Denne guide er oprindeligt
udgivet på Eksperten.dk

COMPUTERWORLD

CodeDOM - Dynamisk generering og kompilering af kode

Med CodeDOM kan man på runtime generere kode, kompilere det og derefter afvikle det. I denne artikel skal vi kigge på hvordan...

Skrevet den **01. Mar 2009** af **wisen** I kategorien **Programmering / .NET** | ★★☆☆☆

Indledning
Alle der har arbejdet en smule med .NET platformen har næsten med sikkerhed hørt om WebServices og med næsten lige så stor sikkerhed prøvet at lave en WebService og benytte den i en eller anden klientapplikation. For at benytte en WebService i .NET har man brug for en proxy klasse som man fra klientsiden kan instantiere og benytte som enhvert andet objekt - altså kalde dens metoder. Men hvordan kommer man fra at man trykket på "Add Web Reference" i Visual Studio, til der er genereret en proxyklasse?

For at få genereret en proxyklasse skal man som bekendt angive hvilken Webservice man ønsker at konsumere ved at angive en URL til en .WSDL fil. Visual Studio kalder kommandolinie værktøjet WSDL.exe, som genererer proxyklassen, med den angivne .WSDL fil som argument. Ved at dissassemble WSDL.exe klassen kan man se at den benytter klassen System.Web.Services.Description.ServiceDescriptionImporter til at generere selve koden.

MSDN har følgende eksempel på at benytte ServiceDescriptionImporter

```
using System;
using System.Web.Services.Description;
using System.CodeDom;

namespace MyServiceDescription
{
class MyImporter
{
  public static void Main()
  {
   try
   {
    ServiceDescription myServiceDescription =
ServiceDescription.Read("Sample_CS.wsdl");
    ServiceDescriptionImporter myImporter = new ServiceDescriptionImporter();
    myImporter.ProtocolName = "Soap";
    myImporter.AddServiceDescription(myServiceDescription, "", "");
    Console.WriteLine("Style: " + myImporter.Style.ToString());
    CodeNamespace myNamespace = new
CodeNamespace(ServiceDescription.Namespace);
    CodeCompileUnit myUnit = new CodeCompileUnit();
    myUnit.Namespaces.Add(myNamespace);

    ServiceDescriptionImportWarnings myWarning =
myImporter.Import(myNamespace, myUnit);
    Console.WriteLine("ServiceDescriptionImportWarnings value generated: " +
myWarning.ToString());
```

```
    // Get the XMLSchema related to the ServiceDescriptionImporter.
    System.Xml.Serialization.XmlSchemas mySchema = myImporter.Schemas;
   }
   catch (Exception e)
   {
    Console.WriteLine("Following exception was thrown: " + e.ToString());
   }
  }
}
}
```

Langt det meste arbejde med at importere filen "Sample_CS.wsdl" sker i forbindelse med at Import metoden bliver kaldt på myImporter. Metoden tager, som det kan ses 2 argumenter - et CodeNamespace (myNamespace) og et CodeCompileUnit (myUnit). Disse klasser er en del af CodeDOM. Vi kommer tilbage til hvad de kan bruges til, og selvfølgelig hvordan man bruger dem.

Formålet med artiklen
Formålet med denne artikel er at vise et eksempel på hvordan CodeDOM teknologien kan benyttes. Som altid når man begynder at kigge på en ny teknologi er der nærmest en konvention om at man laver et Hello World eksempel - så det skal vi selvfølgelig også her :)

Hvad er CodeDOM så for en størrelse?
CodeDOM teknologien kan f.eks. bruges til at lave kodegeneratorer som WSDL.exe som beskrevet i indledningen, en anden mulighed er at generere en assembly på runtime som varetager en eller anden, typisk krævende, opgave. Faktisk genererer XMLSerializeren en assembly på runtime i forbindelse med at man benytter WebServices - har I lagt mærke til at det tager 1-2 sekunder at instantiere en Webservice-proxy første gang ?

CodeDOM står er en forkortelse af "Code Document Object Model". CodeDOM'en er en træstruktur, en graf, der opbygges af et antal elementer fra CodeDOM-namespacet. CodeDOM'en er en abstraktion af et programmeringssprog - og dette er den primære fordel - man kan opbygge een CodeDOM som derefter kan realiseres i et, eller flere, konkrete programmeringssprog. CodeDOM namespacet indeholder klasser til at repræsentere langt de fleste kodemæssige konstruktioner man ellers kan lave ved at kode direkte i f.eks. C#. Der bl.a. findes klasser til at repræsentere type deklarationer, såsom klasser, member-variable (fields), metoder, properties, tildelinger, if-sætninger osv. osv. osv.

Fordele ved CodeDOM
- CodeDOM giver mulighed for at kode kan genereres, kompileres og afvikles på runtime.
- CodeDOM giver mulighed for at generere kode for (principielt) alle .NET sprog. Microsoft understøtter umiddelbart C#, VB.NET og JScript. Der findes også så vidt jeg ved også en CodeGenerator der kan emitte Delphi kode

Begrænsninger ved CodeDOM
- Ikke alle konstruktioner er mulige via CodeDOM. På MSDN står der "The CodeDOM supports the many common types of code elements found in programming languages that support the common language runtime. The CodeDOM was not designed to provide elements to represent all possible programming language features."

Der findes nogle specielle klasser, f.eks. CodeSnippetStatement, som man kan benytte hvis man støder på en begrænsning ved CodeDOM'en. Via CodeSnippetXXX klasserne har man mulighed for at sætte kode direkte ind i outputtet:

```
CodeSnippetStatement literalStatement = new CodeSnippetStatement("Console.Write(\"Test literal statement output\")");
```

Hvis man putter ovenstående CodeSnippetStatement ind i sin CodeDOM vil man når CodeDOM'en emittes få skrevet "Console.Write(\"Test literal statement output\")". Dette har, som det sikkert kan ses, den ulempe at det ikke mere er muligt at tage en CodeDOM og emitte den til ethvert .NET sprog. Da CodeSnippets er sprogspecifikke skal man, når man genererer CodeDOM'en, tage højde for hvilket sprog der skal outputtes til.

Namespaces
Der er specielt to namespaces man skal benytte når man vil arbejde med CodeDOM:

```
- System.CodeDOM - indeholder de elementer der kan benyttes til at opbygge DOM'en
- System.CodeDOM.Compiler - definerer bl.a. interfaces for generering og kompilering af kode
```

Man benytter klassser fra System.CodeDOM til at opbygge strukturen, altså DOM'en og man benytter funktionalitet fra System.CodeDOM.Compiler namespacet for at outputte og kompilere kode udfra denne DOM.

I System.CodeDOM.Compiler namespacet er der defineret to interfaces:

```
- ICodeGenerator - definerer interfacet for klasser der kan generere kode udfra en CodeDOM
- ICodeCompiler - definerer interfacet for klasser der kan benyttes til at kompilere kode
```

I namespacet Microsoft.CSharp henholdsvis Microsoft.VisualBasic findes der en klasse (CSharpCodeProvider henholdsvis VBCodeProvider) der fungerer som factory af ICodeGenerator og ICodeCompiler implementationer for de respektive sprog.

Hello world
Målet med dette eksempel har været at komme omkring de fleste "almindelige" konstruktioner. Eksemplet går ud på at lave en forholdsvis simpel Hello World klasse som har en metode til at vise en messagebox, derudover har man mulighed for at ændre den besked den skal vise. Jeg har forsøgt at sætte relevante kommentarer ind for at koden bliver så forståelig som mulig - håber at det virker ;)

Eksemplet kommer ind over klassedefinitioner, membervariable, properties, metoder, exceptions, tildeling af variable og properties osv.

Instantiér et CodeCompileUnit
Første trin er at lave et CodeCompileUnit:

```
private static CodeCompileUnit CreateCompileUnit()
{
// Create CompileUnit
CodeCompileUnit compileUnit = new CodeCompileUnit();

// Add references assemblies
compileUnit.ReferencedAssemblies.Add("System");
```

```
compileUnit.ReferencedAssemblies.Add("System.Windows.Forms");

return compileUnit;
}
```

Tilføj et namespace
Dernæst skal man tilføje et namespace vi kan putte klasser osv. ind i. Dette namespace skal tilføjes til det compileUnit vi skabte ovenfor.

```
private static CodeNamespace CreateNamespace( CodeCompileUnit compileUnit )
{
// Create namespace
CodeNamespace codeNamespace = new CodeNamespace("HelloWorld");

// Add "using" statements
codeNamespace.Imports.Add( new CodeNamespaceImport("System") );
codeNamespace.Imports.Add( new CodeNamespaceImport("System.Windows.Forms") );

// Add namespace to compileUnit
compileUnit.Namespaces.Add( codeNamespace );

return codeNamespace;
}
```

Definér en klasse
Derefter kan vi definere en klasse "HelloWorldClass" som vi kan tilføje til namespacet:

```
private static CodeTypeDeclaration CreateHelloWorldClass( CodeNamespace
codeNamespace )
{
// Create MyHelloWorldClass
CodeTypeDeclaration classDeclaration = new
CodeTypeDeclaration("HelloWorldClass");
classDeclaration.IsClass = true;
classDeclaration.Attributes = MemberAttributes.Public;

// Add class to namespace
codeNamespace.Types.Add( classDeclaration );

return classDeclaration;
}
```

Tilføj en membervariabel
Membervariable defineres ved at skabe instanser at CodeMemberField:

```
private static void CreateFields(CodeTypeDeclaration classDeclaration)
{
```

```
// Create private field "m_message"
CodeMemberField field = new CodeMemberField(typeof(string), "m_message");
field.Attributes = MemberAttributes.Private;
field.InitExpression = new CodePrimitiveExpression("Hello world");

// Add field to class
classDeclaration.Members.Add( field );
}
```

Tilføj en konstruktor
En konstruktor defineres ved at skabe instanser af CodeConstructor. På denne instans kan der tilføjes kode i form af CodeStatements/CodeExpressions.

```
private static void CreateConstructor(CodeTypeDeclaration classDeclaration)
{
// Create constructor with parameters...
CodeConstructor constructor = new CodeConstructor();
constructor.Attributes = MemberAttributes.Public | MemberAttributes.Final;

// ... add a string parameter "message"
constructor.Parameters.Add( new
CodeParameterDeclarationExpression(typeof(string), "message"));

// .. if argument is null then return
constructor.Statements.Add( new CodeConditionStatement(
  new CodeBinaryOperatorExpression( new
CodeArgumentReferenceExpression("message"),
    CodeBinaryOperatorType.ValueEquality, new CodePrimitiveExpression(null)),
    new CodeStatement[] { new CodeMethodReturnStatement() } ));

// ... assign value of parameter "message" to field "m_message"
constructor.Statements.Add( new CodeAssignStatement(
     new CodeFieldReferenceExpression( new CodeThisReferenceExpression(),
"m_message" ),
     new CodeArgumentReferenceExpression("message")));

// ... add constructors to class
classDeclaration.Members.Add( constructor );
}
```

Tilføj en property
En property defineres ved at skabe en instans af CodeMemberProperty. En CodeMemberProperty har to CodeStatementCollections for henholdvis get og set delene, den har desuden to properties (HasGet og HasSet) der kan benyttes til at fortælle om den overholdet har en get henholdsvis set del - default er de begge true.

```
private static void CreateProperty(CodeTypeDeclaration classDeclaration)
{
// Create property to access private field...
CodeMemberProperty property = new CodeMemberProperty();
```

```
property.Name = "Message";
property.Type = new CodeTypeReference(typeof(string));
property.Attributes = MemberAttributes.Public | MemberAttributes.Final;

// ... add setter
property.SetStatements.Add( new CodeAssignStatement(
    new CodeFieldReferenceExpression( new CodeThisReferenceExpression(),
"m_message" ),
     new CodePropertySetValueReferenceExpression()));

// ... add getter
property.GetStatements.Add( new CodeMethodReturnStatement(
    new CodeFieldReferenceExpression( new CodeThisReferenceExpression(),
"m_message" ) ));

// ... add property to class
classDeclaration.Members.Add( property );
}
```

Tilføj en metode
Metoder defineres ved at skabe instanser af CodeMemberMethods:

```
private static void CreateMethods(CodeTypeDeclaration classDeclaration)
{
// Create method to show message...
CodeMemberMethod method = new CodeMemberMethod();
method.Name = "ShowMessage";
method.Attributes = MemberAttributes.Public | MemberAttributes.Final;

// ... if property is null throw exception
method.Statements.Add( new CodeConditionStatement(
  new CodeBinaryOperatorExpression( new CodePropertyReferenceExpression(
  new CodeThisReferenceExpression(), "Message"),
CodeBinaryOperatorType.ValueEquality,
  new CodePrimitiveExpression(null)), new CodeStatement[] {
  new CodeThrowExceptionStatement( new
CodeObjectCreateExpression(typeof(System.Exception),
new CodeExpression[] { new CodePrimitiveExpression("Message is null")})})}));

// ... else add call to System.Windows.Forms.MessageBox.Show with property
Message as argument
method.Statements.Add( new CodeMethodInvokeExpression(
new CodeTypeReferenceExpression( typeof(System.Windows.Forms.MessageBox)),
"Show",
new CodeExpression[] { new CodePropertyReferenceExpression(
  new CodeThisReferenceExpression(), "Message" )}));

// Create overloaded method to show message...
CodeMemberMethod overloadedMethod = new CodeMemberMethod();
overloadedMethod.Name = "ShowMessage";
overloadedMethod.Attributes = MemberAttributes.Public |
MemberAttributes.Final;
```

```
// ... add parameter
overloadedMethod.Parameters.Add(
  new CodeParameterDeclarationExpression(typeof(string), "message") );

// ... if property is null throw exception
overloadedMethod.Statements.Add( new CodeConditionStatement(
  new CodeBinaryOperatorExpression( new
CodeArgumentReferenceExpression("message"),
  CodeBinaryOperatorType.ValueEquality, new CodePrimitiveExpression(null)),
  new CodeStatement[] { new CodeThrowExceptionStatement(
  new CodeObjectCreateExpression(typeof(System.Exception),
  new CodeExpression[] { new CodePrimitiveExpression("Message is null")}))}});

// ... else add call to System.Windows.Forms.MessageBox.Show with parameter as
argument
overloadedMethod.Statements.Add( new CodeMethodInvokeExpression(
new CodeTypeReferenceExpression(typeof(System.Windows.Forms.MessageBox)),
"Show",
new CodeExpression[] { new CodeArgumentReferenceExpression("message")}));

// ... add method to class
classDeclaration.Members.Add( method );
classDeclaration.Members.Add( overloadedMethod );
}
```

Tilføj et EntryPoint
Som bekendt skal der i enhver assembly være et EntryPoint - sådan et defineres ved at skabe en instans af CodeEntryPointMethod.

```
private static void CreateEntryPoint( CodeTypeDeclaration classDeclaration )
{
// Create entry point
CodeEntryPointMethod entryPoint = new CodeEntryPointMethod();

// Add entry point to class
classDeclaration.Members.Add( entryPoint );

// Create a try-catch block
CodeTryCatchFinallyStatement tryCatch = new CodeTryCatchFinallyStatement();

// Add try-catch block to entrypoint
entryPoint.Statements.Add( tryCatch );

// Create instance of HelloWorldClass and assign it to a variable
"helloWorldClass"
tryCatch.TryStatements.Add(
  new CodeVariableDeclarationStatement("HelloWorldClass", "helloWorldClass",
  new CodeObjectCreateExpression("HelloWorldClass",
  new CodeExpression[] { new CodePrimitiveExpression(null) } )) );

// Invoke ShowMessage method of HelloWorldClass
```

```
tryCatch.TryStatements.Add( new CodeMethodInvokeExpression(
new CodeVariableReferenceExpression("helloWorldClass"), "ShowMessage",
new CodeExpression[0]) );

// Set property Message of HelloWorldClass - set to "Hej Verden"
tryCatch.TryStatements.Add( new CodeAssignStatement(
  new CodePropertyReferenceExpression(
   new CodeVariableReferenceExpression("helloWorldClass"), "Message"),
   new CodePrimitiveExpression("Hej Verden")) );

// Invoke ShowMessage method of HelloWorldClass
tryCatch.TryStatements.Add( new CodeMethodInvokeExpression(
new CodeVariableReferenceExpression("helloWorldClass"), "ShowMessage",
new CodeExpression[0]) );

// Invoke ShowMessage method of HelloWorldClass with argument "Hello World"
tryCatch.TryStatements.Add( new CodeMethodInvokeExpression(
  new CodeVariableReferenceExpression("helloWorldClass"), "ShowMessage",
  new CodeExpression[] {
   new CodePrimitiveExpression("Hello World") } ) );

// Create an catch clause...
CodeCatchClause catchClause = new CodeCatchClause("e",
    new CodeTypeReference(typeof(System.Exception)));
catchClause.Statements.Add( new CodeMethodInvokeExpression(
   new CodeTypeReferenceExpression(typeof(System.Windows.Forms.MessageBox)),
"Show",
   new CodeExpression[] {
    new CodePropertyReferenceExpression( new
CodeArgumentReferenceExpression("e"), "Message") } ));

// Add catch clause to try-catch block
tryCatch.CatchClauses.Add( catchClause );
}
```

Kodegenerering - ICodeGenerator
Som beskrevet ovenfor kan man via Microsoft.CSharp.CSharpCodeProvider få fat i en ICodeGenerator som kan emitte et CodeCompileUnit til en fil:

```
private static void GenerateCode(CodeCompileUnit compileUnit)
{
// Create code from compileUnit...
CodeDomProvider provider = new Microsoft.CSharp.CSharpCodeProvider();
ICodeGenerator codeGenerator =  provider.CreateGenerator();

CodeGeneratorOptions generatorOptions = new CodeGeneratorOptions();
generatorOptions.BracingStyle = "C";

IndentedTextWriter writer =
  new IndentedTextWriter( new System.IO.StreamWriter("HelloWorld.cs", false),
"    ");
codeGenerator.GenerateCodeFromCompileUnit( compileUnit, writer,
```

```
generatorOptions );
writer.Close();
}
```

Kompilering - ICodeCompiler
Ligesom ved generering af koden kan man via Microsoft.CSharp.CSharpCodeProvider få fat i en ICodeCompiler som kan kompilere den fil vi har genereret ovenfor:

```
private static void CompileCode()
{
// Compile code...
CodeDomProvider provider = new Microsoft.CSharp.CSharpCodeProvider();
ICodeCompiler codeCompiler = provider.CreateCompiler();

CompilerParameters compilerOptions = new CompilerParameters();

compilerOptions.ReferencedAssemblies.Add("System.dll");
compilerOptions.ReferencedAssemblies.Add("System.Windows.Forms.dll");

compilerOptions.GenerateInMemory = false;
compilerOptions.GenerateExecutable = true;
compilerOptions.OutputAssembly = "C:\\HelloWorld.exe";

CompilerResults compilerResults =
codeCompiler.CompileAssemblyFromFile(compilerOptions, "HelloWorld.cs");
}
```

Putting it all together
Så mangler vi bare at sætte det hele sammen:

```
public static CodeCompileUnit doStuff()
{
CodeCompileUnit compileUnit = CreateCompileUnit();
CodeNamespace codeNamespace = CreateNamespace( compileUnit );
CodeTypeDeclaration classDeclaration = CreateHelloWorldClass( codeNamespace );

CreateFields(classDeclaration);
CreateConstructor(classDeclaration);
CreateProperty(classDeclaration);
CreateMethods(classDeclaration);
CreateEntryPoint(classDeclaration);

GenerateCode( compileUnit );
CompileCode();
}
```

Resultatet

Herunder er resultatet fra genereringen:

```
namespace HelloWorld
{
    using System;
    using System.Windows.Forms;

    public class HelloWorldClass
    {
        private string m_message = "Hello world";

        public HelloWorldClass(string message)
        {
            if ((message == null))
            {
                return;
            }
            this.m_message = message;
        }

        public string Message
        {
            get
            {
                return this.m_message;
            }
            set
            {
                this.m_message = value;
            }
        }

        public void ShowMessage()
        {
            if ((this.Message == null))
            {
                throw new System.Exception("Message is null");
            }
            System.Windows.Forms.MessageBox.Show(this.Message);
        }

        public void ShowMessage(string message)
        {
            if ((message == null))
            {
                throw new System.Exception("Message is null");
            }
            System.Windows.Forms.MessageBox.Show(message);
        }

        public static void Main()
        {
            try
            {
```

```
                HelloWorldClass helloWorldClass = new HelloWorldClass(null);
                helloWorldClass.ShowMessage();
                helloWorldClass.Message = "Hej Verden";
                helloWorldClass.ShowMessage();
                helloWorldClass.ShowMessage("Hello World");
            }
            catch (System.Exception e)
            {
                System.Windows.Forms.MessageBox.Show(e.Message);
            }
        }
    }
}
```

Referencer
CodeDOM Quick Reference

Kommentar af burningice d. 03. Nov 2004 | 1

Kommentar af edutasia d. 20. Apr 2005 | 2

Kommentar af williamengbjerg d. 05. Nov 2004 | 3

sejt..