Eenvoudig code genereren met T4 templates

David Stroobants

Text Template Transformation Toolkit, kortweg T4, is een toolkit waarmee je code kunt genereren vanuit Visual Studio 2008 en is een alternatief voor codegeneratoren als CodeSmith, MyGeneration, Nvelocity en dergelijke. Deze toolkit is inbegrepen bij Visual Studio 2008, maar er zijn slechts weinig ontwikkelaars die deze kennen, voornamelijk omdat de Add New Project Item dialog geen specifiek item bevat voor T4 templates.

In dit artikel leg ik aan de hand van een eenvoudig en bruikbaar voorbeeld uit, hoe u deze toolkit kunt gebruiken tijdens uw dagdagelijkse activiteiten als ontwikkelaar. U zult verbaasd zijn van de mogelijkheden én de eenvoud van deze toolkit. Recent vroeg een collega mij om 'iets' te maken waardoor we gemakkelijk enumeraties zouden kunnen genereren op basis van waarden in de codetabellen van onze databank. Hierdoor zouden we businesscode als ...

```
if (Customer.Country == 1)
```

... kunnen schrijven als ...

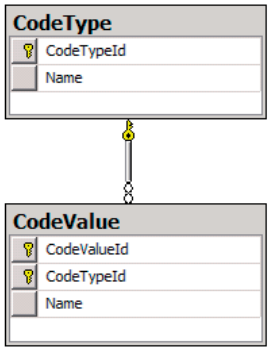
```
if (Customer.Country == Country.Belgium)
```

... waardoor de code veel beter leesbaar en onderhoudbaar wordt.

Deze enumeraties handmatig uitschrijven is geen leuk werk, zeker als er veel codetabellen en waarden moeten worden overgenomen. Ook deze enumeraties onderhouden en telkens afstemmen met de waarden uit de databank is iets dat je liever niet manueel wil doen. Deze enumeraties laten genereren is dus een aangewezen oplossing voor dit probleem.

Stap 1: Definieer de code tabellen

Gemakshalve definieer ik twee tabellen. Een eerste tabel 'CodeType' and andere tabel 'CodeValue' met de volgende layout:



... en de volgende records:

	CodeTypeId	Name
1	1	Country
2	2	Status

	CodeValueId	CodeTypeId	Name
1	1	1	Belgium
2	2	1	Netherlands
3	3	1	Germany
4	4	1	France
5	5	1	Italy
6	1	2	New
7	2	2	Revision
8	3	2	Approved

Met het T-SQL script in Codevoorbeeld 1 kunt u deze tabellen aanmaken op SQL Server.

```
USE Master
GO
CREATE DATABASE T4Samples
GO
USE T4Samples
GO

CREATE Table CodeType
(
  CodeTypeId INTEGER NOT NULL,
  Name  VARCHAR(50) NOT NULL,
  CONSTRAINT PK_CodeType PRIMARY KEY CLUSTERED
  (
    CodeTypeId ASC
  )
)
GO

CREATE Table CodeValue
(
  CodeValueIdINTEGER NOT NULL,
  CodeTypeId INTEGER NOT NULL,
  Name  VARCHAR(50) NOT NULL,
  CONSTRAINT PK_CodeValue PRIMARY KEY CLUSTERED
  (
    CodeValueId ASC,
    CodeTypeId ASC
  )
)
```

```
GO

ALTER TABLE CodeValue WITH CHECK
ADD CONSTRAINT FK_CodeValue_CodeType
FOREIGN KEY (CodeTypeId)
REFERENCES CodeType (CodeTypeId)
GO

ALTER TABLE CodeValue CHECK CONSTRAINT FK_CodeValue_CodeType
GO

INSERT INTO CodeType VALUES (1, 'Country')
INSERT INTO CodeType VALUES (2, 'Status')
GO

INSERT INTO CodeValue VALUES (1, 1, 'Belgium')
INSERT INTO CodeValue VALUES (2, 1, 'Netherlands')
INSERT INTO CodeValue VALUES (3, 1, 'Germany')
INSERT INTO CodeValue VALUES (4, 1, 'France')
INSERT INTO CodeValue VALUES (5, 1, 'Italy')
GO

INSERT INTO CodeValue VALUES (1, 2, 'New')
INSERT INTO CodeValue VALUES (2, 2, 'Revision')
INSERT INTO CodeValue VALUES (3, 2, 'Approved')
GO

SELECT * FROM CodeType
SELECT * FROM CodeValue ORDER BY CodeTypeId, CodeValueId
GO
```

Uw applicatie gebruikt wellicht een andere benadering met betrekking tot codetabellen, maar dat is op zich geen probleem. Dit artikel heeft niet de bedoeling om de ideale werkwijze met betrekking tot codetabellen te demonstreren, want iedere ontwikkelaar heeft hierover wel een eigen idee. U zult echter merken dat de codegeneratie eenvoudig kan worden aangepast naar uw persoonlijke situatie.

Stap 2: Query om codewaarden op te halen

Met behulp van de query uit codevoorbeeld 2 kunt u op basis van de naam van het codetype de nodige waarden voor de enumeratie ophalen. Ik ga er van uit dat de meeste ontwikkelaars deze query wel kunnen lezen en begrijpen.

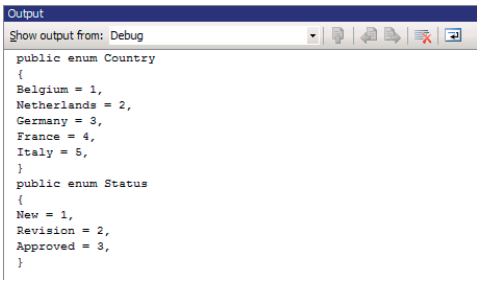
```
SELECT
  CodeValue.CodeValueId,
  CodeValue.Name
 FROM CodeValue
INNER JOIN CodeType
   ON CodeValue.CodeTypeId = CodeType.CodeTypeId
WHERE CodeType.Name = 'Country'
ORDER BY CodeValue.CodeValueID
```

Stap 3: .NET code die de query gebruikt

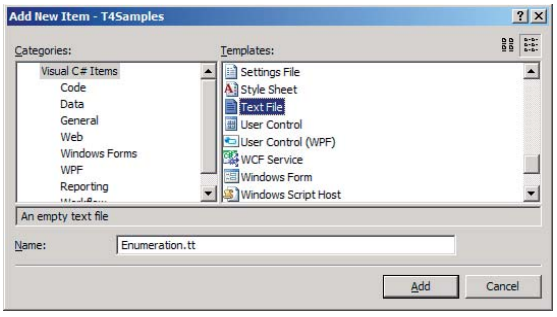
Ook de code uit codevoorbeeld 3 is wellicht door iedere C# ontwikkelaar te verstaan. De naam van ieder codetype waarvoor we een enumeratie wensen te creëren voegen we toe aan een lijst van strings. Daarna gebruiken we de eerder gemaakte query om een DataSet te creëren die de waarden zal bevatten van elk van de codetabellen. Ik weet het, ik had even eenvoudig gebruik kunnen maken van een stored procedure, een DataReader of LINQ, maar ik wilde deze demo bewust eenvoudig houden. Welke technologie u gebruikt om gegevens uit een databank te halen, is uiteindelijk ook van ondergeschikt belang voor dit voorbeeld.

```
string dbConnection = @"Integrated Security=SSPI;Initial Catalog=
T4Samples;Data Source=localhost";
System.Data.SqlClient.SqlConnection sqlConnection = new System.
Data.SqlClient.SqlConnection(dbConnection);

System.Collections.Generic.List<string> codeTypeNames = new List
<string>();
codeTypeNames.Add("Country");
codeTypeNames.Add("Status");

foreach (string codeTypeName in codeTypeNames)
{
    string query = "SELECT CodeValue.CodeValueId, CodeValue.Name " +
                   "FROM CodeValue " +
                   "INNER JOIN CodeType ON CodeValue.CodeTypeId =
                   CodeType.CodeTypeId " +
                   "WHERE CodeType.Name = @CodeTypeName " +
                   "ORDER BY CodeValue.CodeValueId";

    System.Data.DataSet dataSet = new System.Data.DataSet();
    System.Data.SqlClient.SqlCommand sqlCommand = new System.Data.
    SqlClient.SqlCommand(query, sqlConnection);
    System.Data.SqlClient.SqlParameter sqlParameter = sqlCommand.
    Parameters.Add("@CodeTypeName", SqlDbType.NVarChar);
    sqlParameter.Value = codeTypeName;
    System.Data.SqlClient.SqlDataAdapter sqlDataAdapter = new
    System.Data.SqlClient.SqlDataAdapter(sqlCommand);
    sqlDataAdapter.Fill(dataSet, "CodeValue");

    System.Diagnostics.Debug.WriteLine("public enum " + codeType-
    Name);
    System.Diagnostics.Debug.WriteLine("{");

    foreach (System.Data.DataRow dataRow in dataSet.Tables
    ["CodeValue"].Rows)
    {
        System.Diagnostics.Debug.WriteLine(dataRow["Name"].
        ToString() + " =
" + dataRow["CodeValueId"].ToString() + ",");
    }

    System.Diagnostics.Debug.WriteLine("}");
}
```

Door het runnen van deze code, zult u zien dat in het Output Window van Visual Studio, de gewenste C# code wordt uitgeschreven door de System.Diagnostics.Debug.WriteLine statements.



In plaats van naar het Output Window te schrijven, hadden we even goed de output kunnen wegschrijven naar een *.cs bestand, deze includen in een project en einde verhaal. Maar door deze code om te zetten naar een T4 template, wordt het allemaal een stuk interessanter.

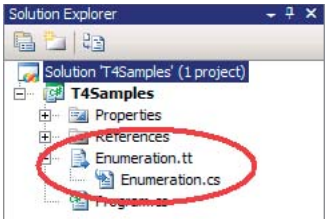


Stap 4: Maak een T4 template file

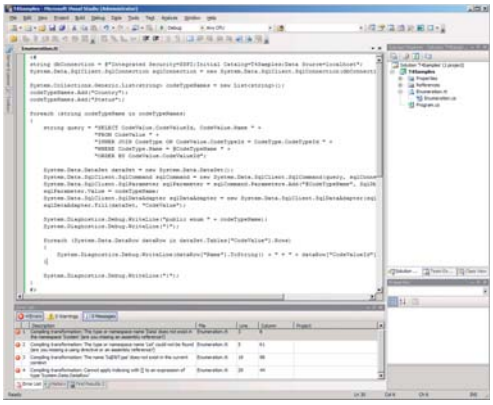
Het zou me verbazen mocht u in de voorgaande stappen iets nieuws hebben geleerd. En dat is eigenlijk ook het punt. Om T4 templates te kunnen maken en gebruiken, heeft u genoeg aan de tools en de kennis die u vandaag de dag als developer reeds bezit. Hoe gaat u nu te werk?

Maak een C# project aan naar eigen keuze. Klik rechts op het project en selecteer 'Add - New Item'. Kies 'Text File' en noem dit bestand 'Enumeration.tt'.

Wees er zeker van dat u het bestand de extensie '.tt' geeft en niet '.txt'! U zult zien dat Visual Studio dit bestandstype herkent en aan dit bestand een C# codebestand linkt met dezelfde naam.



Om de .NET code van stap 3 te gebruiken als T4 template, verplaats je deze code naar de TT-file - niet in het daaraan gekoppelde .cs-bestand! - en plaats je de tags, <# en #>, rondom de code. Met deze tags definieer je een statement block. Dit wordt gebruikt om controlecode aan te duiden voor de codegeneratie, waarover straks meer. Wanneer je nu de TT-file tracht te bewaren, krijg je compilatiefouten en dat is ook normaal.



Net als met 'gewone code' heb je genoeg aan enkele referenties naar assemblies om de code te kunnen laten werken. Door de volgende richtlijnen op te nemen in de template kan je dit verhelpen:

```
<#@ assembly name="System.Data.dll" #>
<#@ assembly name="System.XML.dll" #>
```

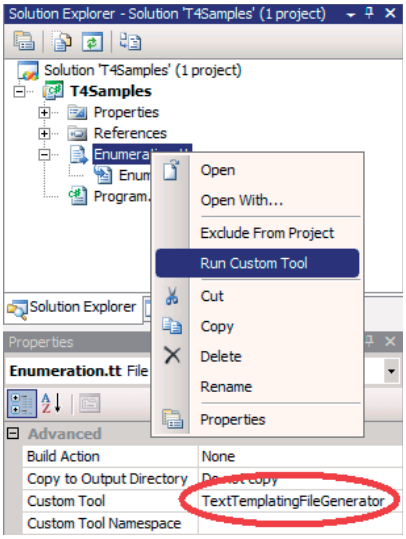
Vergelijkbaar met een 'using' statement in C# of een 'imports' statement in VB.NET, dien je ook nog enkele namespaces te specifiëren. Dit doe je door bijkomend volgende richtlijnen op te nemen bovenaan de template:

```
<#@ import namespace="System.Data" #>
<#@ import namespace="System.Collections.Generic" #>
```

Wanneer je de TT-file nu bewaart, zul je zien dat alles compileert én werkt. Alleen bevat het Enumeration.cs bestand dat gelinkt is aan de TT-file nog steeds geen code. Het enige wat je hiervoor nog moet doen, is stop en start statement block tags plaatsen rond de code waar er momenteel wordt weggeschreven naar het Output Window en die code vervangen door letterlijke tekst.
Alle tekst die buiten een statement block staat zal namelijk aanzien worden als tekst die dient te worden weggeschreven naar het gelinkte Enumeration.cs bestand. Wanneer je de waarde van een variabele moet wegschrijven, kunt u gebruik maken van een <#= en #> tag. Deze tags vormen zogenaamde expression blocks. De finale code van uw template ziet er nu uit als in codevoorbeeld 5:

```
<#@ assembly name = "System.Data.dll" #>
<#@ assembly name = "System.XML.dll" #>
<#@ import namespace = "System.Data" #>
<#@ import namespace = "System.Collections.Generic" #>
<#
string dbConnection = @"Integrated Security=SSPI;Initial
Catalog=T4Samples;Data Source=localhost";
System.Data.SqlClient.SqlConnection sqlConnection = new System.
Data.SqlClient.SqlConnection(dbConnection);

System.Collections.Generic.List<string> codeTypeNames = new
List<string>();
codeTypeNames.Add("Country");
codeTypeNames.Add("Status");

foreach (string codeTypeName in codeTypeNames)
{
    string query = "SELECT CodeValue.CodeValueId, CodeValue.Name " +
                   "FROM CodeValue " +
                   "INNER JOIN CodeType ON CodeValue.CodeTypeId =
                   CodeType.CodeTypeId " +
                   "WHERE CodeType.Name = @CodeTypeName " +
                   "ORDER BY CodeValue.CodeValueId";

    System.Data.DataSet dataSet = new System.Data.DataSet();
    System.Data.SqlClient.SqlCommand sqlCommand = new System.Data.
    SqlClient.SqlCommand(query, sqlConnection);
    System.Data.SqlClient.SqlParameter sqlParameter = sqlCommand.
    Parameters.Add("@CodeTypeName", SqlDbType.NVarChar);
    sqlParameter.Value = codeTypeName;
    System.Data.SqlClient.SqlDataAdapter sqlDataAdapter = new
    System.Data.SqlClient.SqlDataAdapter(sqlCommand);
    sqlDataAdapter.Fill(dataSet, "CodeValue");
#>
    public enum <#=codeTypeName#>
    {
<#
    foreach (System.Data.DataRow dataRow in dataSet.
    Tables["CodeValue"].Rows)
    {
#>
        <#=dataRow["Name"].ToString()#> = <#=dataRow["CodeValueId"].
        ToString()#>,
<#
    }
#>
    }
<#
}
#>
```

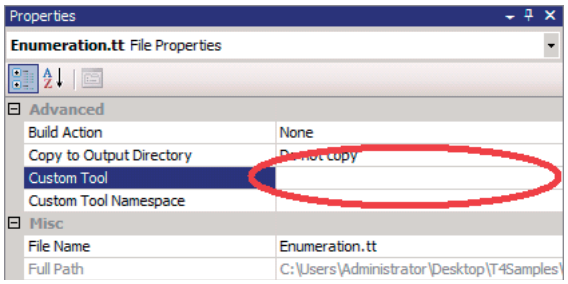
Wanneer je deze template bewaart, zal je zien dat het codebestand, dat gelinkt is aan uw T4 template, de enumeraties bevat die werden opgehaald vanuit de databank. Voortaan kan je dus de enumeraties genereren rechtstreeks vanuit uw Visual Studio development omgeving, door simpelweg de TT-file te bewaren of door de Custom Tool 'TextTemplatingFileGenerator' te runnen.



Ook wanneer uw code in Source Control zit (TFS, SourceSafe, etc.) worden beide bestanden automatisch uitgechecked op het moment van code generatie. Maar, er is nog ruimte voor verbetering ...

Stap 5: Optimaliseer uw template

Om de template te kunnen delen tussen verschillende projecten en het gebruik van parameters toe te laten, kunt u nog enkele verbeteringen aanbrengen.
Selecteer de TT-file en in het Properties Window verwijdert u de Custom Tool die default op 'TextTemplatingFileGenerator' staat.



Door deze Custom Tool te verwijderen verhinder je dat het gelinkte C# bestand met de enumeraties wordt aangemaakt. De TT-file wordt nu aangezien als een gewoon tekstbestand. Vervolgens plaats je de definitie van de connection string variabele en de lijst met codetypenamen onderaan de template en plaats je er <#+ en #> tags omheen. Hiermee definieer je een zogenaamd class feature block. Daarna voeg je een tweede TT-file toe aan het project met de lijnen uit het volgende codevoorbeeld.

```
<#
  dbConnection = @"Integrated Security=SSPI;Initial Catalog=
  T4Samples;Data Source=localhost";
  codeTypeNames.Add("Country");
  codeTypeNames.Add("Status");
#>
<#@ include file="Enumeration.tt" #>
```

Wanneer je het tweede TT-bestand nu bewaart, zul je zien dat de enumeraties gegenereerd worden zoals voorheen. Alleen is nu de eerste template herbruikbaar geworden tussen verschillende projecten en kunnen de codetypenamen en connectionstring worden doorgegeven als parameter.

Interessante weetjes

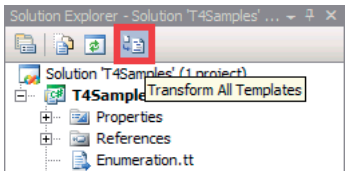
Het bestand dat gelinkt is aan uw TT-file heeft standaard de extensie '.cs' gekregen. Indien u graag een andere extensie toekent om een ander bestandstype te creëren, zoals '.vb', kunt u dit verwezenlijken door volgende richtlijn op te nemen in de template:

```
<#@ output extension=".vb" #>
```

Dit stelt u in staat om niet alleen C# code te genereren, maar evenzeer Visual Basic code, T-SQL, XML of elk ander type van tekstueel bestand.
De templatecode zelf was geschreven in C#. Indien u liever uw templatecode zou baseren op Visual Basic code, kunt u volgende richtlijn opnemen in uw template:

```
<#@ template language = "VB" #>
```

Wanneer een project meerdere T4 templates bevat en u wenst deze allemaal achter elkaar te runnen, dan kunt u gebruik maken van de knop 'Transform All Templates' die beschikbaar is in het Solution Explorer Window.



Verder dient u ook te weten dat u in plaats van stop en start tags te gebruiken, om letterlijke waarden weg te schrijven, u evenzeer gebruik kunt maken van templatecode als in codevoorbeeld 8.

```
<#
    PushIndent("\t");
    this.WriteLine("This will be written on a single line");
    this.Write("This will be written ... ");
    this.Write("on the same line as this.");
    PopIndent();
#>
```

Het had dus al voldoende geweest om in het originele voorbeeld de System.Diagnostics.Debug.WriteLine statements simpelweg te vervangen door this.WriteLine statements.

T4 onder de motorkap

Nu u weet wat T4 templates zijn en wat u er allemaal zoal mee kunt doen, zal ik de twee stappen toelichten die gebeuren tijdens het codegeneratieproces. Schermafbeelding 11 verduidelijkt dit.

In de eerste stap gaat de T4 engine de template compileren. De processing instructies worden geparsed evenals de tekst en codeblokken. Er wordt een concrete TextTransformation class gemaakt, die vervolgens wordt gecompileerd tot een .NET assembly. De bestanden die in deze fase worden aangemaakt kunt u terugvinden onder %USERPROFILE%\Local Settings\Temp.
In de tweede fase gaat de T4 engine een instantie maken van de GeneratedTextTransformation class. De TransformText methode wordt aangeroepen en het resultaat hiervan, een string, wordt bewaard en weggeschreven naar de output file.
Fouten die voorvallen tijdens de eerste fase van het generatie proces worden zichtbaar als compilatiefouten in Visual Studio, zoals we reeds gezien hebben in het begin van dit artikel. Fouten die voorvallen tijdens de tweede stap van het proces zullen resulteren in runtime errors.

David Stroobants, is werkzaam als .Net Solutions Architect bij Euricom te Mechelen (België). Hij is te bereiken via david.stroobants@euri.com en via de blog op http://www.euri.com.