Leon Krancher is consultant bij Avanade

Custom Data Flow Components

SQL SERVER 2005 INTEGRATION SERVICES UITBREIDEN

Van native ondersteuning van XML tot transformaties op basis van fuzzy logic. Ten opzichte van zijn voorganger biedt SQL Server 2005 Integration Services (SSIS) heel wat nieuwe mogelijkheden. Desondanks blijven proprietary file-formaten en business-specifieke logica moeilijk te vangen met generieke componenten. Toch is het mogelijk ook deze uitzonderlijke gevallen met SSIS te verwerken. Microsoft heeft namelijk de standaardfunctionaliteit van SSIS uitgebreid met zogenaamde custom data flow-components. In dit artikel gaan we in op de vraag hoe zo'n custom data flow-component precies werkt en hoe je er zelf één ontwikkelt.

Voordat we met het ontwikkelen van een custom data flow-component beginnen, besteden we eerst aandacht aan wat een custom data flow-component precies is. In essentie is een custom data flow-component een .NET-assembly die een specifieke functie kan vervullen en deze functie zo aanbiedt dat een data-flow de functie op een specifieke manier kan configureren en gebruiken. De parallel is goed te leggen met een windows forms-control. De button-control heeft bijvoorbeeld als functie het aanbieden van een knop. Een applicatie moet de button-control vervolgens zo configureren dat ze de gewenste functie, zoals het afsluiten van de applicatie, gaat vervullen. Een custom data flow-component is eigenlijk hetzelfde. De standaard meegeleverde 'sort'-functie heeft bijvoorbeeld als generieke functie het sorteren van data. Dit betekent dat je in een specifieke data flow alle klanten op leeftijd kunt sorteren. De kracht van het model van custom data flow-components is dan ook dat je functionaliteit eenvoudig in verscheidene transformations en zelfs in meer SSIS-packages kunt hergebruiken. Intern maakt SQL Server 2005 Integration Services een onderscheid tussen drie verschillende typen custom data flow-components:

- Datasources, verantwoordelijk voor het lezen van data.
- Targets, verantwoordelijk voor het wegschrijven van data.
- Transformations, verantwoordelijk voor het leveren van alle overige functionaliteit.

Binnen de categorie transformations wordt verder nog een onderscheid gemaakt tussen synchrone en asynchrone transformaties. Synchrone transformaties zijn componenten die voor de werking alleen naar een enkel record kijken. Een voorbeeld hiervan is de 'conditional split' die op basis van de inhoud van een record bepaalt naar welke output het record moet worden weggeschreven. Een asynchrone transformatie heeft juist weer als eigenschap dat zij alle records nodig heeft om een resultaat te kunnen bereiken. Denk hierbij bijvoorbeeld aan de count-functie, die pas weet hoeveel rijen een dataset bevat als ze de volledige input heeft ontvangen.
Hoewel voor het ontwikkelen van al deze vier typen data flow components ongeveer dezelfde stappen moeten worden doorlopen, kent elk type toch zijn eigen specifieke aandachtspunten. Aangezien de beschikbare ruimte voor dit artikel beperkt is, richten we ons voor dit artikel specifiek op het ontwikkelen van een synchroon data flow-component. Voor het ontwikkelen van zo'n component zullen we in ieder geval:

- logica moeten schrijven, zodat de component in de data flow-designer kan worden gebruikt;
- logica moeten schrijven voor het daadwerkelijk verwerken van de data;
- er voor moeten zorgen dat de component daadwerkelijk kan worden gebruikt;
- moeten weten hoe we de run- en design time-methoden van de component kunnen debuggen.

Aangezien deze stappen nogal abstract zijn, illustreren we ze door het uitwerken van een eenvoudig voorbeeld: een component die aan de hand van een configureerbare regular expression bepaalt of de input wel of niet voldoet.

Aan de slag

Het programmeren van een custom data flow-component begint met het aanmaken van een nieuw Class Library Project in Microsoft Visual Studio 2005. Voor dit voorbeeld noemen we het project 'SyncDataFlow'. De automatisch aangemaakte 'Class1.cs' hernoemen we meteen naar het meer toepasselijke 'RegExMatch.cs', het herkennen van een patroon met behulp van een reguliere expressie.
Voor het ontwikkelen van custom data flow-components maken we gebruik van een aantal specifieke SSIS-classes en -interfaces.
De volgende stap is het aan het project toevoegen van references naar de .NET-components die deze objecten voor ons project beschikbaar maken. Dit zijn:

- Microsoft.SqlServer.DTSPipelineWrap
- Microsoft.SqlServer.DTSRuntimeWrap
- Microsoft.SqlServer.PipelineHost

Na het aanmaken van deze referenties kunnen de specifieke SSIS-namespaces in de using-sectie van de 'RegExMatch' worden opgenomen. Omdat we voor dit voorbeeld van de classes Arraylist en Regex gebruikmaken, moeten we ook de namespaces van die classes in de using-sectie van de class 'RegexFilter' opnemen. De volledige using-sectie met zowel de SSIS(a) als de namespaces(b) Arraylist en Regex is weergegeven in codevoorbeeld 1.

Een custom data flow-component moet de interfaces IDTSDesigntimeComponent90 en IDTSRuntimeComponent90 implementeren om door respectievelijk Visual Studio en SSIS runtime te kunnen worden aangeroepen. In een poging het ontwikkelen van een custom data flow-component eenvoudiger te maken, heeft Micro-

soft de CustomPipeLine helper-class in het leven geroepen. Naast het object ComponentMetaData dat verderop wordt behandeld, biedt deze helper-class een blanco implementatie van alle methoden uit beide interfaces. Door de component van deze helperclass over te erven, kunnen wij ons beperken tot het overriden van de methoden die we daadwerkelijk willen gebruiken.

Om de component ook op de juiste manier in de Data Flow-toolbox van Visual Studio zichtbaar te maken, hebben we het attribuut DtsPipelineComponent nodig. Met dit attribuut kunnen zaken als de naam, de beschrijving en het icoontje van de component op een eenvoudige manier worden gespecificeerd. De nieuwe declaratie van deze class, inclusief overerven van het DtsPipelineComponent is weergegeven in codevoorbeeld 2.

ProvideComponentProperties()

De eerste methode waar we aandacht aan besteden, is de methode ProvideComponentProperties() uit de interface IDTSDesigntimeComponent90. Deze methode wordt door de data flow-designer aangeroepen op het moment dat de component aan een data-flow wordt toegevoegd en heeft als doel het initialiseren van alles wat nodig is om de component in de data flow-designer te kunnen gebruiken. Voor deze component moeten de volgende zaken geïnitialiseerd worden:

- De input, zodat het aanleveren van data geconfigureerd kan worden;
- De output, zodat het wegschrijven van data geconfigureerd kan worden;
- Een object waarmee we een regular expression kunnen configureren.

De waarden die voor deze objecten worden geconfigureerd, moeten als onderdeel van het SSIS-package worden opgeslagen. Om dit mogelijk te maken, beschikt de base class PipelineComponent over het object ComponentMetaData. Alle objecten die door middel van het object ComponentMetaData zijn aangemaakt, worden automatisch als onderdeel van de data flow opgeslagen. Bij het uitvoeren van het SSIS-package worden deze opgeslagen waarden vervolgens weer automatisch uitgelezen en de objecten opnieuw gecreëerd, zodat de data flow-specifieke configuratie automatisch wordt meegenomen. De syntax voor het aanmaken van de input en output door middel van het object ComponentMetaData, is weergegeven in sectie a van codevoorbeeld 3.

Naast het aanmaken van een input en output is het voor een synchroon data flow-component ook van belang beide aan elkaar te koppelen. Een data flow-component kan immers meer dan één input hebben; dan is het belangrijk te weten welke outputs met een specifieke input geassocieerd zijn. De code om deze koppeling te realiseren, is weergegeven in sectie b van codevoorbeeld 3.
Bij het schrijven van een custom transformation is het aanmaken en initialiseren van één input en één output in principe genoeg. Het vervelende is alleen dat één output betekent, dat alle rijen die niet voldoen automatisch verloren gaan. In voorkomende gevallen willen we deze rijen ook kunnen inzien. Daarom gaan we een tweede outputobject aanmaken dat als doel heeft de foutieve records af te voeren.

De eerste stap in het aanmaken van output, die specifiek bedoeld is om error-records weg te schrijven, is door het object ComponentMetaData aan de overkoepelende data flow bekend te laten maken dat we van deze optionele functionaliteit gebruik willen maken. Om te voorkomen dat de input naar zowel de originele als de error-output wordt weggeschreven, kunnen we voor de originele output de property exclusiongroup specificeren. Door dezelfde waarde vervolgens ook aan de error-output mee te geven, wordt automatisch voorkomen dat één ingaande rij naar meer outputs wordt geschreven. De error-output kan vervolgens worden gecreëerd door het aanroepen van de methode AddErrorOutput. De AddErrorOutput-methode wordt door de pipeline component base-class geleverd en heeft de naam van de error-output, het ID van de corresponderende input en de exclusiongroup als parameters. De volledige syntax voor het aanmaken van de error-output is weergegeven in sectie c van codevoorbeeld 3.

Naast het aanmaken van de in- en output moesten we ook nog de geconfigureerde regular expression opslaan. Ook dit soort custom properties wordt opgeslagen met behulp van de eerder geïntroduceerde componentmetadata. Specifiek voor custom properties bevat de class componentmetadata een property-collection die in staat is tijdens het design geconfigureerde waarden op te slaan en tijdens het uitvoeren van het package automatisch opnieuw te laden. Het is overigens aan te raden de naam en description-property van custom properties altijd te specificeren. Als deze niet zijn gespecificeerd, is het niet mogelijk in de data flow-designer de functie van een custom propertry te achterhalen. De code om dit te realiseren is weergegeven in sectie d van codevoorbeeld 3.

```
// 1a. De using statements specifiek voor SSIS
using Microsoft.SqlServer.Dts.Pipeline;
using Microsoft.SqlServer.Dts.Pipeline.Wrapper;
using Microsoft.SqlServer.Dts.Runtime;
using Microsoft.SqlServer.Dts.Runtime.Wrapper;

// 1b. De using statements voor de Arraylist en regex
using System.Text.RegularExpressions;
using System.Collections;
```

Codevoorbeeld 1.

```
//2. Implementatatie DtsPipelineComponent interface
//   en overerven PipelineComponent helperclass
[DtsPipelineComponent(DisplayName = "RegExMatch",
  ComponentType = ComponentType.Transform)]
public class RegExMatch : PipelineComponent
{
  ...
}
```

Codevoorbeeld 2.

```
public override void ProvideComponentProperties()
{
  // 3a. het aanmaken van de in- en output objecten
  IDTSInput90 primaryInput =
            ComponentMetaData.InputCollection.New();
  IDTSOutput90 primaryOutput =
            ComponentMetaData.OutputCollection.New();

  // 3b. het koppelen van de output aan de input
  primaryOutput.SynchronousInputID = primaryInput.ID;

  primaryInput.Name = "Input";
  primaryOutput.Name = "Valid rows";

  // 3c. Het toevoegen van de errorOutput
  ComponentMetaData.UsesDispositions = true;
  primaryOutput.ExclusionGroup = 1;
  AddErrorOutput("Invalid rows", primaryInput.ID, 1);

  // 3d. Het toevoegen van de custom property
  IDTSCustomProperty90 regExPattern =
         ComponentMetaData.CustomPropertyCollection.New();
  regExPattern.Name = "RegEx Pattern";
  regExPattern.Description = "Specify your regular expression here";
}
```

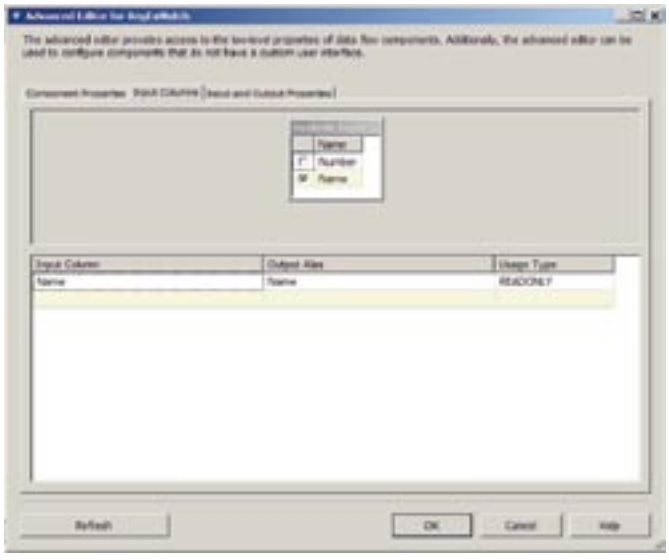
Codevoorbeeld 3.

Validate()

De methode Validate wordt aangeroepen wanneer de configuratie van een data flow-component in een SSIS-package wordt aangepast en heeft als doel de aangepaste configuratie van de component te valideren. De werking van de methode is eenvoudig; afhankelijk van de configuratie van de component geeft de methode een waarde van de enumeration DTSValidationStatus terug. Het is niet mogelijk een SSIS-package te compileren, totdat alle gebruikte data flow-componenten DTSValidationStatus.IsValid() retourneren. Het resultaat van de methode Validate() is niet zichtbaar in de designer. Om te voorkomen dat configuratieproblemen pas aan het licht komen tijdens het compileren van het SSIS-package, beschikt de class ComponentMetaData over de methoden FireError, de FireWarning en de FireInformation. Het aanroepen van deze methoden tijdens de Validate() zorgt er voor dat de gebruiker de corresponderende melding ook in de SSIS-package-designer te zien krijgt. Omdat we voor de component gebruik kunnen maken van de in het .NET Framework aanwezige RegEx-functionaliteit, is de implementatie van de validatielogica gemakkelijk. Het enige dat we moeten doen is het uitlezen van de geconfigureerde expression en het creëren van een nieuw regex-object op basis van deze expression. Als dit goed gaat is de configuratie valide en als er een fout optreedt, is de configuratie ongeldig. De syntax voor de validate-methode voor dit voorbeeld is weergegeven in codevoorbeeld 4.

VirtualInput en OnInputPathAttached()

In SQL Server 2005 Integration Services krijg je standaard niet alle kolommen terug uit het input-object. Tijdens het ontwerp van een SSIS-package moet expliciet gespecificeerd worden welke kolommen voor een bepaalde component toegankelijk zijn. Helaas is dit gedrag niet altijd even goed gedocumenteerd. Het is dan ook mogelijk dat je dit pas ontdekt terwijl je druk bezig bent met het debuggen van een SSIS-package. Het daadwerkelijk beschikbaar stellen van de kolommen is betrekkelijk eenvoudig. Tijdens het ontwerpen van een SSIS-package kan je de 'advanced editor' voor een specifieke component starten door met de rechtermuisknop op de component te drukken en de optie 'show advanced editor' te selecteren. Onder het tabblad 'input' vind je vervolgens alle opties die nodig zijn voor het specificeren van de toegankelijkheid van de verschillende kolommen. De advanced editor is weergegeven in afbeelding 1. Naast het handmatig beschikbaar maken van de kolommen uit de input kan dit uiteraard vanuit de code. De beste plaats hiervoor is de methode OnInputPathAttached(). Deze methode wordt aangeroepen op het moment dat input aan de component wordt gekoppeld en is daarmee de eerste plaats waar de kolommen uit de input daadwerkelijk beschikbaar komen. Met behulp van de gespecificeerde inputID is eenvoudig te achterhalen voor welke IDTSInput90 de methode is aangeroepen. Door vervolgens de methode getVirtualInput() van deze IDTSInput90 aan te spreken, krijg je het object IDTSVirtualInput90 dat alle beschikbare kolommen bevat. Met behulp van de methode SetUsageType kunnen deze kolommen vervolgens beschikbaar worden gesteld. In codevoorbeeld 5 is de methode OnInputPathAttached() uitgewerkt, zodat ze de bovenstaande stappen doorloopt en automatisch alle kolommen van het type string of unicode string toegankelijk maakt.



Afbeelding 1. De advanced editor

Input en de functie van de PreExecute()

Geheel afhankelijk van de hoeveelheid data en de structuur van het SSIS-package kan het zo zijn dat een data flow-component tijdens het uitvoeren van een SSIS-package niet alle data in één keer krijgt aangeleverd. Vooral bij synchrone transformaties is het immers niet handig om met het verwerken van de eerste records te wachten totdat ook de laatste zijn ingeladen. Je moet er daarom vanuit gaan dat het niet erg is als de verwerkende methode ProcessInput()

```
//4. De validatie logica
public override DTSValidationStatus Validate()
{
  bool pbCancel = false;
  // 4a. Probeer op basis van de geconfigureerde expression
  //  een Regex object te maken.
  try
  {
    Regex testRegex = new Regex((string)
    ComponentMetaData.CustomPropertyCollection["RegEx Pattern"]
      Value);
    return DTSValidationStatus.VS_ISVALID;
  }
  catch (Exception e)
  {
    // 4b. Als er een Exception optreed is de geconfigureerde
    //  expression niet geldig, raise een error
    ComponentMetaData.FireError(0, ComponentMetaData.Name,
    "The Regular Expression is invalid, error: " +e. Message,
    "", 0, out pbCancel);
    return DTSValidationStatus.VS_ISBROKEN;
  }

}
```

Codevoorbeeld 4.

```
public override void OnInputPathAttached(int inputID)
{
  // 5.1 Bepaal voor welke input de methode wordt aangeroepen
  IDTSInput90 newInput =
    ComponentMetaData.InputCollection.FindObjectByID(inputID);

  // 5.2 Haal de virtual input op en loop door alle kolommen
  IDTSVirtualInput90 virtualInput = newInput.GetVirtualInput();
  foreach (IDTSVirtualInputColumn90 virtualColumn in
    virtualInput.VirtualInputColumnCollection)
  {
    // 5.3 Als een kolom van het type string is moet hij
    //     beschikbaar worden gemaakt
    if ((virtualColumn.DataType == DataType.DT_STR) ^
      (virtualColumn.DataType == DataType.DT_WSTR))
    {
      SetUsageType(newInput.ID, virtualInput,
        virtualColumn.LineageID,
        DTSUsageType.UT_READONLY);
    }
  }
}
```

Codevoorbeeld 5.

meer dan één keer wordt aangeroepen. Om dit mogelijk te maken is de methode PreExecute() in het leven geroepen. Deze methode wordt gebruikt op het moment dat de component tijdens het uitvoeren van een SSIS-package voor de eerste keer is gecreëerd en is daarmee bij uitstek geschikt om variabelen te initialiseren en externe resources te laden.

Het object PipelineBuffer bevat de invoerdata en is in feite een tabel die wordt meegegeven bij het aanroepen van de methode ProcessInput(). Het enige doel van dit object is het aanbieden van de records die moeten worden verwerkt. Het PipelineBuffer-object bevat naast datakolommen ook een aantal kolommen met interne SSIS-data. Om te voorkomen dat we met de component per ongeluk de verkeerde data manipuleren, kun je door het aanroepen van de methode BufferManager.FindColumnByLineageID() voor elke geconfigureerde tabel in de input uitzoeken welke data voor jou relevant zijn. Deze informatie kun je vervolgens in een globale variabele opslaan. Ondanks dat tijdens design-time al is aangegeven waar de component zijn foutieve records naartoe moet schrijven, moet je dit tijdens runtime nogmaals specificeren. Gelukkig kunnen we aan de hand van de property IsErrorOut eenvoudig zien welke van de twee inputs de error is en deze in een global variable opslaan. De laatste stap in de PreExecute() van het specifieke voorbeeld is het uitlezen van de custom property met het regex-patroon en het creëren van een nieuw regex-patroon op basis van deze waarde. In codevoorbeeld 6 is het proces weergegeven van het lokaliseren van de kolommen(6.1), het lokaliseren van de juiste output(6.2), het creëren van het regex-object(6.3) en het creëren van de benodigde global variables om dit allemaal in op te slaan(6.4).

ProcessInput

Het enige dat we nog moeten doen in de ProcessInput-methode is het uitlezen van de juiste kolommen uit de buffer, het testen van de waarden tegen de regex en - afhankelijk van het resultaat daarvan - de rij wegschrijven naar de validOutput of de errorOutput. De code voor de volledige methode processInput() is weergegeven in codevoorbeeld 7.

Building & deploying

Om custom data flow-componenten zo eenvoudig mogelijk te kunnen hergebruiken, wordt in een SSIS-package alleen de specifieke configuratie van de component opgeslagen. De component zelf moet in de Global Assembly Cache (GAC) worden opgeslagen. Componenten die in de GAC worden opgeslagen, moeten zijn gesigned. Dus voordat we de component kunnen compileren, moeten we hem eerst signen. De gemakkelijkste manier om dit te doen is in Visual Studio bij project-properties. Onder het tabblad 'Signing' vind je de mogelijkheid de assembly te signen. Mocht je nog geen bestaande key hebben, dan kun je hier ook eenvoudig een key laten genereren. Met het signen en het plaatsen van de component in de GAC alleen zijn we er nog niet. Om de custom data flow-component ook in de data flow-designer te kunnen gebruiken, moet er een kopie van de assembly in de 'C:\Program Files\Microsoft SQL Server\90\DTS\PipelineComponents'-directory worden geplaatst. Om het kopiëren en het in de GAC plaatsen te automatiseren, kun je de getoonde script in codevoorbeeld 8 als post build-event configureren. (Een post build-event kun je onder het tabblad 'build Events' van het project properties-menu configureren.)

Gebruiken & debuggen

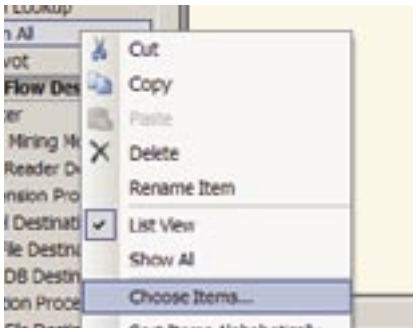
Om de custom data flow-component daadwerkelijk in een SSIS-package te kunnen gebruiken, moeten we eerst in Visual Studio een nieuw Integration Services-project creëren en een data flow-task aan dit project toevoegen. Door met de rechtermuisknop op de toolbox te drukken en de optie 'choose items' te selecteren, kunnen we vervolgens een item aan de SSIS data flow-toolbox toe-

```
public override void PreExecute()
{
  // 6.1 lokaliseer de gewenste kolommen in de input buffer.
  IDTSInput90 input = ComponentMetaData.InputCollection[0];
  foreach (IDTSInputColumn90 currentColumn in
    input.InputColumnCollection)
  {
    bufferColumnIndex.Add(
    (int)BufferManager.FindColumnByLineageID
    (input.Buffer, currentColumn.LineageID));
  }

  // 6.2 Stel vast welke Ouput de juiste is en welke de errorOutput
  foreach (IDTSOutput90 configuredOutput in
    ComponentMetaData.OutputCollection)
  {
    if (configuredOutput.IsErrorOut)
    {
      errorOutput = configuredOutput;
    }
    else
    {
      validOutput = configuredOutput;
    }
  }

  // 6.3 Creëer op basis van de geconfigureerde waarde
  //     het benodigde Regex object
  lineValidator = new Regex((string)
    ComponentMetaData.CustomPropertyCollection["RegEx Pattern"]
    .Value);
}
```

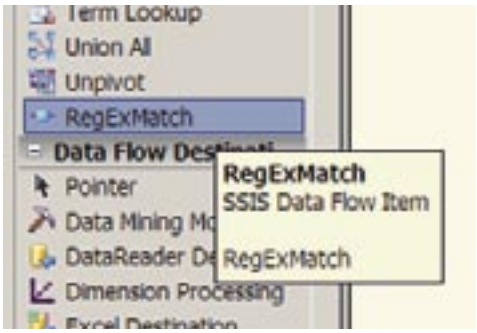
Codevoorbeeld 6.

```
public override void ProcessInput(int inputID, PipelineBuffer buffer)
{
  bool rowIsValid;

  // 7.1 lees alle regels uit de buffer
  while (buffer.NextRow())
  {
    rowIsValid = true;
    // 7.1 loop door alle tijdens de pre-execute inkaart gebrachte
    //     kolommen
    foreach (int indexPos in bufferColumnIndex)
    {
        // 7.2 Test of de kolom voldoet aan de regular expression
        string columnValue = buffer.GetString(indexPos);
        if (!lineValidator.IsMatch(columnValue))
        {
            // 7.3 Onderneem actie als dit niet zo is
            rowIsValid = false;
            buffer.DirectErrorRow(errorOutput.ID, 100, indexPos);
            break;
        }
    }

    // 7.4 Als alle kolommen geldig waren kan de rij naar de
    //     normale output worden weggeschreven.
    if (rowIsValid)
    {
        buffer.DirectRow(validOutput.ID);
    }
  }
}
```

Codevoorbeeld 7.



Afbeelding 2. Item toevoegen.



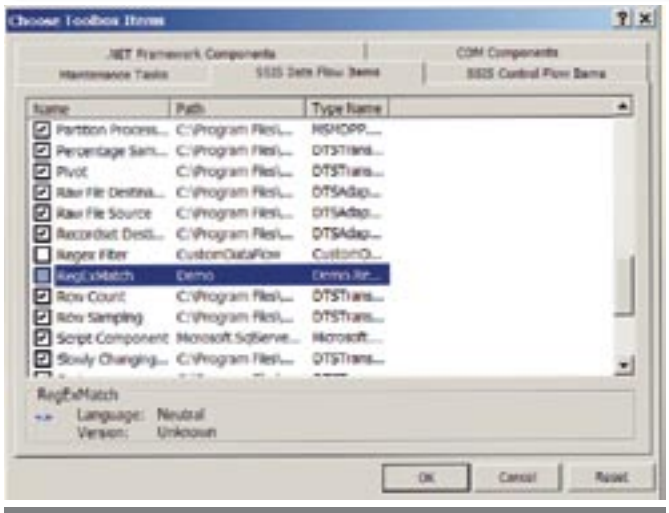
Afbeelding 4. item daadwerkelijk beschikbaar in de toolbox.

voegen; zie afbeelding 2. Dit net gecreëerde object regexFilter kan worden teruggevonden onder het tabblad 'SSIS Data Flow Items' zoals weergegeven in afbeelding 3. Het aanvinken van de checkbox zorgt er voor dat hij ook daadwerkelijk in de toolbox beschikbaar komt; zie afbeelding 4.

Met het beschikbaar maken van de component in Visual Studio is het ook mogelijk de methoden uit de IDTSDesigntimeComponent90 te debuggen. Dit doen we door het starten van het Class library-project, waarin we de component hebben ontwikkeld en de debugger aan het devenv.exe-proces hebben gekoppeld waarin het Integration Services-project draait. Wanneer we nu in het Integration Service-project een breakpoint uit de pipeline component-class raken, komen we automatisch in de debugger terecht. Waar je tijdens het debuggen van de IDTSDesigntimeComponent90-interface wel op moet letten is dat Visual Studio de component cached. Dit betekent dat je de Visual Studio-instantie waarin het Integration Service-project draait, opnieuw moet starten als een codewijziging plaatsvindt. Doe je dit niet, dan wil de debugger wel attachen, maar gaat de debugger zich onvoorspelbaar gedragen. Methoden uit de interface IDTSRuntimeComponent90 kun je debuggen door de component in een SSIS-package op te nemen en het proces dat dit SSIS-package uitvoert in de debugger te attachen. Dit gaat het eenvoudigst door de DTSExec-utility als laatste stap van het post build-event aan te roepen. De DTSExec-tool is een commandline-utility die het uit te voeren SSIS-package als parameter accepteert. Door het aanroepen van deze tool als post build-event op te nemen, wordt de debugger automatisch aan het proces gekoppeld en wordt er bij het bereiken van een breakpoint automatisch naar Visual Studio terug gesprongen.

```
"C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\gacutil.exe"
/u "$(TargetName)"
"C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\gacutil.exe"
/i "$(TargetDir)$(TargetFileName)"

copy /y "$(TargetDir)$(TargetFileName)" "C:\Program Files\Microsoft SQ
Server\90\DTS\PipelineCom ponents"
```

Codevoorbeeld 8.



Afbeelding 3. Tabblad 'SSIS Data Flow Items'.

Een beetje werk, maar ...

Bij het schrijven van een custom data flow-component komt heel wat kijken.

- Het implementeren van de interface IDTSDesigntimeComponent90 voor de designer;
- Het gebruiken van de methoden van het object ComponentMetaData, zodat configuratiedata worden opgeslagen;
- Het implementeren van de interface IDTSRuntimeComponent90, zodat de SSIS-runtime de component kan aanroepen;
- Het signen van de code, zodat de component kan worden gedeployed en gebruikt.

Maar als je dit eenmaal hebt gedaan, heb je ook echt iets. Een component die snel en eenvoudig kan worden hergebruikt, is op termijn zeker de moeite waard.
Tot slot nog één advies: In de eerste versie van de SQL Server 2005 books online is de documentatie over het bouwen van custom data flow-components summier en bij vlagen zelfs onjuist. In latere versies van de books online is dit gecorrigeerd en is de documentatie bovendien behoorlijk uitgebreid. Zorg er bij het ontwikkelen van custom data flow-components dus voor dat je altijd over de recentste versie van de SQL Server 2005 books online beschikt.

Leon Krancher is als consultant en SQL Server-specialist werkzaam bij Avanade (www.avanade.com), een samenwerkingsverband van Microsoft en Accenture. Voor vragen en opmerkingen is hij te bereiken op leonk@avanade.com.

Referenties

De laatste versie van de SQL Server 2005 books online http://www.microsoft.com/technet/prodtechnol/sql/2005/downloads/books.mspx
Online documentatie van Microsoft over het ontwikkelen van een custom data flow-component: http://msdn2.microsoft.com/en-us/library/ms136078.aspx
Specifieke documentatie voor het ontwikkelen van een userinterface voor een custom data flow-component: http://msdn2.microsoft.com/en-us/library/ms136029.aspx