeld aan het standaard string datatype. Standaard zijn er al veel visualizers aanwezig in de Visual Studio-omgeving. Bijvoorbeeld drie visualizers voor het string datatype. Er komen echter genoeg situaties voor waarin de standaard aanwezige visualizers niet compatibel zijn voor het datatype dat je wilt bekijken. Als het een datatype uit de .NET library betreft, kun je op internet vaak wel een geschikte visualizer vinden. Zo staan er op www.CodePlex.com visualizers voor Web-development, XAML, WCF-classes en nog veel meer. Visualizers voor applicatiespecifieke datatypes moeten we echter zelf maken.

Een eigen visualizer maken

Je kunt nagenoeg voor alle soorten managed datatypes visualizers maken, ongeacht of het een eigen datatype betreft of een bestaand datatype zoals string of form. Hierbij geldt wel de beperking dat het datatype serializeerbaar moet zijn!
Een visualizer bestaat meestal uit drie delen:

- Een attribute dat de koppeling legt tussen het datatype en de visualizer.



AFBEELDING 1. KIEZEN VAN EEN VISUALIZER

```
using System.Diagnostics;

[DebuggerVisualizer(typeof(Vector2DVisualizer))]
[Serializable]
public class Vector2D
{
    ...
}
```

CODEVOORBEELD 1. DEBUGGERVISUALIZER ATTRIBUTE BOVEN DE CLASS

- Een class, afgeleid van DialogDebuggerVisualizer. Deze class fungeert als 'startpunt' bij het activeren van de visualizer.
- Een eigen form dat de data zal tonen.

Het DebuggerVisualizer attribute definieert de koppeling tussen het datatype en de visualizer class. Als parameter van de DebuggerVisualizer geef je op welke afgeleide class van de DialogDebuggerVisualizer bij het activeren van de visualizer wordt gebruikt.
Het DebuggerVisualizer attribute kunnen we op twee manieren in de code opnemen:

- bij de class die je wilt visualizen;
- in een losse file of assembly.

Het gemakkelijkst is het opnemen van het attribute boven de class waarvoor de visualizer-class geschikt is. Er bestaat dan namelijk direct duidelijkheid dat er een visualizer aanwezig is voor de betreffende class. De code van de visualizer zelf zal dan meestal wel onderdeel van de assembly moeten uitmaken. Of je neemt een referentie op naar de assembly, waarin de visualizer gedefinieerd staat. Dit laatste kan een probleem opleveren, omdat je dan ook een referentie naar de Microsoft.VisualStudio.DebuggerVisualizers assembly introduceert. Deze assembly is alleen aanwezig bij Visual Studio 2005 of hoger. Codevoorbeeld 1 koppelt de Vector2DVisualizer aan de Vector2D class door het opnemen van het DebuggerVisualizer attribute boven de class.

Als je geen afhankelijkheid van je code naar de visualizer assembly wilt introduceren, of als je een visualizer wilt maken voor een classtype waarvan de code niet beschikbaar is, moet je een losse visualizer assembly maken die zelf de koppeling legt tussen de visualizer en het classtype waar de visualizer bij hoort. Codevoorbeeld 2 laat zien hoe het DebuggerVisualizer attribute 'los' gedefinieerd wordt. Bij deze declaratie staat in de 'Target' vermeld voor welk classtype de visualizer geschikt is.

De class waarin de visualizer zelf geïmplementeerd wordt, moet worden afgeleid van de abstracte DialogDebuggerVisualizer class. Binnen deze class hoeft alleen de Show-functie geïmplementeerd te worden. De assembly waarin de visualizer code is opgenomen, moet een referentie hebben naar de Microsoft.VisualStudio.DebuggerVisualizers assembly.

```
[assembly: System.Diagnostics.DebuggerVisualizer(
        typeof(VisualizerExample.Vector2DVisualizer),
        Target = typeof(VisualizerExample.Vector2D),
        Description = "Vector 2D")]
```

CODEVOORBEELD 2. LOS GEDEFINIEERDE DEBUGGERVISUALIZER ATTRIBUTE

Let hierbij op dat er verschillende versies van deze assembly bestaan voor Visual Studio 2005 en Visual Studio 2008.
De Show-functie van de visualizer zal eerst de te tonen data ophalen en dan de form of usercontrol instantie aanmaken dat de data zal tonen. De implementatie van de Show-functie verloopt meestal zeer eenvoudig. Codevoorbeeld 3 toont visualizercode die een form aanmaakt voor de data, waarbij de data in de caption getoond wordt. Meer geavanceerde visualizers zullen een eigen form gebruiken om de data te tonen.
Hoewel het maken van een visualizer al gemakkelijk gaat, bestaat er binnen Visual Studio een template om een eigen visualizer class aan je project toe te voegen. Als je "Add new item" uitvoert voor je project, kun je uit de getoonde lijst van templates de "Debugger Visualizer" template kiezen (zie afbeelding 2).

Als resultaat van het uitvoeren van de Debugger Visualizer-template wordt een visualizer class aangemaak. Daarin staan de Show-functie en een static TestShowVisualizer-functie, die voor een deel al zijn ingevuld. Indien nodig zal ook een referentie naar de DebuggerVisualizer assembly worden toegevoegd.
De template definieert echter nog niet de koppeling tussen de visualizer en het datatype. Dit moet je nog zelf in de code opnemen (zie codevoorbeelden 1 en 2).

Veranderen van data

De visualizer loopt in het process van de debugger, terwijl het te debuggen-programma in een eigen process loopt (de zogeheten debuggee). Na het activeren krijgt de visualizer altijd een kopie van de data doorgezonden. Voor de dataoverdracht wordt de objectProvider doorgegeven aan de visualizer. Veranderingen aan de data worden niet automatisch doorgevoerd naar het process dat gedebugged wordt. Om dat toch voor elkaar te krijgen moet de visualizer expliciet de Replace of Transfer methods aanroepen op de objectProvider.

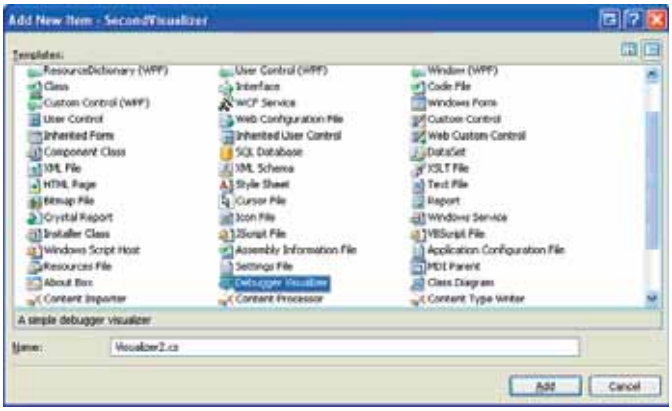
Installeren van de visualizer

De debugger moet de visualizer vinden op het moment dat de debugger wordt gestart. Als de visualizer reeds onderdeel uitmaakt van de assembly die gedebugged wordt, herkent Visual Studio hem direct. Als de visualizer in een losse assembly staat, moet de assembly nog naar de visualizer folder

```
using Microsoft.VisualStudio.DebuggerVisualizers;

public class Vector2DVisualizer : DialogDebuggerVisualizer
{
    protected override void Show(IDialogVisualizerService windowSer-
    vice, IVisualizerObjectProvider objectProvider)
    {
      // Haal de data die gevisualizeerd moet worden op.
      Vector2D vec = (Vector2D)objectProvider.GetObject();
// Als voorbeeld wordt een leeg form gebruikt waarbij de caption
// de "ToString" inhoud van de overgezonden variabele zal tonen.
      using (Form visualizerForm = new Form())
      {
        visualizerForm.Text = vec.ToString();
        windowService.ShowDialog(visualizerForm);
      }
    }
}
```

CODEVOORBEELD 3. EEN EENVOUDIGE VISUALIZER CLASS



AFBEELDING 2. DEBUGGER VISUALIZER-TEMPLATE

```
protected override void Show(IDialogVisualizerService windowSer-
vice, IVisualizerObjectProvider objectProvider)
{
    Vector2D vec = (Vector2D)objectProvider.GetObject();
    ...
    if (objectProvider.IsObjectReplaceable)
    {
      // Voorbeeld: Verander de data en stuur deze terug.
      vec.X = 999;
      objectProvider.ReplaceObject(vec);
    }
}
```

CODEVOORBEELD 4. VERANDERDE DATA TERUGZENDEN

van Visual Studio worden gekopieerd. Dit is een van de volgende folders:
My Documents\Visual Studio 2005\Visualizers
of
My Documents\Visual Studio 2008\Visualizers
of
%VSINSTALLDIR%\Common7\Packages\Debugger\Visualizers

Het kopiëren van een visualizer assembly in één van bovenstaande folders is voldoende om de visualizer te gebruiken bij de eerstvolgende keer dat de debugger wordt gestart.

Testen van de visualizer

Als je zelf een visualizer hebt gemaakt, moet je deze nog testen. Dit kan door door de visualizer te installeren en dan in de debugger te activeren om zo te zien wat er wel of niet werkt. Maar er zijn eenvoudiger manieren om te testen.
Om het testen en debuggen van visualizers te vergemakkelijken bestaat in de Microsoft.VisualStudio.DebuggerVisualizers namespace de VisualizerDevelopmentHost class. Deze class doet de Visual Studio debugger na.
Als je de visualizer met behulp van de wizard hebt aangemaakt, is er een static TestShowVisualizer-functie aangemaakt met code waarin de VisualizerDevelopmentHost wordt gebruikt om de visualizer te starten. Testen van de visualizer is dan mogelijk door de TestShowVisualizer-functie van de visualizer aan te roepen met de data die getoond moet worden. We kunnen dan de debugger gebruiken om door de code van de visualizer heen te lopen.
Houdt er bij het gebruik van de VisualizerDevelopmentHost rekening mee dat de visualizer niet in het debugger-process loopt, maar in het process dat je aan het runnen bent. Als je data met de visualizer aanpast, zal deze dus ook in het aanroepende programma veranderen.

Niet serializeerbare types

De te visualiseren data wordt door middel van serialization van het debuggee process naar het debugger process overgebracht. Hierdoor

werkt de visualizer altijd met een kopie van de data. De VisualizerObjectSource voert het serializeren van de data uit.

Stel dat er dataoverdracht moet plaatsvinden van datatypes die niet voldoen aan de serialization-criteria. Dan kunnen we een eigen VisualizerObjectSource afgeleide class inzetten om het doel toch te bereiken. De eigen VisualizerObjectSource moet je dan opgeven bij de declaratie van het DebuggerVisualizer attribute. Als je geen specifieke VisualizerObjectSource definieert, wordt voor de dataoverdracht de standaard VisualizerObjectSource class gebruikt.

VisualizerObjectSource afgeleide class inzetten om doel toch te bereiken

Voor de dataoverdracht zijn twee functies van de VisualizerObjectSource van belang.

- *GetData*
  De GetData-functie wordt aangeroepen wanneer "GetObject" of "GetData" van de objectprovider wordt aangeroepen. Dit gebeurt dus als je data overhaalt naar de visualizer.
- *CreateReplacementObject*
  De CreateReplacementObject-functie wordt aangeroepen als je aangepaste data wilt terugzenden van debugger naar het debuggee process. De CreateReplacementObject-functie wordt via de aanroep van één van de Replace of Transfer-functies van de ObjectProvider aangeroepen.

Codevoorbeeld 5 toont een custom VisualizerObjectSource. Hierbij wordt een eigen wel-serializeerbaar datatype gebruikt om de niet-serializeerbare data over te zenden. De visualizer krijgt dit wel-serializeerbare type dan binnen wanneer deze vraagt om het te tonen object.

Beperkingen van visualizers

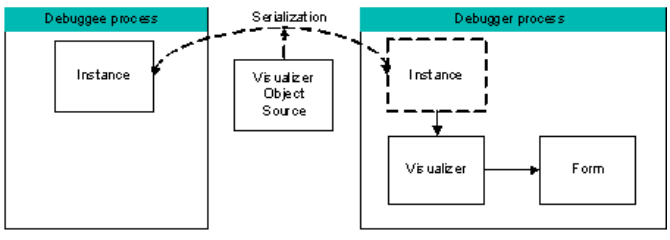
Ondanks dat je visualizers gemakkelijk kunt maken, moet je toch rekening houden met een aantal beperkingen.

- Je kunt geen visualizer maken voor de basistypes "Object" of "Array".
- Je kunt geen visualizer maken voor interfacetypes.
- Er is maar beperkte ondersteuning voor generics.

De visualizer voor generics is alleen geldig voor open types. Het gaat om generics waar geen type voor is ingevuld. De visualizer krijgt dan een object waarvan het voor de generic gebruikte type niet bekend is. Om aan de generic-data te komen, moet je dan dan reflection gebruiken.

- De visualizer-code wordt niet ge-JIT.

Daardoor is het niet raadzaam grote operaties, of operaties op grote groepen data uit te voeren, aangezien dit veel tijd kan kosten.



AFBEELDING 3. SERIALIZATION VAN DATA VAN EN NAAR DE VISUALIZER

```
[assembly: System.Diagnostics.DebuggerVisualizer(
         typeof(NotSerializableVisualizer.NSVector2DVisualizer),
         typeof(NotSerializableVisualizer.NSVector2DObjectSour-
         ce),
         Target = typeof(NSVector2D.NSVector2D),
         Description = "NS Vector 2D Visualizer")]

public class NSVector2DObjectSource : VisualizerObjectSource
{
    // Interne wel-serializeerbare class die de niet-serializeer-
    bare data
    // in zich zal opnemen.
    [Serializable]
    public class Vector2DData
    {
      public Vector2DData(NSVector2D.NSVector2D vec)
      {
        _X = vec.X;
        _Y = vec.Y;
      }
      public int _X;
      public int _Y;
    }

    public override void GetData(object target, System.IO.Stream
    outgoingData)
    {
      // Zet niet-serializeerbare structuur om in structuur die
      wel
      // serializeerbaar is.
      NSVector2D.NSVector2D vec = target as NSVector2D.NSVec-
      tor2D;
      Vector2DData serializableData = new Vector2DData(vec);
      // Laat base class de data serializeren
      base.GetData(serializableData, outgoingData);
    }
}

public class NSVector2DVisualizer : DialogDebuggerVisualizer
{
// Let op: Je ontvangt het datatype dat geserializeerd wordt.
// Dit hoeft dus niet hetzelfde datatype te zijn als het datatype
// waarvoor je de visualizer gestart hebt.
NSVector2DObjectSource.Vector2DData data = (NSVector2DObjectSour-
ce.Vector2DData)objectProvider.GetObject();
   ...
}
```

CODEVOORBEELD 5. CUSTOM SERIALIZATION

Conclusie

In dit artikel heb ik geprobeerd duidelijk te maken hoe eenvoudig het is zelf visualizers te maken. De beste manier om de kracht en eenvoud te leren kennen, is door er zelf mee aan de slag te gaan. Als je de visualizers direct ontwikkelt met de reguliere code, zul je daar gedurende de rest van de levensduur van de applicatie voordeel van hebben. Uiteraard zijn er op het web ook de nodige visualizers te krijgen voor common datatypes. .net

Referenties

http://www.ddj.com/windows/184406197
http://msdn.microsoft.com/en-us/magazine/cc163974.aspx
http://davidhayden.com/blog/dave/archive/2005/12/26/2645.aspx
http://karlshifflett.wordpress.com/mole-for-visual-studio/

Ed van de Pitte werkt als softwarearchitect en competence leader bij Atos Origin Technical Automation (www.atosorigin.com) waar hij betrokken is bij mechatronica applicaties. Voor vragen en opmerkingen is Ed bereikbaar via Ed.vandePitte@AtosOrigin.com