Olaf Conijn en Dennis Mulder

Nieuwe application blocks besparen werk

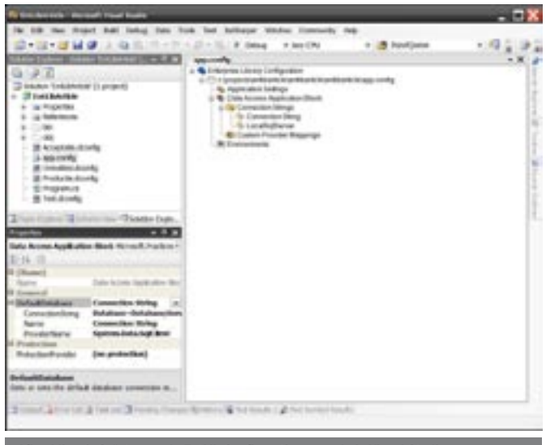
ENTERPRISE LIBRARY 3.0

Zoals we gewend zijn van Microsoft patterns & practices volgt de doorontwikkeling van Enterprise Library (EntLib) de roadmap van het .NET Framework nauwgezet. Microsoft is inmiddels toe aan de derde versie van EntLib. Het gebruik van EntLib is bij veel bedrijven inmiddels gemeengoed en in de communities worden talloze uitbreidingen ontwikkeld. Met dit artikel willen wij jullie inzicht geven in de nieuwe features van Enterprise Library voor .NET 3.0. We staan wat langer stil bij de twee nieuwe application blocks.

Enterprise Library is ontstaan uit het integreren van zeven application blocks, te weten Data Access, Exception Handling, Configuration, Caching, Logging & Instrumentation, Security en Cryptography. Een application block kan gezien worden als een herbruikbaar en uitbreidbaar stuk software dat helpt bij een veelvoorkomend ontwikkelscenario. De eerste release van EntLib bracht Microsoft uit in januari 2005. Een update volgde in juni 2005 met de komst van het .NET Framework-versie 2.0. In november 2005 is de patterns & practices group begonnen met het doorontwikkelen van EntLib om optimaal aan te sluiten op de nieuwe functionaliteiten die het nieuwe .NET Framework 3.0 biedt. In navolging op de lancering van .NET 3.0 in november 2007 is EntLib 3 beschikbaar sinds april 2007. Microsoft heeft, naast een betere aansluiting op het nieuwe framework, gekozen voor innovatie in de vorm van nieuwe application blocks. De structuur en implementatie van de core-componenten zijn in deze versie nauwelijks veranderd, hetgeen kopzorgen bespaart bij het migreren vanuit de laatst gereleasede versie.

Nieuwe functionaliteit

Naast de twee nieuwe application blocks, Validation en Policy Injection, is er voor de bestaande application blocks goed gekeken hoe je deze optimaal kunt laten aansluiten op .NET 3.0. Zo kan het Exception Handling Block geconfigureerd worden om 'exception shielding' te doen binnen WCF-services en in het Logging Block is het mogelijk audit-informatie uit WCF te loggen. Het Data Access Application Block heeft nu ondersteuning voor SQL Server Compact Edition. Zie hiervoor het artikel van Jacqueline van der Holst elders in dit blad. Er is ook een hoop feedback op de vorige versie verwerkt in deze release. Dit is merkbaar vanaf het moment van installeren. Je kunt nu apart kiezen voor de installatie van de source-code. Het is nu niet meer noodzakelijk zelf de code te compileren en indien nodig zelf te voorzien van een strong name. Voor gebruikers die EntLib willen installeren bij een externe hoster waar .NET-assemblies niet met de volledige rechten mogen draaien, is Partial Trust-support ingebouwd. Deze functionaliteit was al beschikbaar als patch voor EntLib 2.0, maar is nu een geïntegreerd onderdeel van EntLib 3.0, evenals de mogelijkheid om Enterprise Library-applicaties te configureren over group policy. Als laatste algemene wijziging is de Configuration Editor nu geïntegreerd in Visual Studio 2005, zoals te zien is in afbeelding 1.



Afbeelding 1. Geïntegreerde EntLib Configuration-console in Visual Studio

Environmental Overrides

Veel applicaties worden op verschillende omgevingen uitgerold, denk bijvoorbeeld aan een OTAP-straat (Ontwikkel, Test, Acceptatie en Productie). Het aantal omgevingen, waarvan de configuratie-instellingen allemaal verschillend zijn, groeit in grotere projecten sterk. EntLib maakt veel gebruik van configuratie-instellingen, het beheren en uitrollen van de verschillen in configuratie is vaak een foutgevoelige en tijdrovende bezigheid. EntLib 3.0 kent daarom een feature die Environmental Overrides heet. Dit is een uitbreiding die ook voor eerdere versies van EntLib beschikbaar was gemaakt op het internet. In EntLib 3.0 is het nu mogelijk in de configuratieconsole aan te geven dat je gebruik wilt maken van verschillende omgevingen, waarvoor de verschillen (in een delta-file) onafhankelijk kunnen worden onderhouden en met het origineel gecombineerd kunnen worden tot een nieuw configuratiebestand, specifiek voor een van de omgevingen. In afbeelding 2 is in één overzicht te zien hoe voor ontwikkel, test, acceptatie en productie een database-connectionstring anders ingesteld staat. Hierdoor kunnen beheerders, na het deployen van jouw oplossing, gevoelige informatie instellen in de delta-file; zodat jij als ontwikkelaar geen weet hoeft te hebben van deze informatie.
Er zijn twee manieren om tot een nieuw configuratiebestand te komen, bestaande uit het origineel en de delta:

- Vanuit de Configuration-console kun je de files mergen en opslaan.
- Via een command line-tool die het mogelijk maakt dit proces te automatiseren.

Bij het automatiseren van dit proces kun je bijvoorbeeld denken aan een buildserver die de verschillende configuratiebestanden maakt, alvorens voor iedere omgeving een eigen installer (dan wel zip-bestand) wordt samengesteld.

Application Block Software Factory

Wie zich heeft gewaagd aan het bouwen van eigen application blocks, weet dat dit een ietwat steile leercurve kan zijn. Om de ontwikkelaar meer houvast te geven bij het ontwikkelen van eigen application blocks is de Application Block Software Factory aan EntLib toegevoegd. Uiteindelijk maakt dit ook het uitwisselen van zelfgemaakte uitbreidingen gemakkelijker. Met het gebruik van de Software Factory weet je zeker dat het op de juiste manier met de rest integreert.
Feitelijk biedt de Application Block Software Factory je een wizard waarmee je verschillende taken kunt uitvoeren die komen kijken bij het ontwikkelen of uitbreiden van application blocks. Denk bijvoorbeeld aan:

- Het maken van een eigen provider voor een bestaand application block (bijvoorbeeld het toevoegen van een nieuw soort logging-technologie).
- Het integreren van een eigen configuratieformaat in de Configuration-console.
- Het ontwikkelen van een geheel nieuw application block.

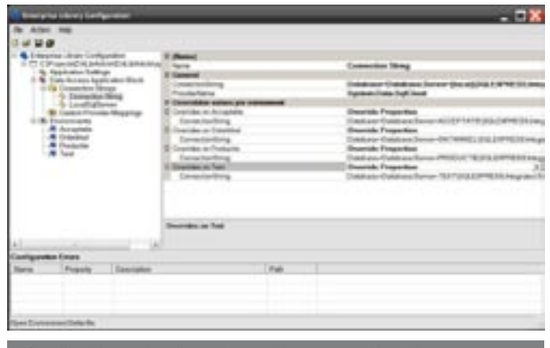
In afbeelding 3 kun je zien hoe de wizards uit de Software Factory worden aangeboden in de Visual Studio-omgeving. De werking van de Application Block Software Factory is vergelijkbaar met die van de andere Software Factories zoals we die van patterns & practices kennen, zoals de Web Service Software Factory, Smart Client Software Factory en de Web Client Software Factory.

Strong Naming Guidance Package

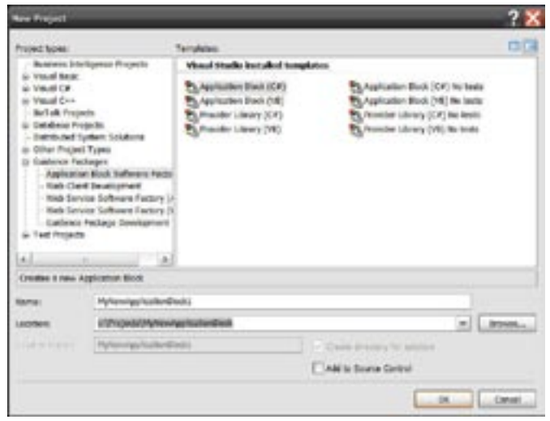
.NET-assemblies kunnen voorzien worden van een strong name. Nu zullen vele .NET-ontwikkelaars wel weten hoe je dit moet doen, maar er zijn ook velen onder ons die zich afvragen waarom je dit eigenlijk doet. Het voordeel van het signen van een .NET- assembly met een strong name is dat daardoor niemand anders de assembly kan vervangen door een aangepaste versie. Een .NET-assembly die voorzien is van een strong name bevat een publieke sleutel en een geëncrypte hash. De hash bevat een aantal belangrijke onderdelen van de assembly en wordt geëncrypt met de private sleutel die de ontwikkelaar van de assembly in zijn bezit heeft. De publieke sleutel wordt gebruikt om de geëncrypte hash te valideren. Het signen van een solution zoals EntLib met een kleine 50 projecten (exclusief tests) is een hoop werk. Voorheen moest je voor elk van deze projecten handmatig naar de 'private key' verwijzen. Toch wordt dit vaak gedaan in projecten en loont het de moeite om dit proces te automatiseren. Zowel het aanmaken van een nieuw publiek en privaat sleutelpaar als het opnemen van de informatie in de projecten die aangeven dat deze gesigned moeten worden, kunnen nu via de 'Strong Naming Guidance Package' gedaan worden.

Validation Application Block

Het validation application block stelt je in staat op een eenvoudige en eenduidige manier validatielogica op te nemen in jouw applicatie.



Afbeelding 2. Voorbeeld environmental overrides voor connection-strings



Afbeelding 3. Het aanmaken van een nieuw application block

De validatieregels kun je op drie verschillende smaken gebruiken in code: vanuit configuratie, via attributen en door deze zelf aan te roepen via de API. Vervolgens kun je deze validatieregels hergebruiken over de verschillende lagen van de applicatie. Of je dit nu vanuit de UI (winforms, webforms), businesslogica of services (WCF) wilt doen, er bestaan voor de verschillende technologieën componenten die de integratie doen. EntLib komt met een hele set aan voorgedefinieerde regels die gecombineerd kunnen worden tot nieuwe regels, dan wel

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Practices.EnterpriseLibrary.Validation.Validators;

namespace EntLibArticle
{
  public class Customer
  {
    private string firstName;
    private string lastName;
    private string email;

    [StringLengthValidator(1, 50, Ruleset = "RuleSetA", MessageTemplate
    = "Voornaam moet bestaan uit tussen de 1 en 50 tekens")]
    public string FirstName
    {
      get { return firstName; }
      set { firstName = value; }
    }

    [StringLengthValidator(1, 50, Ruleset = "RuleSetA", MessageTemplate
     = "Achternaam moet bestaan uit tussen de 1 en 50 tekens")]
    public string LastName
    {
      get { return lastName; }
      set { lastName = value; }
    }

    [RegexValidator(@"\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*",
MessageTemplate = "Ongeldig e-mailadres", Ruleset = "RuleSetA")]
    public string Email
    {
      get { return email; }
      set { email = value; }
    }
  }
}
```

Codevoorbeeld 1.

Afbeelding 4. Validatieregels configureren

uitgebreid kunnen worden met zelfontwikkelde regels. Enkele voorbeelden van meegeleverde validatieregels zijn: lengte (numeriek, text, datum/tijd), reguliere expressies, ranges (10-20), bevat karakters, not null en validaties of diverse type conversies mogelijk zijn.
In codevoorbeeld 1 is te zien hoe je een type voorziet van validatieregels door gebruik te maken van custom-attributes. In afbeelding 4 is te zien hoe je dezelfde regels in configuratie aanmaakt. Een groot voordeel van het aangeven in configuratie is dat je niet per se de beschikking hoeft te hebben over de source-code van het type en je de regels kunt aanpassen zonder opnieuw te compileren. Eventueel zijn de foutmeldingen op

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Text;
using Microsoft.Practices.EnterpriseLibrary.Data;
using Microsoft.Practices.EnterpriseLibrary.Validation;

namespace EntLibArticle
{
  class Program
  {
    static void Main(string[] args)
    {
      Customer customer = new Customer();
      ShowValidationResults(customer);
      customer.Email = "dennism@avanade.com";
      customer.LastName = "Mulder";
      customer.FirstName = "Dennis";
      ShowValidationResults(customer);

      Console.ReadLine();
    }

    private static void ShowValidationResults(Customer customer)
    {
      Validator<Customer> validator =
        ValidationFactory.CreateValidator<Customer>("RuleSetA");

      ValidationResults results = validator.Validate(customer);
      if (results.IsValid)
      {
        Console.WriteLine("Deze klant is geldig");
      }

      foreach (ValidationResult result in results)
      {
        Console.WriteLine(result.Message);
      }
    }
  }
}
```

Codevoorbeeld 2.

```
//De 'QuoteRetriever' leidt af van een interface, dit bepaald welke
//methodes beschikbaar zijn voor 'Injection'. Ook kun je van
//MarshalByRefObject afleiden, om alle methodes mogelijk te injection.
public class QuoteRetriever : IQuoteRetriever
{
  //Deze methode is gedefineerd in interface IQuoteRetriever.
  public List<Quote> RetrieveQuotes(string symbol)
  {
    List<Quote> quotes = new List<Quote>();
    //... do hier het echte werk

    return quotes;
  }
}
interface IQuoteRetriever
{
  List<Quote> RetrieveQuotes(string symbol);
}
```

Codevoorbeeld 3.

te nemen in een resourcefile, zodat ook het meertalig maken van jouw applicatie eenvoudig is. In codevoorbeeld 2 is te zien hoe je vervolgens een instantie van een klasse kunt valideren. De voorbeelden in dit artikel zijn erg eenvoudig; voor ingewikkeldere voorbeelden en integratie met bijvoorbeeld user-interface-controls verwijzen we graag naar de QuickStarts die met EntLib worden meegeleverd.

Policy Injection Application Block

Hoewel je met het gebruik van Enterprise Library veel minder code zelf hoeft te schrijven, was er altijd de noodzaak om die ene regel waarin je bijvoorbeeld het Logging Block aanroept in je eigen code op te nemen. We nemen niet aan dat deze noodzaak overbodig aandoet, het is de manier waarop software veelal ontwikkeld wordt. Toch zou er iets voor te zeggen zijn als de code die je zelf schrijft enkel over de werking van jouw specifieke programma gaat, zonder hier concepten als logging, exception handling of validatie in te betrekken. Het Policy Injection Application Block geeft de mogelijkheid de andere blocks te gebruiken, zonder dat de aanroep hiervoor in jouw eigen code opgenomen hoeft te worden. In de Configuration-console kun je voor het Policy Injection Application Block een 'policy' definiëren. Deze policy is een samenstelling van 'handlers' die verwijzen naar een application block van Enterprise Library en zou gedefinieerd kunnen worden als: "Alleen gebruikers in de groep Manager krijgen toegang tot deze functionaliteit [Security Block], alle input moet worden gevalideerd [via het Validation Block], daarom moeten fouten worden gelogged in de database [Exception Handling Block]". Naast handlers kent een policy ook 'matching rules'. De matching rules definiëren wanneer een gegeven policy van

```
public class Program
{
  public static void PrintQuotes()
  {
    //Hier wordt een instantie van de QuoteRetriever gemaakt d.m.v. de
    //PolicyInjection factory. In de configuratie staan verijwingen naar
    //de methodes waar EntLib aangeroepen met worden.
    IQuoteRetriever quoteRetriever = PolicyInjection.
    Create<QuoteRetriever, IQuoteRetriever>();

    //Deze methode wordt uitgevoerd middels het Policy Injection block.
    //hier zou je bijvoorbeeld authenticatie kunnen doen, alvorens de
    //geschreven code uit te voeren.
    foreach (Quote q in quoteRetriever.RetrieveQuotes("MSFT"))
    {
      Console.WriteLine(q.ToString());
    }
  }
}
```

Codevoorbeeld 4.

toepassing is. Zo kun je definiëren dat de bovenstaande policy van toepassing is op alle classes in de AccountManagement-namespace.
De informatie uit de handlers ('wat') en de matching rules ('waar') stelt het Policy Injection Block in staat om EntLib te gebruiken zonder dat hier code aan te pas komt! Wel moeten de objecten, waarop deze policies van toepassing zijn, door een factory van EntLib geïnstantieerd worden. In codevoorbeeld 3 zie je hoe je een klasse definieert die geschikt gemaakt is voor gebruik binnen het Policy Injection-application block. In codevoorbeeld 4 zie je hoe je de klasse kan instantiëren via een EntLib-factory, zodat het Policy Injection-application block zijn werk kan doen. Het meest voor de hand liggende voordeel bij het gebruik van Policy Injection is dat je minder regels code hoeft te schrijven. Er zijn echter nog wat minder voor de hand liggende voordelen waarvan geprofiteerd kan worden:

- Er ontstaat een betere scheiding tussen zaken als 'Logging', 'Autorisatie', 'Exception Handling', enzovoort
- De geschreven code is beter te interpreteren en gemakkelijker te onderhouden, omdat deze randvoorwaarden niet meer in de code opgenomen hoeven te worden.
- Het veranderen van de werking van een policy kan op één plek aangepast worden.
- Bij het ontwikkelen van frameworks kan naderhand functionaliteit worden toegevoegd, zonder dat het framework zelf aangepast hoeft te worden.

Migratie

De publieke API van EntLib 3.0 is compatibel gebleven met de laatst gereleasede versie. Dit maakt dat je bij het migreren van versie 2.0 enkel versienummers en referenties aan hoeft te passen. Het is mogelijk meer versies van EntLib op één machine te draaien, maar het is niet mogelijk meer versies binnen één applicatie te gebruiken. Voor het migreren van projecten die gebruikmaken van eerdere versies van EntLib wordt documentatie meegeleverd.

Werkbesparing

Als je net begint met nieuwe projecten op basis van .NET Framework 2.0 of 3.0, dan kun je jezelf een hoop werk besparen door gebruik te maken van EntLib 3.0. Ook het nieuwe Policy Injection Application Block en het Validation Application Block zullen weer veel code vervangen die vaak zelf ontwikkeld moest worden. Het is wel van belang dat goed gekeken wordt naar de eisen van jouw specifieke project of organisatie. Over de afweging welke componenten uit EntLib wel of niet te gebruiken zijn, is veel informatie beschikbaar. De eindverantwoording over migratie of wat het 'juiste gebruik' is in jouw project houd je gelukkig zelf in de hand.

Olaf Conijn is een onafhankelijk consultant en is door Microsoft Corporation voor de tweede opeenvolgende keer ingehuurd om mee te werken aan Enterprise Library. Voor deze recentste versie is hij - naast divers ontwikkelwerk - vooral betrokken geweest bij het ontwerp en de implementatie van het Policy Injection Block en de configuratie tooling. Olaf is te bereiken op email@olafconijn.com of via zijn blog http://bloggingabout.net/blogs/olaf.

Dennis Mulder is senior principal consultant bij Avanade (www.avanade.com), een samenwerkingsverband van Microsoft en Accenture en heeft EntLib al veelvuldig ingezet op projecten. Dit is het derde artikel van Dennis in .NET Magazine over EntLib. Voor vragen en opmerkingen is Dennis te bereiken op dennism@avanade.com of via zijn blog http://www.dennismulder.net.

Referenties

http://www.codeplex.com/entlib
http://bloggingabout.net/blogs/olaf
http://www.dennismulder.net
http://blogs.msdn.com/tomholl
http://blogs.msdn.com/edjez