**Patrick Vorgers** is technisch architect bij Ordina. Ewart Nijburg is als technisch architect werkzaam bij diverse ondernemingen

# Zet Visual Studio naar je hand

## ONTWIKKELOMGEVING NAAR EIGEN SMAAK INRICHTEN MET BEHULP VAN EXTENSIES

Visual Studio 2005 biedt standaard al veel instellingen en ondersteunende tools om de ontwikkelomgeving en het ontwikkelproces naar eigen smaak in te richten. Maar elke ontwikkelaar is anders en heeft zo zijn eigen voorkeuren over hoe hij wil ontwikkelen. Hierdoor is het niet mogelijk om de perfecte omgeving voor iedereen te maken. Maar hoe kunnen we als ontwikkelaar de uitbreidbaarheid van Visual Studio gebruiken om deze te laten aansluiten op ons ontwikkelproces?

Van nature zijn we als ontwikkelaars lui aangelegd. Waarschijnlijk is dit ook een van de redenen dat we ons bezighouden met het ontwikkelen van software. Waarom zouden we iets steeds weer handmatig uitvoeren als we dat ook eenvoudig kunnen automatiseren? We hebben een hekel aan steeds hetzelfde te moeten doen. Dit geldt natuurlijk in het bijzonder voor het ontwikkelen van software. Standaard biedt Visual Studio met behulp van macro's, add-ins, wizards en de VSIP (Visual Studio Industry Partner) API, de ontwikkelaar de mogelijkheid om Visual Studio uit te breiden en aan te passen aan de eigen wensen. Voor de meeste extensies zijn macro's, add-ins en wizards een prima oplossing. Maar wat nu als we als ontwikkelaar echt willen ingrijpen in de IDE (zoals de code editor)? Dan moeten we ons verdiepen in de VSIP API. Deze is erg uitgebreid en legt de complete interne werking van Visual Studio bloot. Dit is voor veel ontwikkelaars weer iets te veel van het goede. Er zijn namelijk niet veel ontwikkelaars die de behoefte hebben om zelf de support voor een eigen taal in Visual Studio te ontwikkelen. Het bedrijf Developer Express zag dit probleem ook en heeft gratis zijn eigen uitbreidingsframework DXCore voor Visual Studio beschikbaar gesteld. Deze is iets minder uitgebreid dan de VSIP API, maar biedt een uitgebreide interface om de IDE met behulp van plug-ins uit te breiden. Afbeelding 1 laat zien hoe de complexiteit van de diverse uitbreidingsmogelijkheden zich verhouden ten opzichte van het gebruiksgemak.

## Waarom Visual Studio uitbreiden?

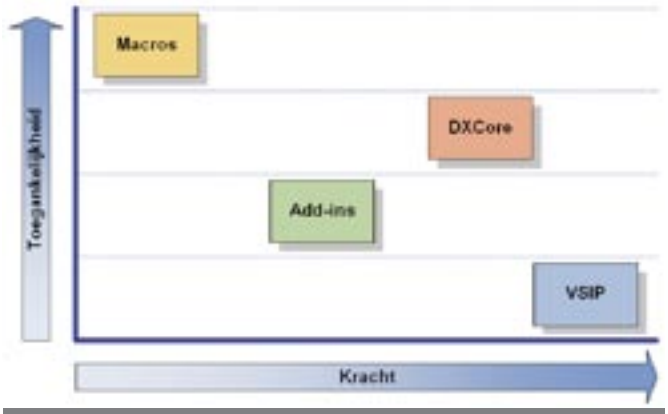
Dat we een ontwikkelomgeving kunnen uitbreiden is natuurlijk geen reden om dat dan ook maar te doen. In de inleiding werd al aangegeven dat het onmogelijk is een ontwikkelomgeving te maken die voor iedereen perfect is. Zoveel mensen zoveel wensen! De volgende opsomming geeft een beperkte lijst van redenen waarom door middel van extensies de Visual Studio-ontwikkelomgeving over het algemeen wordt uitgebreid.

- Veel voorkomende handelingen automatiseren (tijdbesparing)
- Uniforme manier van ontwikkelen afdwingen in een ontwikkelteam (kwaliteit)
- Verhogen van de kwaliteit van de code
- Efficiënter ontwikkelproces
- Integratie met eigen ontwikkelproces (ontwikkelstraat)
- Integratie met externe tools en programma's
- Ondersteuning van de ontwikkelaars bij complexe taken (wizards)
- Ondersteuning voor andere ontwikkeltalen

Visual Studio-uitbreidingen zijn op zich eenvoudig te classificeren in twee categorieën. Onder de eerste categorie vallen de uitbreidingen die het leven van de gebruiker van Visual Studio aangenamer maken, maar niet echt noodzakelijk zijn. Deze uitbreidingen zijn dus vooral 'handig'. Ze doen veel voor de ontwikkelaar, maar mochten ze er niet zijn, dan kan hij zijn taken alsnog uitvoeren, al zullen deze hem meer tijd kosten. Logischerwijs zijn er dan ook de uitbreidingen die in de categorie 'noodzakelijk' vallen. Op zich is er maar een beperkt aantal uitbreidingen dat hier onder valt aangezien er meestal wel een workaround kan worden gevonden. De integratie van bijvoorbeeld een niet standaard source-code control-systeem is niet noodzakelijk. Het maakt het leven van de ontwikkelaar echter een stuk aangenamer als deze er wel is. Net zoals voor softwareprojecten, waarbij gekeken wordt naar de kosten en baten, moet er voor een uitbreiding ook hier naar worden gekeken. Een eenvoudige definitie die kan worden gebruikt om te bepalen of een uitbreiding moet worden gerealiseerd is:

*"Ontwikkel een uitbreiding als de hoeveelheid tijd die verloren gaat door gebruik te maken van de workaround groter is dan de hoeveelheid tijd die het kost om de uitbreiding te maken."*

Door gebruik te maken van deze definitie is het eenvoudig te bepalen of het verantwoord is om een Visual Studio-uitbreiding voor een project, ontwikkelafdeling of ontwikkelstraat te ontwikkelen. Natuurlijk is de maatstaf tijd het eenvoudigste criterium waarmee kan worden gerekend. Criteria zoals kennis, stress en RSI geven misschien een heel ander beeld. De keuze om een uitbreiding te ontwikkelen is dus zeer afhankelijk van de context waarin deze wordt ontwikkeld.




**Afbeelding 1. Uitbreidingsmogelijkheden in Visual Studio 2005**

## Uitbreidingsmogelijkheden voor Visual Studio

De standaard uitbreidingsmogelijkheden van Visual Studio vallen uiteen in twee categorieën. De eerste categorie maakt gebruik van het Visual Studio automation objectmodel. De tweede categorie gaat verder waar het automation objectmodel ophoudt en maakt gebruik van de zogenaamde Visual Studio 2005 SDK. In de eerste categorie vallen de macro's, add-ins en wizards, terwijl de VSIP API in de tweede categorie valt. Een niet standaard uitbreidingsmogelijkheid is DXCore van Developer Expres, deze valt ook in de tweede categorie. Het automation objectmodel bestaat uit een aantal modellen dat zich elk concentreert op een bepaald aspect van de Visual Studio IDE. Zo is er een model voor de code editor, projecttypes, tool windows, debugger, enzovoort. Door gebruik te maken van macro's is eenvoudig te zien welke functionaliteit waar zit in het automation objectmodel. De code die met behulp van de macrorecorder wordt gegenereerd, maakt direct gebruik van het automation objectmodel. Om gebruik te maken van het automation objectmodel moeten de referenties naar de automation libraries *EnvDTE* en *EnvDTE80* worden opgenomen. Voor macro's wordt dit al standaard gedaan. De Visual Studio 2005 SDK is een superset van het automation objectmodel en biedt bijvoorbeeld ook mogelijkheden om in te grijpen op de sourcecode-control en teamsystem-functionaliteit. De keuze of een uitbreiding nu een macro, add-in, wizard of DXCore- of zelfs een VSIP- extensie moet zijn, is afhankelijk van de taak. Het volgende overzicht geeft een opsomming van de kenmerken van de extensies:

- **Macro's**
  - Eenvoudige taken
  - Gemakkelijk voor tijdelijk gebruik
  - Bij distributie is de code inzichtelijk voor iedereen
  - Geen of eenvoudige gebruikersinvoer
  - Opstarten via een toetscombinatie of de Macro Explorer
- **Add-in**
  - Complexe taken
  - Bij distributie is de code niet inzichtelijk voor iedereen
  - Eenvoudige gebruikersinvoer
  - Opstarten via een menuitem of command bar
  - Automatisch laden met Visual Studio
- **Wizard**
  - Complexe taken
  - Bij distributie is de code niet inzichtelijk voor iedereen
  - Complexe gebruikersinvoer met meerdere stappen en/of verificatie
  - Opstarten bijvoorbeeld via New Project of New Item
- **VSIP**
  - Zeer complexe taken zoals nieuwe projecttypes, designers of ondersteuning van nieuwe programmeertalen
  - Bij distributie is de code niet inzichtelijk voor iedereen
  - Vereist kennis van C++
  - Automatisch laden met Visual Studio
- **DXCore**
  - Complexe tot zeer complexe taken
  - Bij distributie is de code niet inzichtelijk voor iedereen
  - Complexe gebruikersinvoer
  - Automatisch laden met Visual Studio (delayloaded)
  - Uitgebreide integratie in de code editor waardoor bijvoorbeeld het tekenen in de code editor eenvoudig wordt
  - Programmeertaal onafhankelijk

## Macro's

Het gebruik van macro's is de eenvoudigste manier om Visual Studio uit te breiden.
Het opnemen van een macro is de gemakkelijkste en waarschijnlijk de meest gebruikte manier om een macro te maken (CTRL+SHIFT+R of via het '**Record TemporaryMacro**' in het '**Macros**'-submenu van het '**Tools**'-menu). Daarnaast is het mogelijk zelf macro's te maken via de Macro's geïntegreerde ontwikkelomgeving (IDE). Deze 'Macros IDE' is apart van de Visual Studio ontwikkelomgeving en wordt gebruikt voor het ontwikkelen, bewerken, testen en runnen van macro's. Het opnemen van een macro is eigenlijk niets anders dan het 'onthouden' van de acties die je als gebruiker uitvoert in de ontwikkelomgeving. Deze acties zijn binnen Visual Studio aanroepen van functionaliteit die deze door middel van een automationmodel beschikbaar stelt. Het resultaat van de opgenomen macro kan dan ook in Visual Basic-code worden bekeken in de 'Macros IDE' (Alt-F11 of via het '**Macros IDE**' in het '**Macros**' submenu van het '**Tools**' menu).

## Add-ins

Add-ins geven de ontwikkelaar meer mogelijkheden dan macro's om de Visual Studio-omgeving uit te breiden. Het zijn gecompileerde applicaties die op verschillende manieren kunnen worden geactiveerd:

- Add-in manager
- Toolbar-commando's
- Buttons
- Devenv command line
- Events zoals het opstarten en afsluiten van Visual Studio

De eenvoudigste manier om een add-in te maken is door gebruik te maken van de add-in Wizard. Deze projectwizard creëert een raamwerk waarin de add-in verder kan worden ingevuld (CTRL+SHIFT+N of via 'Project' in het 'New' submenu van het 'File' menu. Dan 'Visual Studio Add-in' via de 'Extensibility' subfolder in de 'Other Project Types'-folder). Op basis van de wizard wordt een raamwerk gemaakt waarin via de 'Connect'-class een implementatie moet worden gemaakt van de 'IDTExtensibility2'-interface. De add-in die via de wizard is gemaakt, kan direct worden opgestart. Tijdens het runnen van een add-in zal Visual Studio een nieuwe instantie van zichzelf opstarten waarin de nieuwe add-in wordt geladen. Hierdoor kun je eenvoudig de nieuwe add-in te testen en te debuggen. Na het ontwikkelen van een add-in kan deze worden beheerd via de 'add-in manager'. Hierin kun je aangeven of een add-in geladen moet worden, wanneer deze geladen moet worden en met welke parameters deze moet worden opgestart. Om een add-in te kunnen gebruiken in de 'add-in manager' zal deze eerst door middel van een registrykey moeten worden geregistreerd[1].

## Wizards

Bij het maken van een add-in hebben we gebruikgemaakt van de Add-in wizard om een nieuwe add-in te maken. Een wizard in Visual Studio stelt in het algemeen de gebruiker een aantal vragen op basis waarvan code wordt gegenereerd. In Visual Studio zijn drie typen wizards te onderscheiden.

- New Project wizards
  - Deze wizards genereren nieuwe code op basis van een type project. De Add-in wizard valt onder deze categorie. Het helpt de ontwikkelaar een eind op weg door het raamwerk voor het type project alvast op te zetten.
- Add New Items wizards
  - Deze wizards worden gebruikt om nieuwe items zoals HTML-pagina's, XML-pagina's, formulieren, enzovoort aan een project toe te voegen.
- Custom wizards
  - Dit zijn de resterende wizards en worden rechtstreeks vanuit macro's, add-in's of code aangeroepen. Het kan zijn dat deze wizards helemaal geen userinterface hebben en alleen maar code genereren. Deze wizards komen dan ook het minst voor.

Alle wizards hebben gemeenschappelijk dat ze de execute-methode van de IDTWizard- interface implementeren. Deze methode wordt aangeroepen om de wizard op te starten. Wizards dienen een heel specifiek doel. Ze stellen de gebruiker een aantal vragen op basis waarvan in de meeste gevallen code wordt gegenereerd. Dit is natuurlijk niet verplicht. Wizards kunnen bijvoorbeeld ook worden gebruikt om de gebruiker door middel van een aantal vragen een complexe configuratie te laten uitvoeren. Het algemene doel van een wizard is wel om via een stappenplan een complexe handeling te vereenvoudigen.

## VSIP

De VSIP API kunnen we gebruiken om deep-down integratie met Visual Studio te realiseren. Microsoft levert hiervoor de Visual Studio SDK. Met deze SDK is het mogelijk om bijvoorbeeld de volgende functionaliteit te realiseren:

- Nieuwe ontwikkeltaal
- Designers en editors
- Custom debugging
- Integratie van tooling
- Nieuwe projecttypes
- Team System-functionaliteit

Aan de lijst is te zien dat de VSIP API grote gedeeltes van het complete onderliggende framework van Visual Studio blootlegt. Het maakt niet uit of het nu om de debugger, editor of designer gaat. Als ontwikkelaar kun je met behulp van de VSIP API integreren in Visual Studio. Het is dus niet verwonderlijk dat Microsoft's eigen ontwikkelteams voor C# en Visual Basic gebruik hebben gemaakt van de VSIP API om te integreren in de Visual Studio .NET-ontwikkelomgeving. De VSIP API is ontwikkeld in C++ en vereist dan ook aanzienlijke C++-kennis van de ontwikkelaar. Dit in combinatie met het feit dat de VSIP API wel erg uitgebreid is, zorgt ervoor dat de leercurve van de VSIP API steil is. Als ontwikkelaar zul je dan ook flink wat tijd moeten investeren om de VSIP API onder de knie te krijgen.

## DXCore

DXCore is een framework voor integratie met Visual Studio van het bedrijf Developer Express (www.devexpress.com). Het dient als basis voor de producten CodeRush en Refactor! Als je DXCore vergelijkt met de VSIP API, dan kom je al snel tot de conclusie dat de VSIP API uitgebreider is dan DXCore. DXCore is echter gemakkelijker te begrijpen en maakt die delen van Visual Studio beschikbaar die voor de ontwikkelaar van extensies echt nodig zijn. Er is maar een zeer beperkt aantal ontwikkelaars dat de behoefte heeft om zelf de ondersteuning voor een ontwikkeltaal te implementeren in Visual Studio. Mocht je dit of aanverwante taken toch willen, dan moet je de VSIP API gebruiken. Voor de rest van ons die productiviteitverhogende extensies willen schrijven is DXCore een uitkomst. Het biedt net de juiste balans tussen complexiteit en flexibiliteit. Het produkt Refactor! is een mooi voorbeeld van de kracht van DXCore. Refactor! laat de gebruiker visueel zien wat er van hem verwacht wordt zonder gebruik te maken van irritante modal dialogboxes. Alles wordt in de editor weergegeven. DXCore kan gratis worden gedownload van de site van Developer Express. Het kan echter niet worden gebruikt in de express edities van Visual Studio.

## Een voorbeeld

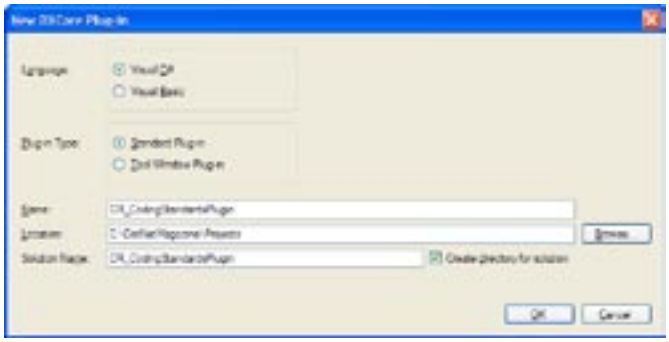
Al die uitbreidingsmogelijkheden vragen natuurlijk om een voorbeeld. Als voorstanders van 'onder architectuur ontwikkelen' gaan we een stukje van de codingstandards voor C# integreren in de IDE van Visual Studio. Het idee is om afwijkingen in naamconventies visueel onder de aandacht te brengen van de ontwikkelaar. Op dit moment is DXCore van Developer Express de enige mogelijkheid om door middel van plug-ins eenvoudig visuele aspecten toe te voegen aan de IDE. Het voorbeeld is dan ook gebaseerd op DXCore.

De architectuur van de plug-in bestaat uit een DXCore-derivaat met daarin een mechanisme om handler-objecten voor taalelementen te registreren. Elke keer dat de Visual Studio IDE de tekst in de editor opnieuw tekent, wordt de tekst door de handlers gehaald. Als een handler heeft bepaald dat het taalelement niet aan de conventie voldoet, wordt een decoratie onder het taalelement getekend. Door met de muis over deze decoratie te zweven wordt een hint getoond met daarin de oplossing voor de conventieovertreding. De plug-in implementeert voor tien taalelementen een naamgevingconventie. De plug-in is gemakkelijk uit te breiden met eigen conventies. Naast controle op naamgeving kan door middel van een nieuwe handler ook eenvoudig worden gecontroleerd op andere codingconventies.

Het maken van een DXCore-plug-in begint met het downloaden en installeren van DXCore. Na het downloaden[2] en installeren van DXCore is er in het hoofdmenu van de Visual Studio IDE een nieuw menuitem met de tekst **'DevExpress'**. Via het **'New Plug-in'**-submenu in het |**'DevExpress'**-menu kun je door middel van een wizard (afbeelding 2) een nieuwe DXCore-plug-in te maken.
Kies voor de gewenste taal, in dit geval C# en kies voor een standaard plug-in. Een 'Tool Window' plug-in bevat een venster dat binnen de IDE gedockt kan worden. Deze functionaliteit hebben we overigens niet nodig voor dit artikel. Voer als laatste de naam in van de plug-in en kies voor OK. In het volgende scherm (afbeelding 3) typen we de titel van de plug-in en bepalen we het moment waarop de plug-in wordt geladen.

Het standaard ingestelde 'On Demand' load type is het efficiëntst en dit laten we zo. De optie System Plug-in zorgt er voor dat een plug-in geladen wordt voordat alle andere plug-ins worden geladen. Deze optie heb je zelden nodig. Na het bevestigen via de OK-knop opent Visual Studio de door de wizard gemaakte solution. In de solution is het plug-in-project te vinden met daarin de plug-in. DXCore plug-ins hebben een eigen designer. Dit houdt in dat veel functionaliteit via de toolbox en properties kan worden geregeld, zoals we later ook zullen zien. DXCore bevat een uitgebreide architectuur die de meeste onderdelen van de Visual Studio IDE beschikbaar stelt aan ontwikkelaars. In ons geval zijn we op zoek naar een event dat ons de mogelijkheid geeft om taalelementen in de editor van een visuele decoratie te voorzien als ze niet voldoen aan de conventie. Taalelementen heten in DXCore Language Elements en zijn te vinden in de DevExpress.CodeRush.StructuralParser-namespace. Een overzicht van de door onze add-in gebruikte language elements is te vinden in afbeelding 4.

De DxCore StandardPlugIn-klasse heeft een event genaamd EditorPaintLanguageElement. Dit event is beschikbaar via de properties van de plug-in designer en het is precies wat we zoeken. Iedere keer dat



Afbeelding 2. Nieuwe DXCore plug-in


Afbeelding 3. Plug-in type kiezen

Afbeelding 4. DXCore taalelementen

de IDE de inhoud van de editor tekent wordt voor alle taalelementen dit event een keer aangeroepen. Binnen het event hebben we toegang tot het betreffende taalelement en het tekenoppervlak van de editor. De CodingStandardsPlugin behorend bij dit artikel bevat een raamwerk voor het registreren van taalelementen en hun conventies. Dit raamwerk wordt aangeroepen vanuit het EditorPaintLanguageElement-event om zo te bepalen of een taalelement moet worden verrijkt met een decoratie. We hebben gekozen voor een rode halftransparante lijn voor alle taalelementen die niet aan de conventie voldoen (afbeelding 5).

Het DXCore-raamwerk bevat een klasse genaamd Underline die de functionaliteit van het tekenen van de lijn voor zijn rekening neemt. Het gebruik van deze klasse is als volgt:

```
Underline underline = new Underline();
```

Vervolgens moeten drie properties gezet worden die de klasse vertellen waar en hoe de Underline getekend moet worden. De eerste property is TextView, de instantie van de editor die op dit moment actief is. Deze wordt doorgegeven als property op de event-arguments van het EditorPaintLanguageElement-event.

```
underline.TextView = ea.PaintArgs.TextView;
```

De tweede property is de locatie en breedte van de Underline. Aangezien we alleen de foutieve tekst willen voorzien van een Underline moeten we zoeken naar een range die dit vertegenwoordigt. In eerste instantie lijkt ea.LanguageElement.Range wat we zoeken, maar een snelle test (afbeelding 6) wijst uit dat deze range meer bevat dan alleen de foutieve tekst.

Een language-element omvat alle code die tot dat element behoort, inclusief commentaar en gereserveerde woorden. We zoeken een deel van deze range, namelijk het gedeelte dat de naam van het

```
namespace CodingStandards
{
    /// <summary>
    /// Class with wrong convention
    /// </summary>
    class classWithWrongConvention
    {
    }

    /// <summary>
    /// Class with correct convention
    /// </summary>
    class ClassWithCorrectConvention
    {

    }
```

Afbeelding 5. Voorbeeld van foutieve klassenaam

```
/// <summary>
/// Class with wrong convention
/// </summary>
class classWithWrongConvention
{
}
```

Afbeelding 6. Foutieve range

taalelement bevat. Gelukkig bestaat er ook een NameRange, die wel het gewenste resultaat heeft

.

```
underline.Range = ea.LanguageElement.NameRange;
```

De kleur wordt bepaald via de derde property, deze FillColor-property zetten we op een half transparante rode kleur.

```
underline.FillColor = Color.FromArgb(128, Color.Red);
```

Als laatste moeten we de Underline vertellen waarop hij zichzelf moet tekenen. Dit doen we door gebruik te maken van de Paint-methode. Deze Paint-methode verwacht een Graphics-instantie die we ontvangen in de argumenten van het event waar we inzitten.

```
underline.Paint(ea.PaintArgs.Graphics);
```

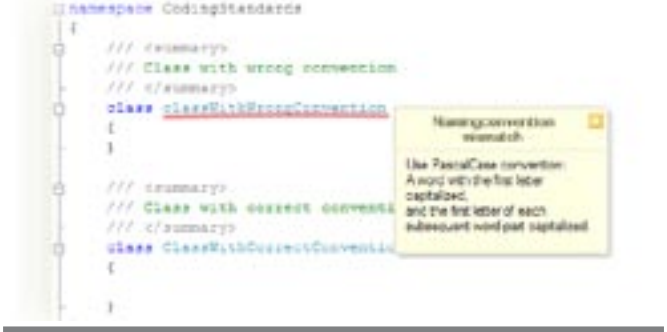
Nu we een lijn kunnen tekenen onder alle namen die niet aan de conventies voldoen, willen we de ontwikkelaar laten zien wat er verkeerd is aan de naamgeving. Als een ontwikkelaar met de muis over een Underline gaat, willen we een hint tonen waarin de uitleg van de conventie staat. De Underline-klasse kan dit niet, maar DXCore biedt wel een andere mogelijkheid om dit te realiseren. Een Tile is een soort van hotspot die kan worden gecreëerd in de editor.  Door iedere Underline te voorzien van een Tile kunnen we per Underline MouseEnter- en MouseLeave-events krijgen. Tiles zijn iets complexer in het gebruik dan Underlines. Een Tile heeft namelijk een EventHub nodig om de gewenste notificaties door te sturen naar de plug-in. De StandardPlugIn waarvan onze klasse is afgeleid, bevat al een eventhub en de bijbehorende events, TileMouseEnter en TileMouseLeave, kunnen direct worden gebruikt. Een Tile met de plug-in als EventHub kan snel worden verkregen door de methode NewTile aan te roepen op de plug-in.

```
Tile tile = this.NewTile(underline.Bounds, handler);
```

Wat nog resteert is het koppelen van de Tile aan de editor. Dit gebeurt door de Tile toe te voegen aan de tile-collectie van de editor. Deze TextView wordt doorgegeven via de event arguments van het EditorPaintLanguageElement-event.

```
ea.PaintArgs.TextView.AddTile(tile);
```

Via de events van de plug-in kunnen we gemakkelijk de event handlers creëren voor de events TileMouseEnter en TileMouseLeave. De



Afbeelding 7. Big hint services

events voor alle door ons aangebrachte tiles komen via deze events binnen. Een van de properties van de Tile-klasse biedt de mogelijkheid om een eigen object door te geven met de Tile. Tijdens het maken van de Tile hebben we deze property voorzien van de naamconventie-handler die wij voor dat taalelement hebben aangemaakt. In de muisevents kunnen we via de doorgegeven Tile weer toegang krijgen tot deze handler.
In het laatste stuk van deze plug-in tonen we de hint als gevolg van een MouseEnter en verbergen we de hint als gevolg van de MouseLeave. De DXCore-achitectuur bevat een prachtige oplossing voor hints in de vorm van BigHintServices. De BigHintServices kunnen we benaderen via het CodeRush publieke object. Voor het tonen van een hint hebben we drie zaken nodig. Als eerste de locatie waar de hint getoond moet worden, als tweede de titel van de hint en als laatste de inhoud van de hint. We willen de hint aan de rechterkant van de foutieve naam weergeven. Hiervoor moeten we het meest rechtsliggende punt van de Tile vertalen naar schermcoördinaten.

```
Point location = ea.Tile.TextView.ToScreenPoint(new
   Point(ea.Tile.Bounds.Right, ea.Tile.Bounds.Top));
```

De inhoud van de hint is ook dynamisch en deze halen we uit de handler die we hebben verbonden aan de Tile.

```
ConventionHandlerBase handler = ea.Tile.Object as ConventionHandlerBase;
...
hint.Text = handler.GetHintText();
```

Als laatste tonen we de hint (afbeelding 7) op de berekende positie door de ShowAt-methode aan te roepen op de verkregen hint.

```
hint.ShowAt(location);
```

## Net even verder

Met dit voorbeeld is duidelijk geworden dat zelfs op het oog complexe uitbreidingen eenvoudig zijn te implementeren. De standaard uitbreidingsmogelijkheden zoals macro's, add-ins en wizards geven ons als ontwikkelaar al veel mogelijkheden om Visual Studio aan te passen. Met de VSIP API en DXCore hebben we echter de mogelijkheden om net even dat stapje verder te gaan. Hierdoor kunnen we naadloos integreren in de ontwikkelomgeving en ontstaan er nieuwe mogelijkheden om Visual Studio naar je hand te zetten.

**Voetnoot**

1 Zie de referentie over het registreren van Add-ins over welke entries er precies moeten worden aangemaakt.

2 DXCore kan worden gedownload via http://www.devexpress.com/Downloads/NET/DXCore/

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

**Referenties:**

VSIP – http://msdn.microsoft.com/vstudio/partners
Extend Visual Studio – http://msdn.microsoft.com/vstudio/extend/
Registratie Add-ins - http://msdn.microsoft.com/library/en-us/vsintro7/html/vxconadd-inregistration.asp
DevExpress DXCore – http://www.devexpress.com/Downloads/NET/DXCore