Delphi OplossingsCourant

Vol. 11. No. 1, Een gratis electronische publicatie van **Bob Swart Training & Consultancy** (**eBob42**) - http://www.eBob42.com

Helmond, 15 juni 2009,

Welkom bij het nieuwe nummer van de Delphi OplossingsCourant. Dit nummer staat in het teken van Delphi 2009 en Delphi Prism 2009 – en de combinatie van deze twee in RAD Studio 2009 (waar overigens ook C++Builder 2009 deel van uitmaakt). Wie nog geen Delphi 2009 gebruikt, en eigenlijk ook interesse heeft in Delphi Prism zou gebruik kunnen maken van de speciale aanbieding (geldig in de maand juni – zie pagina 5) waarbij je bij een upgrade naar een editie van Delphi 2009 een vergelijkbare editie RAD Studio 2009 ontvangt (dus inclusief C++Builder 2009 en Delphi Prism).
Ook vraag ik in dit nummer aandacht voor de SDN en de komende SDN events: de SDE op vrijdag 26 juni en de SDC op 19 en 20 oktober. Als nieuwe voorzitter van de Delphi sectie hoop ik daar ook vele lezers van de Delphi OplossingsCourant te zien. Wie de SDN niet kent – zie www.sdn.nl voor meer over het Software Development Network.
Naast een artikel over Delphi 2009 en een tweetal artikelen over Delphi Prism 2009, staat er in deze DOC ook een verslag van een bezoek dat de Delphi sectie van de SDN mocht brengen aan Wayne Williams, de CEO van Embarcadero. En inmiddels zijn er al wat meer kleine stukjes nieuws bekend gemaakt waaruit we kunnen afleiden dat cross-platform weer een belangrijk aandacht punt geworden is voor Delphi. Linux en MacOS worden genoemd, wellicht nog voor de Windows 64-bits versie. Tijdens mijn eerstvolgende gratis Delphi seminar in Helmond Brandevoort op 11 september hoop ik hier meer over te kunnen vertellen. Hou m'n weblog in de gaten voor de aankondingen…

Groetjes,

Bob Swart Training & Consultancy

Met mijn bedrijf Bob Swart Training & Consultancy hou ik me met name bezig met de ondersteuning van Delphi ontwikkelaars door middel van consultancy, trainingen en workshops, coaching en support (ook per e-mail), en het leveren van de benodigde licenties, subscriptions en cursusboeken voor zelfstudie (al dan niet onder begeleiding).

Trainingen en Workshops

Mijn Delphi trainingen zijn inclusief dik cursusboek en e-mail ondersteuning (ook achteraf als de training is afgelopen) plus toegang tot het members-only deel van m'n website. Vanaf 2007 worden de trainingen gegeven in de cursusruimte in Helmond Brandevoort (ideaal met de trein te bereiken), zie ook www.eBob42.org.

Alle Win32 en .NET onderwerpen zijn ook als maatwerk training mogelijk in de vorm van een Delphi Workshop in Helmond Brandevoort, of Custom Clinic (op locatie).
Zie http://www.eBob42.com/workshop voor details.

Reseller

Bob Swart Training & Consultancy is reseller van Embarcadero voor de BeNeLux, en levert zowel Delphi (Prism) als CodeGear RAD Studio inclusief persoonlijke ondersteuning (en nieuwsgroepen). Ook All-Access en subscriptions zijn verkrijgbaar.
Zie http://www.eBob42.com/CodeGear voor details.

Delphi (Prism) 2009 Updates

Er zijn dit jaar al een flink aantal updates voor Delphi 2009 en Delphi Prism verschenen. Voor Delphi 2009 is er een eenvoudige manier om up-to-date te blijven door gebruik te maken van de "Check for Updates" tool in de Delphi 2009 program group onder Windows. Voor Delphi Prism is er geen vergelijkbare mogelijkheid, maar omdat daar ieder kwartaal een nieuwe versie verschijnt die onder de subscription valt, krijgt iedereen met een nog geldige RAD Studio of Delphi Prism subscription automatisch een e-mail van Embarcadero met de download locatie voor de nieuwe versie. Neem contact op met je reseller als dit proces niet duidelijk is of als je daar nog vragen of opmerkingen over hebt (en wie eventueel wil overstappen van reseller: dat kan op het moment waarop de subscription aan verlenging toe is, maar het verdient wel de aanbeveling om dit dan al een maand of twee van te voren kenbaar te maken aan je oude en nieuwe reseller – ik ben te bereiken onder Bob@eBob42.com ;-)).

Delphi 2009

Vanaf november 2008 verschenen de eerste updates van Delphi 2009. Update 2 en 4 zijn database specifiek, update 3 was weer een algemene update. Er zijn twee help updates geweest, de laatste in april 2009.

Datum	Update	Release Notes
november 2008	Update #1	http://edn.embarcadero.com/article/38868
december 2008	Update #2 (DB)	http://edn.embarcadero.com/article/39005
december 2008	Help Update #1	http://edn.embarcadero.com/article/39139
april 2009	Help Update #2	http://edn.embarcadero.com/article/39449
mei 2009	Update #3 en Update #4 (DB)	http://edn.embarcadero.com/article/39611 http://edn.embarcadero.com/article/39390

Mochten er nog bugs zijn in Delphi 2009, dan wordt het aan geraden om te kijken of de bug al in Quality Central staat – zie http://qc.embarcadero.com – en indien nodig zelf een error report aan te maken, met een klein stukje code om het probleem te demonstreren. Neem desnoods contact op met mij (of je eigen reseller) om te zien of ik nog een helpende hand kan bieden..

Delphi Prism 2009

Delphi Prism 2009 is eind 2008 uitgekomen. Het wordt alleen verkocht samen met subscription, zodat iedereen die Delphi Prism "los" heeft gekocht gegarandeerd al van alle upgrades gebruik heeft kunnen maken. Voor Delphi Prism worden namelijk regelmatig – eens per kwartaal – nieuwe versies uitgebracht. Niet in de vorm van een nieuw versienummer (dan zouden we nu al op Delphi Prism R3 zitten), maar gewoon een upgrade met flink wat bug fixes en soms zelfs nieuwe features (zoals in de laatste upgrade waar nu ook Aspect Oriented Programming zaken in terug te vinden zijn).
Wie Delphi Prism krijgt als onderdeel van RAD Studio 2009, heeft geen verplichting tot subscription. Het gevolg is echter dan wel dat een nieuwe upgrade niet beschikbaar komt, omdat deze in principe alleen voor ontwikkelaars met een actieve subscription zijn (tenzij je RAD Studio aanschaft vlak nadat er een nieuwe upgrade van Delphi Prism is uitgekomen).

Er zijn inmiddels al twee upgrades van Delphi Prism uitgekomen:

Datum	Update	Release Notes
maart 2009	Feb 2009	http://edn.embarcadero.com/article/39408
mei 2009	May 2009	http://edn.embarcadero.com/article/39578

Deze zijn beschikbaar voor iedereen met een actieve subscription. De meest recente ISO image is ook te downloaden en op een DVD te branden indien gewenst.

SDN, SDE en SDC

Sinds begin 2009 ben ik voorzitter van de Delphi sectie van de Software Development Network (SDN), en hou me als zodanig o.a. bezig met het selecteren van sprekers en het samenstellen van de Delphi sessies voor de verschillende evenementen.

SDE 26 juni

Op vrijdag 26 juni is de eerstvolgende Software Development Event (SDE – www.sdn.nl/sde) weer, deze keer in Hotel Houten te Houten. Voor de Delphi sectie hebben we Michael Li bereid gevonden om een drietal Delphi sessie te komen verzorgen:

10 Tips to make your Delphi Win32 App .NET Ready

In deze sessie gaat Michael in op de vraag "Do I need to move to .NET?", waar zowel positieve als negatieve argumenten voor te verzinnen zijn. Michael gaat in op 10 specifieke technieken die je kan gebruiken in je Win32 toepassingen om een eventuele migratie naar .NET zo pijnloos mogelijk te laten verlopen. Dus mocht je ooit toch de overstap naar .NET maken, dan zal de migratie impact minimaal zijn. De onderwerpen die Michael in zijn sessie aan de orde laat komen zijn het resultaat en Best Practices die hij verzameld heeft sinds 2003 bij het begeleiden van klanten die de stap van Win32 naar .NET hebben gemaakt.

ER/Studio for Delphi Developers

Nu Embarcadero de nieuwe eigenaar van Delphi is, en Delphi Architect met een editie van ER/Studio komt (in plaats van ECO zoals vroeger), is de vraag: wat kan ik eigenlijk met ER/Studio beginnen?
In deze sessie zal Michael laten zien wat ER/Studio is, en hoe je er als Delphi ontwikkelaar optimaal gebruik van kan maken. Onderwerpen zijn o.a. data modeling, round-trip database support, data reverse-engineering, plus het analyseren en optimaliseren van databases. Iedereen die nog steeds met de hand z'n SQL aan het schrijven is zou eigenlijk deze sessie moeten bijwonen om te zien waarom ER/Studio de leading tool is in de data modeling arena.

Delphi 2009 IDE for Delphi 5-6-7.x developers

Er zijn nog een hele grote groep Delphi 5-6-7.x ontwikkelaars, die om wat voor reden dan ook nog niet de overstap hebben gemaakt naar Delphi 2009 *(er is in de maand juni een leuke aanbieding, hint hint).*
In deze sessie gaat Michael Li met name voor deze ontwikkelaars laten zien wat de Delphi 2009 IDE allemaal voor extra mogelijkheden biedt. Maar zelfs voor wie al een tijdje met Delphi 2009 werkt is het waarschijnlijk dat er nog wel enkele tips bij zitten om effectiever gebruik te maken van de nieuwste Delphi IDE.
Naast de drie Delphi sessies van Michael Li, zal ik ook zelf een Delphi sessie verzorgen op 26 juni:

ASP.NET Development met Delphi Prism

In deze sessie laat ik zelf zien hoe we ASP.NET toepassingen kunnen maken met Delphi Prism. Zowel ASP.NET Web sites als ASP.NET Projects komen aan de orde, de verschillen (code beside vs. code behind bijvoorbeeld), en speciale mogelijkheden plus deployment. Daarnaast besteed ik aandacht aan zaken als master-content pages, membersip & roles, en declarative data binding.
Ook zal ik laten zien hoe we ASP.NET web services en HTTP Handers kunnen toevoegen, en tot slot enkele migratie tips voor wie bestaande projecten met BDe 2006 voor ASP.NET 1.1 en/of RAD Studio 2007 voor ASP.NET 2.0 kunnen migreren naar Delphi Prism.

Zie http://www.sdn.nl/sde voor de volledige agenda en om je in te schrijven. Niet SDN-leden zijn ook van harte welkom (tegen een vergoeding van 95 Euro voor de hele dag, inclusief koffie en lunch).

SDC 19 en 20 oktober

Op 19 en 20 oktober 2009 zal de SDC plaatsvinden in Papendal, Arnhem. De call for speakers is zojuist gesloten, en binnenkort start het proces van de selectie van de spreker en sessies. Ik heb er alle vertrouwen in dat we weer met een sterke groep Delphi sprekers een goed sessieaanbod kunnen leveren. Hou www.sdc.nl in de gaten voor meer details.

Wayne Williams en Embarcadero

Op woensdag 11 maart hadden Johan Parent en Bob Swart namens de Delphi Sectie van de SDN het genoegen om Wayne Williams, CEO van Embarcadero in levende lijve te ontmoeten. Het moet gezegd: het klikte meteen. Wayne is niet een doorsnee CEO, maar een echte ontwikkelaar met liefde voor tools. En dat is toch wat anders dan een CEO die alleen maar op toneel rondhuppeld en "Developers, developers, developers!" uitkraamt. Wayne Williams is begonnen als Clipper ontwikkelaar (zal menig SDN'er bekend in de oren klinken) is met Turbo Pascal 1.0 begonnen en heeft natuurlijk ook Delphi gebruikt. Na een zestal startups (waarvan de helft voor goed geld verkocht), is hij uiteindelijk als CEO bij Embarcadero terecht gekomen.

Embarcadero

Embarcadero heeft zich tot voor kort met name gericht op database tools – niet zozeer op een DBMS, maar juist de tools om met databases te werken. En waar CodeGear zich met ontwikkeltools als Delphi, C++Builder en JBuilder bezighoudt, daar kunnen de oorspronkelijke tools van Embarcadero onder de naam DataBaseGear geschaard worden. De acquisitie van CodeGear door Embarcadero voelde dan ook als het weer bijeen brengen van een bij-de-geboorte-gescheiden-tweeling, alsdus Wayne. Het goede deel van Borland – het oorspronkelijke Borland – is gekocht, en samen vormen ze nu het toolaanbod van Embarcadero.

Dat Embarcadero serieus is betreffende de toekomst van Delphi, C++Builder en InterBase blijkt wel uit de plannen die Wayne toelicht. Zo zal er nog dit jaar ondersteuning voor Interbase (en Firebird) in de Embarcadero database tools komen. Waar hun middle-ware kern nu met databases als IBM DB2, Oracle, Microsoft SQL Server, MySQL, en Sybase werkt, zal daar InterBase aan toegevoegd worden. Daarnaast kunnen we ook voor Delphi en C++Builder het nodige verwachten, waarover later meer.

All-Access

De meest recente aankondiging van Embarcadero betreft All-Access. Dit programma zet de subscription op een enkele tool om in een meberschip, waarbij je toegang krijgt tot alle 18 tools van Embarcadero. In de ogen van Wayne Williams is dit de ultieme gereedschapskist die iedere ontwikkelaar wel zou willen hebben. Zeker als de prijs goed is. Er zijn vier verschillende All-Access levels: brons, zilver, goud en platinum, en per level krijg je verschillende edities van alle Embarcadero tools. Brons staat gelijk aan Delphi Professional bijvoorbeeld, voor Zilver krijg je Enterprise, en Goud en Platinum zijn goed voor de Architect editie.

Instant-On

Los van de hoeveelheid tools, introduceert All-Access nog een ander uniek fenomeen, genaamd Instant On. Dit houdt in dat je met je acess key niet langer de tools fysiek op je machine hoeft te installeren, maar die kan draaien vanaf een All-Access server. Dat scheelt een hoop configuratie en instellingen, zeker als je je eigen instellingen kunt opslaan (in een persistent settings bestand) om later op een andere machine weer te gebruiken met de Instant On functionaliteit. Op dit moment is Instant-On voor de meeste Embarcadero tools beschikbaar. Voor Delphi, C++Builder en Delphi Prism (de onderdelen van RAD Studio) kunnen we Instant-On binnenkort verwachten. Later komt er een versie waarbij ook de installatie van third-party componenten en configuratie van de hele IDE bewaard kunnen worden en kunnen worden meegenomen.

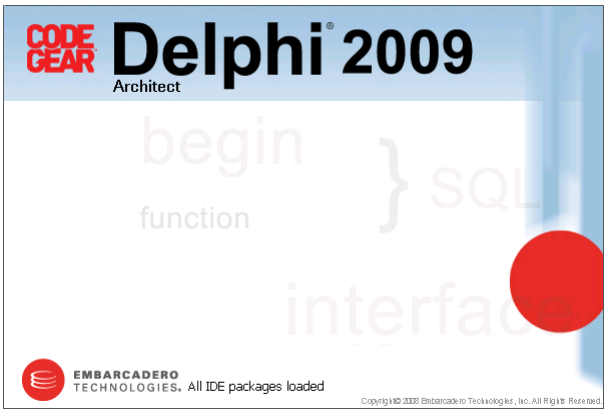
Het gebruik van Instant-On en All-Access betekent ook een vereenvoudiging van het licentiebeheer, vooral voor grotere bedrijven. Met mogelijkheden voor named workstation of concurrent All-Access licenties kan flexibel worden omgegaan met de tools in de Embarcadero gereedschapkist.

Conclusie

De toekomst van CodeGear tools als Delphi, C++Builder en InterBase lijkt in goede handen bij Embarcadero, en het All-Access verhaal is – zeker met Instant-On – een innovatieve manier die het gebruik en beheer van tools en licenties zal vereenvoudigen. Wie nog gewoon losse licenties, upgrades en/of subscriptions koopt: geen angst: daar zal niks in wijzigen. Je zult nog altijd een losse upgrade of nieuwe licentie van Delphi kunnen kopen, al dan niet met een losse subscription voor deze ene tool (in plaats van een "membership" voor de hele toolbox).

Delphi 2009 en Delphi Prism 2009

Dit nummer van de Delphi OplossingsCourant bevat enkele artikelen over Delphi 2009 for Win32 en Delphi Prism 2009. Zij maken samen deel uit van RAD Studio 2009 (waar ook nog C++Builder 2009 in zit overigens).



Voor wie interesse heeft in zowel Delphi 2009 als Delphi Prism 2009, is het goed om te weten dat er een **speciale actie gedurende de maand juni 2009** geldt: bij aanschaf van een Delphi 2009 upgrade krijgt u (na betaling) een RAD Studio 2009 upgrade geleverd. Dat is dus in feite een gratis C++Builder 2009 en RAD Studio 2009. Deze actie is geldig in de BeNeLux, maar alleen voor upgrades van Delphi 2009 – en dus niet voor New User licenties.
Wie bovendien deze upgrade bij mij bestelt, bijvoorbeeld via www.bobswart.nl, die krijgt ook de PDF versie van mijn Delphi 2009 Development Essentials boek er gratis bij, alsmede een preview van mijn Delphi Prism Development Essentials cursusboek. Deze boeken zijn Engelstalig, maar in dit nummer van de DOC zijn twee onderdelen (met nieuwe taalelementen) voor uw gemak vertaald naar het Nederlands.

Concreet zijn de prijzen voor Delphi 2009 upgrades nu als volgt:

Delphi 2009 Professional Upgrade *(levering RAD Studio 2009 Professional Upgrade – normaal € 649)*	**349 Euro**
Delphi 2009 Enterprise Upgrade *(levering: RAD Studio 2009 Enterprise Upgrade – normaal € 1799)*	**1.199 Euro**
Delphi 2009 Architect Upgrade *(levering RAD Studio 2009 Architect Upgrade – normaal € 2799)*	**2.011 Euro**

Deze prijzen zijn nog exclusief 19% BTW (klanten van mij uit België of Luxemburg kunnen de BTW laten verleggen indien zij hun BTW nummer vermelden bij de bestelling).
Zie http://www.bobswart.nl voor de New User prijzen. Ook daarvoor geldt dat u hierbij mijn cursusboeken alsmede persoonlijke ondersteuning en toegang tot mijn nieuwsgroepen er gratis bij krijgt.

Wie nog vragen of opmerkingen heeft, of Delphi 2009 en/of Delphi Prism 2009 zelf een keer wil proberen, kan me altijd per e-mail bereiken of een bezoek komen brengen in Helmond Brandevoort (zie www.eBob42.com voor details). Op 11 september is daar de eerstvolgende gratis Delphi middag in het 't Brandpunt – details worden binnenkort bekend gemaakt.

Delphi 2009 Generics en Anonymous Methods

Wat betreft nieuwe features van Delphi 2009 gaat de aandacht meestal uit naar de Unicode ondersteuning, gevolgd door DataSnap 2009, maar ook de andere taaluitbreidingen zoals Generics en Anonymous Methods zijn beslist de moeite waard. Probleem is een beetje dat de on-line help zoals gewoonlijk wat achterloopt, dus niet alle (on)mogelijkheden zijn even goed gedocumenteerd. Maar ik hoop met dit artikel een beetje licht in de schemering te kunnen werpen.

Generics

Laten we beginnen met Generics, of Parameterised Types, zoals ze ook wel in Delphi worden genoemd. Deze vinden zijn oorsprong in .NET eerlijk gezegd, en waren vorig jaar beloofd als nieuwe feature toen het niet haalbaar bleek om Generics al toe te voegen aan Delphi 2007 for Win32. De syntax is grotendeels vergelijkbaar aan die van de Delphi for .NET syntax, met als uitzondering dat het .NET Framework een aantal zaken regelt die onder Win32 niet mogelijk zijn (in .NET kun je alles naar een object boxen, en bestaan er al standaard interfaces om values met elkaar te vergelijken).
Een generic type definitie kan gebruikt worden om functionaliteit te presenteren waarbij je van te voren nog niet weet of je die voor integers, strings, floats, of de een-of-andere object class wilt laten gebruiken. Als syntax worden de vishaken gebruikt met een type aanduiding ertussen (meestal een T, maar je kan er alles voor gebruiken, behalve reserved keywords).
Om een verzameling van elementen van het type T weer te geven, gerepresenteerd door een array van T, kunnen we het volgende schrijven:

```
type
  Verzameling<T> = Array of T;
```

We kunnen variabelen direct van het type Verzameling<T> declareren, waarbij we dan wel voor de T een daadwerkelijk type moeten aangeven (de instantie van het type parameter als het ware), bijvoorbeeld als volgt:

```
type
  VerzamelingIntegers = Verzameling<integer>;

var
  X: VerzamelingIntegers;
  Y: Verzameling<integer>; // kan natuurlijk ook
```

Behalve de type parameter T kunnen we ook constraints toevoegen, die daarmee aangeven dat we bij de instantiatie van het type aan bepaalde voorwaarden moeten voldoen. Een constraint is ofwel een class type waar de T van afgeleid moet zijn, of een interface die de T moet implementeren.
Dus om een verzameling van elementen van een type T weer te geven waarbij T ten minste een TComponent moet zijn, kunnen we de volgende syntax gebruiken:

```
type
  Verzameling<T: TComponent> = Array of T;
```

Het type Verzameling<integer> zal nu niet meer compileren, maar Verzameling<TComponent> wel, en dat geldt ook voor de elementen die dan minstens van type TComponent moeten zijn.
We kunnen behalve TComponent ook TPersistent gebruiken, maar het is niet mogelijk om TObject als constraint op te geven. In plaats daarvan moeten we dan het keyword class gebruiken:

```
type
  Verzameling<T: class> = Array of T;
```

En dit zegt dat we een verzameling kunnen bouwen van elementen die class instanties zijn (en geen integer of strings bijvoorbeeld).

Generic Stack

Om eens een voorbeeld te geven van het gebruik van Generics is het noodzakelijk om ook functionaliteit toe te voegen. Alleen maar een Array van een type T is niet zo zinvol, maar als je er ook mee kunt manipuleren (of onderling vergelijken, zoals ik aan het eind van dit artikel ga doen), dan is het plotseling een stuk handiger.
Een voor de hand liggend voorbeeld is de Generic TStack, die als volgt gedefinieerd kan worden:

```
type
  TStack<AnyType> = class
  protected
    FCount: Word;
    FStack: Array of AnyType; // needs to grow when needed
  public
    constructor Create(Capacity: Word = 42);
    destructor Destroy; override;
    function Count: Integer;
    function Push(const item: AnyType): Word;
    function Peek: AnyType;
    function Pop: AnyType;
  end;
```

Omdat ik een array gebruik om de elementen van type T in op te slaan, moeten we zorgen dat de stack groot genoeg is (en blijft). Als eerste voorziening heb ik de constructor een default argument gegeven die de stack een opbergruimte van maximaal 42 elementen geeft (waar in het begin nog niks in zit natuurlijk, alleen de maximale ruimte is vast gereserveerd).
De destructor roept de SetLength weer aan om het array weer op te ruimen. Let op: als de elementen instanties van classes zijn, dan wordt hun destructor niet aangeroepen, dus dit kan tot een geheugenlek leiden als je de stack niet hebt leeggemaakt voor je hem weggooit. DE TStack is dus niet de eigenaar van de items die je erin stopt (dat kun je er wel van maken, maar dat moet je bijvoorbeeld aangeven dat de elementen minstens classes zijn via de constraint syntax, en dan in de destructor eerst Free aanroepen van alle elementen voordat je de SetLength op het array aanroept – maar dat laat ik over als oefening voor de lezer).

```
constructor TStack<AnyType>.Create(Capacity: Word = 42);
begin
  inherited Create;
  FCount := 0;
  SetLength(FStack, Capacity)
end;

destructor TStack<AnyType>.Destroy;
begin
  SetLength(FStack,0);
  inherited
end;
```

De Push methode moet eerst kijken of het interne array van de stack wel groot genoeg is om het nieuwe element op te slaan. Zo niet, dan moet SetLength opnieuw worden aangeroepen om het array groter te doen groeien. Hiertoe kunnen we SetLength weer opnieuw aanroepen (die er tevens voor zorgt dat alle bestaande elementen uit het array meegenomen worden naar de nieuwe lengte). In de code vergroot ik het array met stapjes van 1 (door de nieuwe benodigde count mee te geven), maar het is wellicht slimmer om dit in iets grotere stappen te doen, afhankelijk van de behoefte aan extra opslagruimte van de stack.
De implementatie van de Peek en Pop spreekt voor zich neem ik aan. Merk op dat we hier steeds het AnyType gebruiken, waardoor de stack dus inderdaad van alles kan bevatten, van strings tot integers of TDataSets.

```
function TStack<AnyType>.Count: Integer;
begin
  Result := FCount
end;
```

```
function TStack<AnyType>.Push(const item: AnyType): Word;
begin
  Inc(FCount);
  if FCount > Length(FStack) then
    SetLength(FStack, FCount);
  FStack[FCount-1] := item;
  Result := FCount
end;

function TStack<AnyType>.Peek: AnyType;
begin
  Result := FStack[FCount-1]
end;

function TStack<AnyType>.Pop: AnyType;
begin
  Result := Peek;
  Dec(FCount)
end;
```

We kunnen de TStack gebruiken voor een lijst van strings die we in omgekeerde volgorde weer willen printen, als volgt:

```
type
  TStringStack = TStack<string>;

var
  S: TStringStack;

begin
  S := TStringStack.Create;
  try
    S.Push('Delphi 2009');
    S.Push('C++Builder 2009');
    writeln(S.Pop);
    writeln(S.Pop);
  finally
    S.Free
  end;
end.
```

Of als een stack van TPersistent classes, waarbij we bijvoorbeeld de Name en de ClassName maar ook de UnitName (ook een nieuwe property in Delphi 2009) kunnen laten zien:

```
procedure TFormMain.ButtonClick(Sender: TObject);
var
  CompStack: TStack<TPersistent>;
  Comp: TComponent;
begin
  CompStack := TStack<TPersistent>.Create;
  try
    for Comp in Self do CompStack.Push(Comp);
    while CompStack.Count > 0 do
    begin
      Comp := CompStack.Pop as TComponent;
      ShowMessage(Comp.Name + ': ' + Comp.ClassName + ' (' + Comp.UnitName + ')')
    end
  finally
    CompStack.Free
  end
end;
```

Voor dit voorbeeld is het maar goed dat de stack niet zijn elementen allemaal “freed”, want anders zou het Form plotseling leeg zijn nadat we deze code hebben uitgevoerd (leuk voor een demo, maar niet leuk in praktijk).

Overigens zijn er al veel Generic types te vinden in de unit Generics.Collections, dus hoeven we niet het wiel opnieuw uit te vinden.

Generic Methods

Behalve Generic Types maak ik zelf ook veel gebruik van Generic Methods (in het vorige nummer van de Delphi OplossingsCourant stond een artikel over Generic Methods voor Delphi for .NET in CodeGear RAD Studio 2007 – toen was dit nog niet mogelijk in Delphi for Win32).
Het enige jammere is dat ik hier geen gewone lokale routines voor kan gebruiken, maar dat in de vorm van class methods moet doen.
We kunnen hierbij de type parameter op twee plaatsen toevoegen: of bij de class of bij de method zelf. In de volgende code snippet staan beide alternatieven naast elkaar:

```
type
  TGeneric<AnyType> = class
    class procedure Swap(var X,Y: AnyType); static;
  end;

type
  TGeneric = class
    class procedure Swap<AnyType>(var X,Y: AnyType); static;
  end;
```

Als het maar om één method gaat is het verschil niet zo groot, maar als er meerdere methods worden toegevoegd zul je al snel merken dat het flexibeler is om de type parameter toe te voegen aan de method en niet aan de class. Zeker als er methods komen die wellicht meer dan één type parameter hebben.
De implementatie van de Generic Swap methode zelf is redelijk eenvoudig: een derde variabele van type AnyType en klaar is kees:

```
class procedure TGeneric.Swap<AnyType>(var X,Y: AnyType);
var
  Z: AnyType;
begin
  Z := X;
  X := Y;
  Y := Z
end;
```

Bij het aanroepen hoeven we natuurlijk geen instantie van TGeneric aan te maken, maar kunnen we direct de class function Swap aanroepen, met daarbij een specificatie voor de type parameter en de juiste argumenten.
Voor een tweetal integer variabelen kunnen we dit als volgt doen:

```
var
  A,B: Integer;
begin
  A := 42;
  B := 17;
  TGeneric.Swap<integer>(A,B);
```

Op dezelfde manier kunnen we andere class methods bouwen zoals een IFF (afhankelijk van een expressie krijg je de TrueValue of the FalseValue terug) of een ChooseDef (kies een item uit een lijst, maar als de index buiten het bereik van de lijst is krijg je een default waarde terug).

```
type
  TGeneric = class
    class function IFF<AnyType>(const Expression: Boolean;
      TrueValue: AnyType; FalseValue: AnyType): AnyType;

    class function ChooseDef<AnyType>(index: Integer;
      const values: Array of AnyType): AnyType;
  end;
```

Net als in het .NET Framework, kan in Win32 de default waarde van een type T worden gegeven – in dit geval door default(AnyType). Hierdoor wordt de implementatie van beide generic methods als volgt:

```
class function TGeneric.IFF<AnyType>(const Expression: Boolean;
  TrueValue, FalseValue: AnyType): AnyType;
begin
  if Expression then
    Result := TrueValue
  else
    Result := FalseValue
end;

class function TGeneric.ChooseDef<AnyType>(index: Integer;
  const values: array of AnyType): AnyType;
begin
  if (index >= Low(values)) and (index <= High(values)) then
    Result := values[index]
  else
    Result := default(AnyType) // default waarde van AnyType
end;
```

Voor ik verder ga met een laatste Generic Method (om het grootste of kleinste element van een lijst van type AnyType te vinden) wil ik eerst even een uitstapje maken naar een noodzakelijk bouwsteen daarvoor: de Anonymous Methods.

Anonymous Methods

Een Anonymous method is een methode zonder naam, maar wel een stuk code binnen een procedure of function sectie. Net als een stuk code dat je via copy-en-paste ergens hebt neergezet, maar dan met een "wrapper" eromheen. Ze zijn zeker niet overal even goed toepasbaar, en leiden ook niet altijd tot leesbare code, maar in sommige situaties kunnen ze een hulpmiddel zijn om de expressiemogelijkheden van de taal Delphi te vergroten. Aan de hand van een eerste eenvoudig voorbeeld zal ik tot slot een tweetal nuttige toepassingen van Anonymous Methods laten zien.
Een Anonymous Method geven we aan met de keywords "reference to procedure" (of function), inclusief de parameterlijst. Het invullen gaat dan in een blok code, waarbij er geen puntkomma komt tussen de header en de eerste begin.

```
type
  TProc = reference to procedure(x: Integer);

procedure call(const proc: TProc);
begin
  proc(42);
end;

var
  TheAnonymousMethod: TProc;
begin
  TheAnonymousMethod := procedure(a: Integer)
  begin
    Button1.Caption := IntToStr(a)
  end; // einde van de Anonymous Method
  call(TheAnonymousMethod)
end;
```

De assignment van het stukje code aan de variabele "TheAnonymousMethod" ziet er op het eerste gezicht niet zo leesbaar uit, vooral niet als het stuk code lang(er) wordt. Er zijn echter wel voordelen te halen uit het gebruik van Anonymous Methods, zoals ik in de volgende voorbeelden zal laten zien.
Nuttige Anonymous Methods
Een van de plaatsen waar Anonymous Methods van waarde kunnen zijn, is binnen de synchronize methode van een TThread component. Dit zou anders een extra methode kosten, inclusief velden voor de parameters van deze methode. Dit is o.a. in meer detail besproken in de weblog van Allen Bauer (http://blogs.codegear.com/abauer/2008/09/08/38868), betreffende project thrddemo in C:\Documents and Settings\All Users\Documents\RAD Studio\6.0\Demos\DelphiWin32\VCLWin32\Threads. Grappig genoeg is de daadwerkelijke demo niet aangepast volgens zijn suggesties, dus kunnen we dat zelf uitproberen. De oorspronkelijke code is terug te vinden in bovenstaande locatie, maar de aangepaste versie met Anonymous Methods zou er als volgt uit komen te zien:

```
procedure TSortThread.VisualSwap(A, B, I, J: Integer);
begin
  {$IFDEF ANONYMOUS}
  Synchronize(procedure
    begin
      with FBox do
      begin
        Canvas.Pen.Color := clBtnFace;
        PaintLine(Canvas, I, A);
        PaintLine(Canvas, J, B);
        Canvas.Pen.Color := clRed;
        PaintLine(Canvas, I, B);
        PaintLine(Canvas, J, A)
      end
    end)
  {$ELSE}
  FA := A;
  FB := B;
  FI := I;
  FJ := J;
  Synchronize(DoVisualSwap);
  {$ENDIF}
end;
```

De {$IFDEF ANONYMOUS} kan gebruikt worden om te switchen tussen de oorspronkelijke code (die de extra methode DoVisualSwap aanroept) en de aanroep van de Anonymous Method die de parameters A, B, I en J direct ontvangt. De niet-Anonymous Methods variant heeft ook nog een extra method DoVisualSwap nodig – zonder argumenten - die hetzelfde doet als de Anonymous Method. Dat is meteen de crux hier: de anonymous method kan gebruik maken van de context (de parameters A, B, I en J van de VisualSwap routine) om zijn "body" meteen uit te voeren, zonder daarbij properties te moeten gebruiken omdat Synchronize geen argumenten toestaat).

Een tweede voorbeeld van een nuttig gebruik van Anonymous Methods heb ik gevonden bij het maken van generic methods voor het bepalen van het grootste of kleinste element uit een lijst van items van een bepaald type. Omdat hier de vergelijking tussen twee elementen cruciaal is, kunnen we dit niet zomaar implementeren. Daar hebben we iets extra voor nodig, wat we o.a. terugvinden in de Generics.Defaults unit.

Generics.Defaults

Delphi 2009 bevat de units Generics.Defaults en Generics.Collections die al een hoop voorgedefinieerde nuttige voorbeelden van Generics (en Anonymous Methods) bevatten. Zo bevat Generics.Defaults de definitie van het IComparer<T> interface dat we straks willen gebruiken, en ook een TComparison<T> Anonymous Method kunnen we hier terugvinden:

```
type
  IComparer<T> = interface
    function Compare(const Left, Right: T): Integer;
  end;

   TComparison<T> = reference to function(const Left, Right: T): Integer;
```

Deze Anonymous Method kan o.a. gebruikt worden in de aanroep van de Construct methode van de TComparer<T> class, die het IComparer<T> interface implementeert.

```
  TComparer<T> = class(TInterfacedObject, IComparer<T>)
  public
    class function Default: IComparer<T>;
    class function Construct(const Comparison: TComparison<T>): IComparer<T>;
    function Compare(const Left, Right: T): Integer; virtual; abstract;
  end;
```

Daarnaast bevat de Generics.Defaults unit voorbeelden om de gelijkheid van generic values te bepalen, en een niet-reference counted IInterface implementatie in het type TSingletonImplementation, inclusief de implementaties van QueryInterface, _AddRef en _Release. Zie de source code voor details.

Combinatie van Anonymous Methods en Generics

Het laatste voorbeeld is er eentje die ik zelf regelmatig gebruik: een generic method die de grootste – of kleinste – waarde uit een lijst van values van een bepaald type teruggeeft. Vergelijkbaar met de ChooseDef, maar dan inclusief een vergelijking zoals we die in het IComparer interface uit de Generics.Defaults unit zagen.
De definitie van mijn functions Min en Max is als volgt:

```
type
  TGeneric = class
    class function Min<AnyType>(const values: array of AnyType;
      const Comparer: IComparer<AnyType>): AnyType;
    class function Max<AnyType>(const values: array of AnyType;
      const Comparer: IComparer<AnyType>): AnyType;
  end;
```

Als eerste argument geef ik de lijst van values van type AnyType mee, en als tweede argument het IComparer interface van hetzelfde type. Hoe we aan deze IComparer komen is nog even niet van belang – dat wordt tijdens de aanroep geregeld (met een Anonymous Method).
Wat we eerst moeten bekijken is de implementatie van de Min en Max functies, waarbij we de Compare functie van het IComparer interface kunnen gebruiken om steeds twee values van het AnyType te vergelijken met elkaar. De Compare geeft een positief getal terug als links groter is dan rechts, en anders een negatief getal. In feite kun je het zien als "links min rechts" als het getallen zouden zijn.

```
class function TGeneric.Max<AnyType>(const values: array of AnyType;
  const Comparer: IComparer<AnyType>): AnyType;
var
  item: AnyType;
begin
  if length(values) >= 1 then
  begin
    Result := values[Low(values)];
    for item in values do
      if Comparer.Compare(item,Result) > 0 then
        Result := item
  end
end;

class function TGeneric.Min<AnyType>(const values: array of AnyType;
  const Comparer: IComparer<AnyType>): AnyType;
var
  item: AnyType;
begin
  if length(values) >= 1 then
  begin
    Result := values[Low(values)];
    for item in values do
      if Comparer.Compare(item,Result) < 0 then
        Result := item
  end
end;
```

Tot nu toe is de code nog goed te lezen en niet al te vreemd gezien de generic methods die ik eerder liet zien. De slagroom op de taart krijgen we wanneer we de Min of Max willen aanroepen, bijvoorbeeld door het grootste getal uit een dynamisch array van integers op te leveren. Het eerste deel van de aanroep van Max zou er als volgt uit kunnen zien:

```
TGeneric.Max<integer>([1,2,4,8,16,32,64,42,36,13],
```

Voor het tweede argument hebben we dan iets nodig dat het IComparer interface oplevert. En daarvoor moeten we even terugdenken aan de Generics.Defaults unit, met daarin de TComparer class met de Construct class function die het IComparer interface teruggeeft. Omdat de Construct method een class function is, kunnen we gewoon TComparer<integer>.Construct aanroepen om een IComparer<integer> terug te krijgen.

Moeten we alleen nog het Comparison argument van de Construct meegeven, en dat is een TComparison<T> oftewel een reference to function(const Left, Right: T): Integer;
Om een lang verhaal kort te maken: bij de aanroep van Construct kunnen we dus meteen een Anonymous Method schrijven en die meegeven als argument, zoals in de volgende code te zien is:

```
TGeneric.Max<integer>([1,2,4,8,16,32,64,42,36,13],
  TComparer<integer>.Construct(
    function(const Left,Right: integer): integer
    begin
      Result := Left-Right
    end))
```

De Anonymous Method doet in feite niets anders dan Rechts van Links aftrekken om zo het gewenste gedrag te implementeren dat binnen de Max functie verwacht wordt. Tijdens de aanroep van TGeneric.Max<integer> geef ik dus behalve het type en de lijst van elementen van dat type ook meteen een dynamische vergelijke functie mee voor elementen van dat type (in de vorm van een Anonymous Method). En dat moet je misschien twee of drie keer lezen voor het duidelijk wordt, maar werkt wel perfect.

Als variatie op dit thema wil ik nog een laatste voorbeeld geven waarbij ik van een lijst van componenten de grootste wil hebben, waarbij ik de "grootste" definieer als het component met de langste string die de nieuwe ToString functie teruggeeft. Hiermee geef ik meteen aan dat ik van aanroep tot aanroep de implementatie van de vergelijking tussen twee elementen kan aanpassen door een andere Anonymous Method te schrijven.

Het laatste voorbeeld ziet er uiteindelijk als volgt uit, waarbij ik eerst een array van TComponents maak door de Self.Components van een TForm af te lopen:

```
procedure TFormX.ButtonClick(Sender: TObject);
var
  SelfComponents: array of TComponent;
  i: Integer;
begin
  SetLength(SelfComponents, Self.ComponentCount);
  for i:=0 to Self.ComponentCount-1 do
    SelfComponents[i] := Self.Components[i];

  ShowMessage(
    TGeneric.Max<TComponent>(SelfComponents,
      TComparer<TComponent>.Construct(
        function(const Left,Right: TComponent): integer
        begin
          Result := Length(Left.ToString) - Length(Right.ToString)
        end)).ToString
    )
end;
```

Met deze code in de OnClick van een TButton op een form krijg je als resultaat de inhoud van het component met de langste ToString waarde. Maar natuurlijk is het nu niet moeilijk meer om een eigen criterium te implementeren, of dit principe te gebruiken voor andere Generics en/of Anonymous Methods.

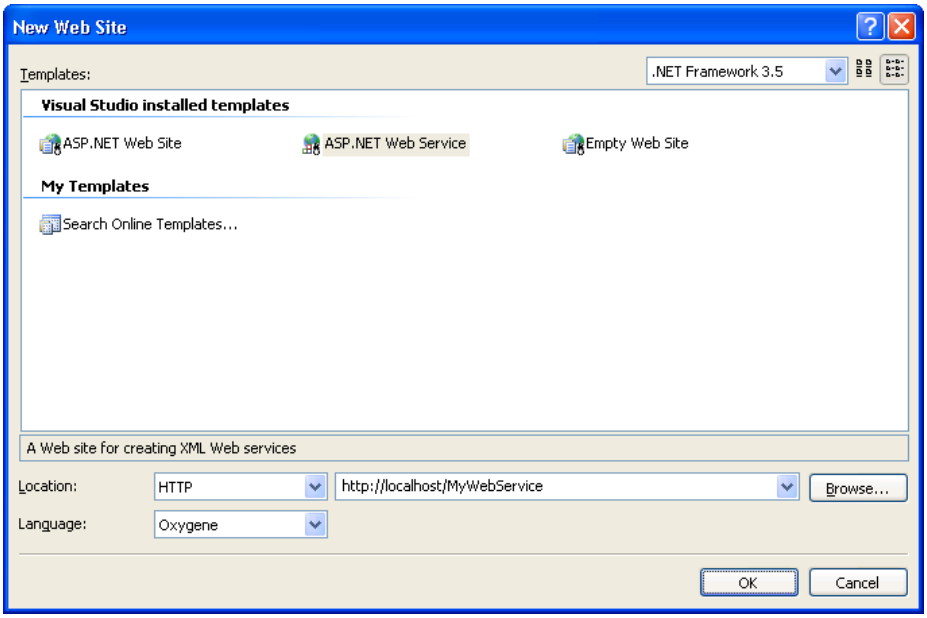
Conclusie

In dit artikel heb ik laten zien hoe de nieuwe Delphi 2009 taalelementen voor generic types en methods alsmede Anonymous Methods in elkaar zitten en hoe we die zelf kunnen maken en gebruiken. Vooral de combinatie van Generics en Anonymous Methods levert fijne nieuwe mogelijkheden die voorheen niet mogelijk waren zonder dit allemaal als vele losse routines uit te schrijven.

Tijdens de eerstvolgende gratis Delphi workshop in Helmond Brandevoort op vrijdag 11 september zal ik ook weer zowel Delphi for Win32 als Delphi Prism onderwerpen aan de orde laten komen. Zie mijn website of weblog te http://www.bobswart.nl/blog binnenkort voor de details.

ASP.NET Web Services met Delphi Prism 2009

Delphi Prism 2009 ondersteunt vele .NET targets, waaronder ASP.NET Web Services. Dat is belangrijk om te weten, aangezien de SOAP Server support in Delphi 2009 op dit moment "under consideration" is. Dus voor zeker voor nieuwe SOAP en Web Services projecten zouden we eigenlijk voornamelijk naar Delphi Prism moeten kijken (in eder geval wat betreft de server side).
Zoals in het eerste screenshot te zien is, krijgen we de ASP.NET Web Service keuze te zien na *File | New – Web Site*. Nadat je de ASP.NET Web Service icon hebt gekozen, moet je eerst nog wat meer informatie in deze dialoog invullen. Verander bijvoorbeeld de Location van File System naar HTTP (het alternatief is FTP), en wijs de locatie naar iets als http://localhost/MyWebService. Laat de .NET Framework 3.5 optie rechtsboven maar staan als je .NET 3.5 features (als LINQ) wilt gebruiken, anders kan die terug naar .NET 2.0 bijvoorbeeld. Deze keuze van het .NET Framework kan nuttig zijn als je bij voorbaat al weet dat de deployment web server alleen maar een specifieke versie van het .NET Framework bevat, of als je al bij voorbaat weet dat je bepaalde features nodig zult hebben.



Als je op OK klikt, zal Delphi Prism een nieuw Web Service project genereren op de opgegeven plek (op mijn voorbeeldmachine komt de logische URL http://localhost/MyWebService overeen met het fysieke pad C:\Inetpub\wwwroot\MyWebService).
Als je naar de Solution Explorer kijkt zie je het nieuwe project, met daarin een App_Code knoop waaronder een Service.pas unit zit, alsmede een lege App-Data knoop (waar je de database verbindingen kwijt kan – meer daarover een volgende keer), een Service.asmx file en een web.config file.
De inhoud van Service.pas is als volgt:

```
namespace;

interface

uses
  System,
  System.Web,
  System.Collections,
  System.Web.Services,
  System.Web.Services.Protocols;
```

```
type
  [WebService(&Namespace := 'http://tempuri.org/')]
  [WebServiceBinding(ConformsTo := WsiProfiles.BasicProfile1_1)]
  Service = public class(System.Web.Services.WebService)
  public
    method HelloWorld: string;
  end;

implementation

[WebMethod]
method Service.HelloWorld: string;
begin
  result := 'Hello World';
end;

end.
```

Er zijn meteen al een aantal interessante zaken die je hier zullen opvallen. Ten eerste blijkt dat het heel eenvoudig is om in .NET een Web Service te maken: je hoeft alleen maar een class af te leiden van de ingebouwde System.Web.Services.WebService class.

[WebService]

Daarnaast moet je ook gebruikmaken van het WebService attribuut, met name om de zogenaamde namespace van de web service te specificeren. Als je dat niet doet, dan zal de namespace de default waarde http://tempuri.org hebben (die overigens nu expliciet staat aangegeven in de gegenereerde code). Zeker in combinatie met het feit dat de web service zelf nu de naam "Service" heeft, leidt dit niet echt tot een unieke combinatie. Zeker niet als je een tweede service probeert te gebruiken die per ongeluk ook dezelfde naam heeft (en dan moet de namespace verschillend zijn). Om een lang verhaal kort te maken: het wordt zeer aangeraden om de default namespace aan te passen en er iets unieks in te zetten om de web service te onderscheiden van andere web services (van andere bedrijven of ontwikkelaars). Mocht je per ongeluk alsnog vergeten om de namespace aan te passen, dan zal de ASP.NET test omgeving je meteen op je vingers tikken zodra je de web service URL in een browser intikt om naar het resultaat te kijken:

In niet mis te verstane woorden wordt je hier duidelijk gemaakt dat deze web service nog steeds de http://tempuri.org namespace gebruikt, en er staan voor C# of VB voorbeelden bij hoe dit op te lossen is. Dat is niet moeilijk om te zetten naar een Delphi Prism compatible syntax, zoals we zullen zien.
Alhoewel de namespace erg op een URL lijkt, is het er geen eentje. Het is een URL (universal resource identifier – vandaar ook de naam: temp uri), en het hoeft dus niet als URL te interpreteren te zijn, maar dat mag wel. Voor mijn eigen voorbeelden en daadwerkelijk beschikbare web services gebruik ik altijd de http://eBob42.org namespace. En dat dat toevallig ook de URL is van mijn website is mooi meegenomen, maar niet meer dan dat.

```
type
  [WebService(&Namespace := 'http://eBob42.org/')]
```

Merk op dat namespace een gereserveerd woord is in de Delphi Prism taal, en om die reden moeten we er een & voor zetten zodat het als property van het WebService attribuut gebruikt kan worden.
Los van de namespace property, kan de WebService attribuut ook gebruikt worden om de omschrijving van de web service op te geven via de Description property. De description wordt niet gebruikt door compilers en importers, maar is nog wel te lezen in de WSDL die geproduceerd wordt, en is al met al een nuttige toevoeging die door mensenogen te lezen is als extra informatie.

```
type
  [WebService(&Namespace := 'http://eBob42.org/',
    Description := 'This web service is a first demo Web Service ' +
      'generated by Delphi Prism for ASP.NET 2.0 or higher')]
```

Als je een hele lange Description wilt opgeven, dan kun je gebruik maken van HTML Tags zoals <gr> en <p> om die mooi zichtbaar te laten worden in de browser (de gebruikelijke manier waarom mensen naar WSDL definities kijken).

[WebMethod]

De gegenereerde web service class Service bevat al een methode genaamd HelloWorld. Dit is een zogenaamde “web method” die door de web service wordt geexporteerd. Helaas is het niet meteen te zien als we naar de definitie van de Service class kijken, dat HelloWorld een web method is, omdat het [WebMethod] attribuut bij de implementatie van HelloWorld staat, en niet bij de definitie. Het is nodig om [WebMethod] neer te zetten bij een methode om hem te exporteren, en [WebMethod] kan of bij de definitie (in de class) of bij de implementatie gezet worden, maar niet op beide plaatsen tegelijk. Mijn voorkeur gaat duidelijk uit naar het gebruik van [WebMethod] bij de definitie, zodat aan de class Service al meteen te zien is welke methodes web methods zijn en welke niet. Het is dan ook minder nuttig (en een beetje jammer eigenlijk) dan Delphi Prism de [WebMethod] in het voorbeeld pas bij de implementatie van HelloWorld zet.
Mijn advies is dan ook om het attribuut [WebMethod] te verwijderen van de implementatie van HelloWorld, en juist neer te zetten in de class bij de definitie van HelloWorld, als volgt:

```
namespace;

interface
uses
  System,
  System.Web,
  System.Collections,
  System.Web.Services,
  System.Web.Services.Protocols;

type
  [WebService(&Namespace := 'http://eBob42.org/',
    Description := 'This web service is a first demo Web Service<br>' +
      'generated by <b>Delphi Prism</b> for ASP.NET 2.0 or higher')]
  [WebServiceBinding(ConformsTo := WsiProfiles.BasicProfile1_1)]
  Service = public class(System.Web.Services.WebService)
  public
    [WebMethod]
    method HelloWorld: string;
  end;
```

```
implementation

method Service.HelloWorld: string;
begin
  result := 'Hello World';
end;

end.
```

WebMethod properties

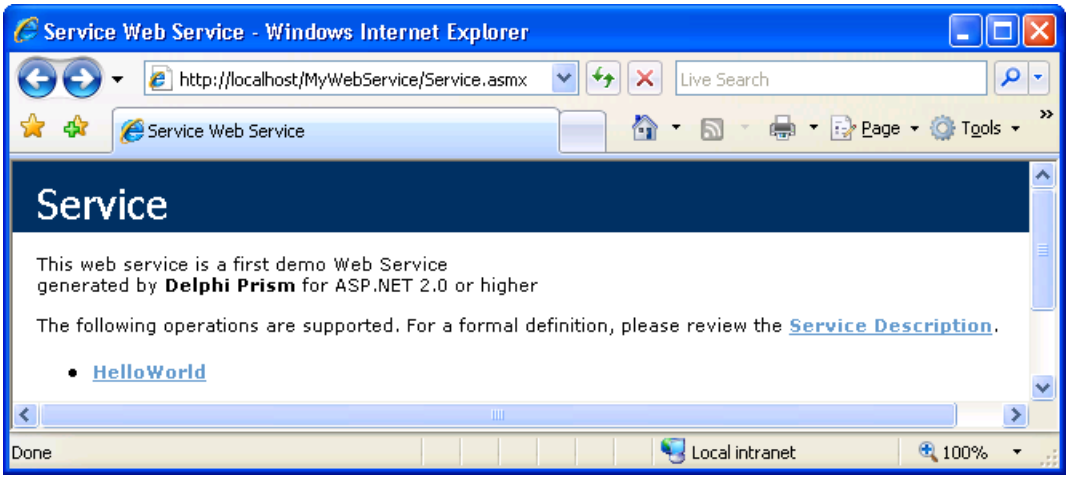
Het WebMethod attribuut heeft een zestal properties die we kunnen gebruiken indien nodig, namelijk Description (voor het meegeven van een beschrijving van de method), MessageName (voor het veranderen van de naam waaronder de message verstuurd wordt), Buffer Response (voor het al dan niet bufferen van het resultaat), CacheDuration (voor het cachen van een antwoord voor hergebruik), TransactionOption (voor transactie support) en EnableSession.
Net als bij het WebService attribuut, kan de Description property van de WebMethod gebruikt worden om een korte beschrijving toe te voegen die menselijke ontwikkelaars kunnen lezen en die bijvoorbeeld informatie bevat over de parameters van de WebMethod zelf en wat ze terug kunnen verwachten als resultaat.
MessageName, BufferResponse, CacheDuration en TransactionOption vallen buiten de scope van dit artikel, maar EnableSession is nog wel interessant genoeg om even te behandelen.

EnableSession

EnableSession staat standaard op False, wat de meest effciente setting is. Je moet EnableSession alleen op True zetten als het gebruik van de ASP.NET Session noodzakelijk is voor deze method (het is een property die je per methode aan moet zetten).
De ASP.NET Application en Session objecten zijn beschikbaar voor ASP.NET ontwikkelaars via de WebService base class. Het Application object is altijd beschikbaar en kan gebruikt worden om key=value waardes in te stoppen – maar iedereen kan ze zien (lezen en schrijven), dus het is niet geschikt om gebruiker (sessie) specifieke informatie in te stoppen. Daarvoor kan het Session object gebruikt worden, dat echter wel voor overhead zorgt doordat het dan voor iedere session aangemaakt en beheerd wordt (gedurende de lifetime van de session). Dat zorgt met name voor een overhead in geheugengebruik van de web service die hierdoor minder schaalbaar wordt. Hierdoor staat EnableSession default op False, maar kunnen we hem op True zetten om voor bepaalde web methods het gebruik van de Session mogelijk te maken.

Eerste Test

Zonder dat we nog een regel code hoeven te schrijven kunnen we de huidige ASP.NET Web Services al testen – in ieder geval vanaf de locale ontwikkelmachine zelf. Nadat we de solution gebuild hebben, kunnen we namelijk de Web Service in een browser bekijken via de volgende URL: http://localhost/MyWebService/Service.asmx



We kunnen zelfs klikken op de link voor de HelloWorld methode om deze te testen in de locale browser. Maar omdat het kijken naar de “Hello, World” string niet echt boeiend is, zal ik dat bewaren tot we wat meer functionaliteit hebben toegevoegd aan de web service. Als real-world voorbeeld vraag ik even de aandacht voor een speciale Num2Word methods die gebruikt kan worden om getallen in woorden uit te schrijven. Deze routine wordt veel gebruikt voor wettelijke documenten, of bankcheques om fraude te voorkomen.
De defnitie van de Service class met de methode Num2Word is als volgt (ik heb het [WebMethod] attribuut nu bij de definitie van beide methods gebruikt).

```
type
  [WebService(&Namespace := 'http://eBob42.org/',
    Description := 'This web service is a first demo Web Service ' + '<br>' +
      'generated by <b>Delphi Prism</b> for ASP.NET 2.0 or higher')]
  [WebServiceBinding(ConformsTo := WsiProfiles.BasicProfile1_1)]
  Service = public class(System.Web.Services.WebService)
  public
    [WebMethod]
    method HelloWorld: string;
    [WebMethod]
    method Num2Word(value: Integer): String;
  end;
```

De implementatie – voor het vertalen van nummers naar woorden – is als volgt:

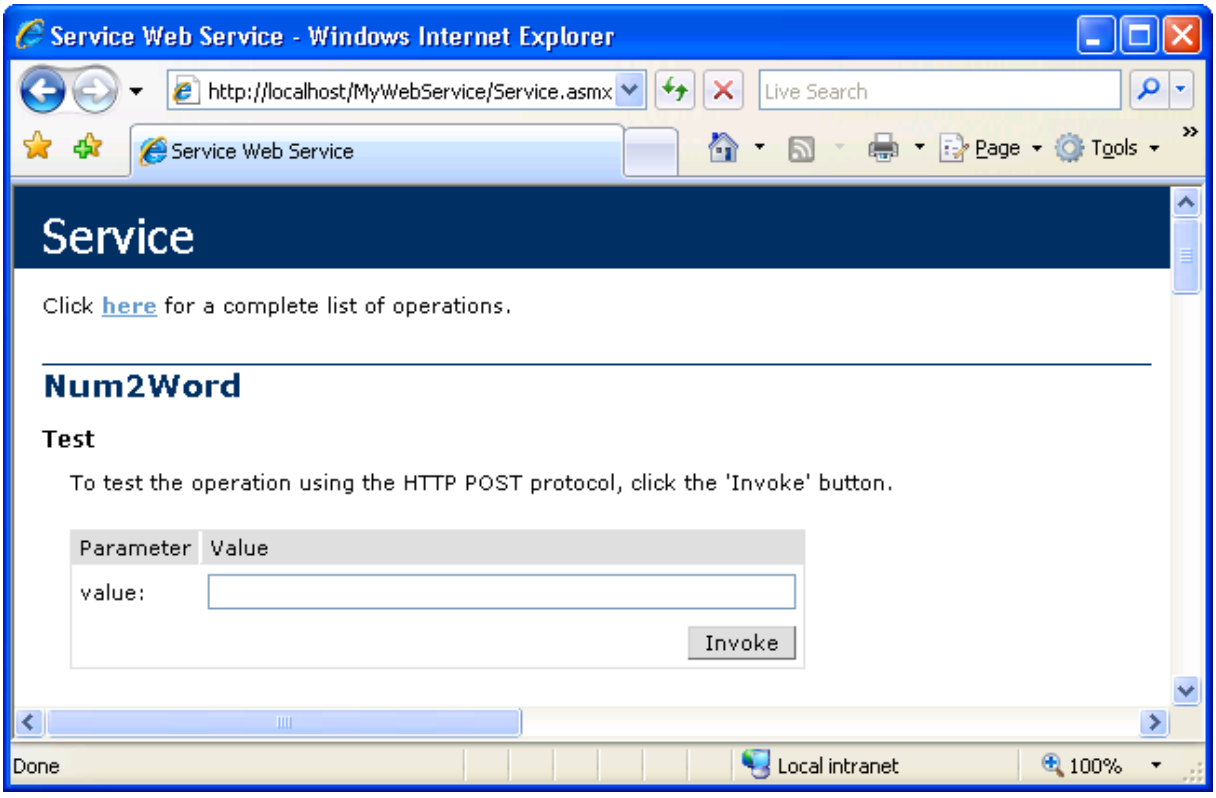
```
method Service.Num2Word(value: Integer): String;
begin
  if value >= 1000000 then
    if (value mod 1000000) = 0 then
      Result := Num2Word(value / 1000000) + 'miljoen'
    else
      Result := Num2Word(value / 1000000) + 'miljoen ' +
                Num2Word(value mod 1000000)
  else
    if value >= 1000 then
      if (value mod 1000) = 0 then
        Result := Num2Word(value / 1000) + 'duizend'
      else
        Result := Num2Word(value / 1000) + 'duizend ' +
                  Num2Word(value mod 1000)
    else
      if value >= 100 then
        if (value mod 100) = 0 then
          Result := Num2Word(value / 100) + 'honderd'
        else
          Result := Num2Word(value /  100) + 'honderd' +
                    Num2Word(value mod 100)
      else
        case (value div 10) of
 5,6,7,9: if (value mod 10) = 0 then
            Result := Num2Word(value / 10) + 'tig'
          else
            Result := Num2Word(value mod 10) + 'en' +
                      Num2Word(value / 10) + 'tig';
       8: if value = 80 then
            Result := 'tachtig'
          else
            Result := Num2Word(value mod 10) + 'entachtig';
       4: if value = 40 then
            Result := 'veertig'
          else
            Result := Num2Word(value mod 10) + 'enveertig';
       3: if value = 30 then
            Result := 'dertig'
          else
            Result := Num2Word(value mod 10) + 'endertig';
       2: if value = 20 then
            Result := 'twintig'
          else
            Result := Num2Word(value mod 10) + 'entwintig';
     0,1: case value of
            0: Result := 'nul';
            1: Result := 'een';
```

```
                2: Result := 'twee';
                3: Result := 'drie';
                4: Result := 'vier';
                5: Result := 'vijf';
                6: Result := 'zes';
                7: Result := 'zeven';
                8: Result := 'acht';
                9: Result := 'negen';
               10: Result := 'tien';
               11: Result := 'elf';
               12: Result := 'twaalf';
               13: Result := 'dertien';
               14: Result := 'veertien';
               15: Result := 'vijftien';
               16: Result := 'zestien';
               17: Result := 'zeventien';
               18: Result := 'achttien';
               19: Result := 'negentien'
             end
           end
end;
```

Delphi ontwikkelaars die recentelijk pas de stap naar Delphi Prism hebben gemaakt voor .NET ontwikkeling (of Delphi Prism nog niet gebruiken maar toch dit artikel lezen), zullen wellicht enige opvallende dingen hebben gezien in de implementatie van Num2Word. Niet alleen gebruik ik weer method in plaats van function (of procedure), maar ik gebruik ook de / voor het delen van twee integers in plaats van div. Bij Delphi for Win32 levert de / altijd een floating point getal op, maar bij Delphi Prism krijg je een integer terug als beide getallen ook een integer zijn – net als de div operator. Let op: in Delphi Prism is 5 / 3 dus gelijk aan 1, en niet 1,66666. Om dat laatste te krijgen moet je 5 / 3.0 doen of in het geval van integer variabelen eerst een vermenigvuldiging met 1.0 doen.

ASP.NET Test

Met deze methods erbij kunnen we de MyWebService solution opnieuw compileren en in de browser bekijken. Deze keer zullen er twee methods te zien zijn, en als we op de link voor de Num2Word method klikken komen we in een speciale pagina waarin we een waarde voor de "value" parameter kunnen invoeren en op de Invoke knop kunnen klikken om de Num2Word methods uit te voeren (met de waarde van value in het SOAP request).



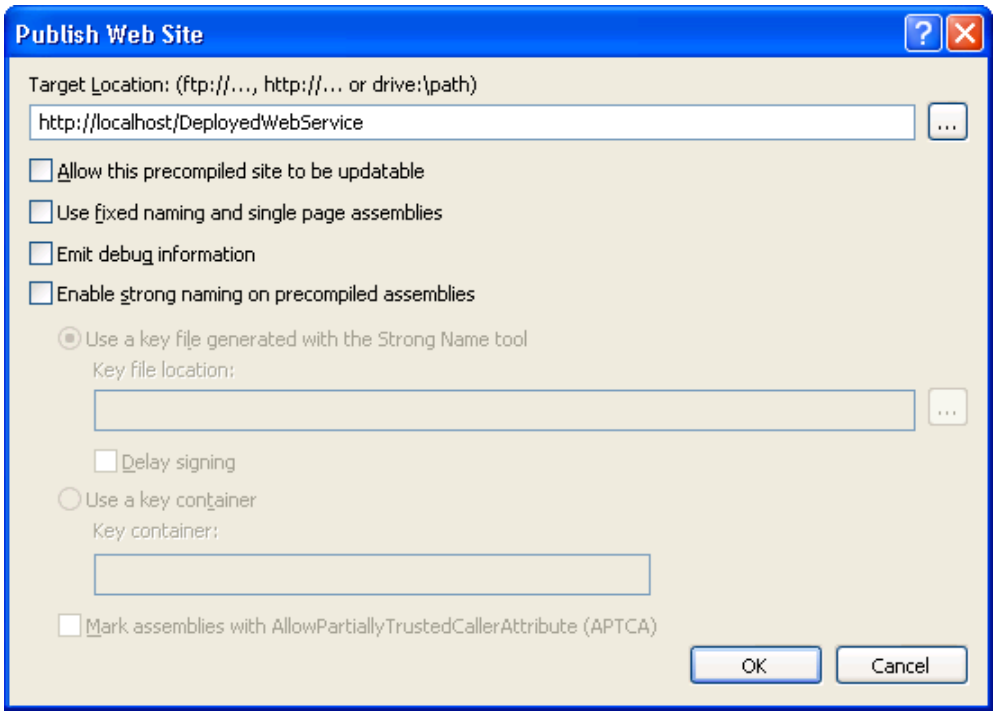
Als ik als voorbeeld 1964 invoer en op Invoke klik, dan krijgen we het volgende resultaat te zien in de SOAP response:

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://eBob42.org/">eenduizend negenhonderdvierenzestig</string>
```

Dit laat meteen de beperking zien van de Num2Word method, omdat ik er vrij zeker van ben dat de meeste mensen 1964 zullen lezen als negentien(honderd)vierenzestig in plaats van het resultaat dat we hierboven zien. Maar dat is nu eenmaal de norm als het gaat om bedragen in plaats van jaartallen.
Het voordeel van testen via de browser is dat je niet eerst een aparte client moeten schrijven. Let op dat we de browser alleen kunnen gebruiken om de web service te testen als de web service op dezelfde machine draait. En bovendien kunnen we alleen "simpele" waardes invullen voor bijvoorbeeld integers, strings en floating points, maar niet voor complex types zoals datasets or structures.

Deployment

Los van het draaien van de web service in een locale browser en vanuit de IDE, zul je op een bepaald moment de web service ook op een echte web server willen deployen. Dit kan via de Publish Web Site optie (te vinden als je met de rechter muisknop klikt op het MyWebService project in de Solution Explorer). Je krijgt dan een dialoog waar we een aantal deployment options kunnen invullen:

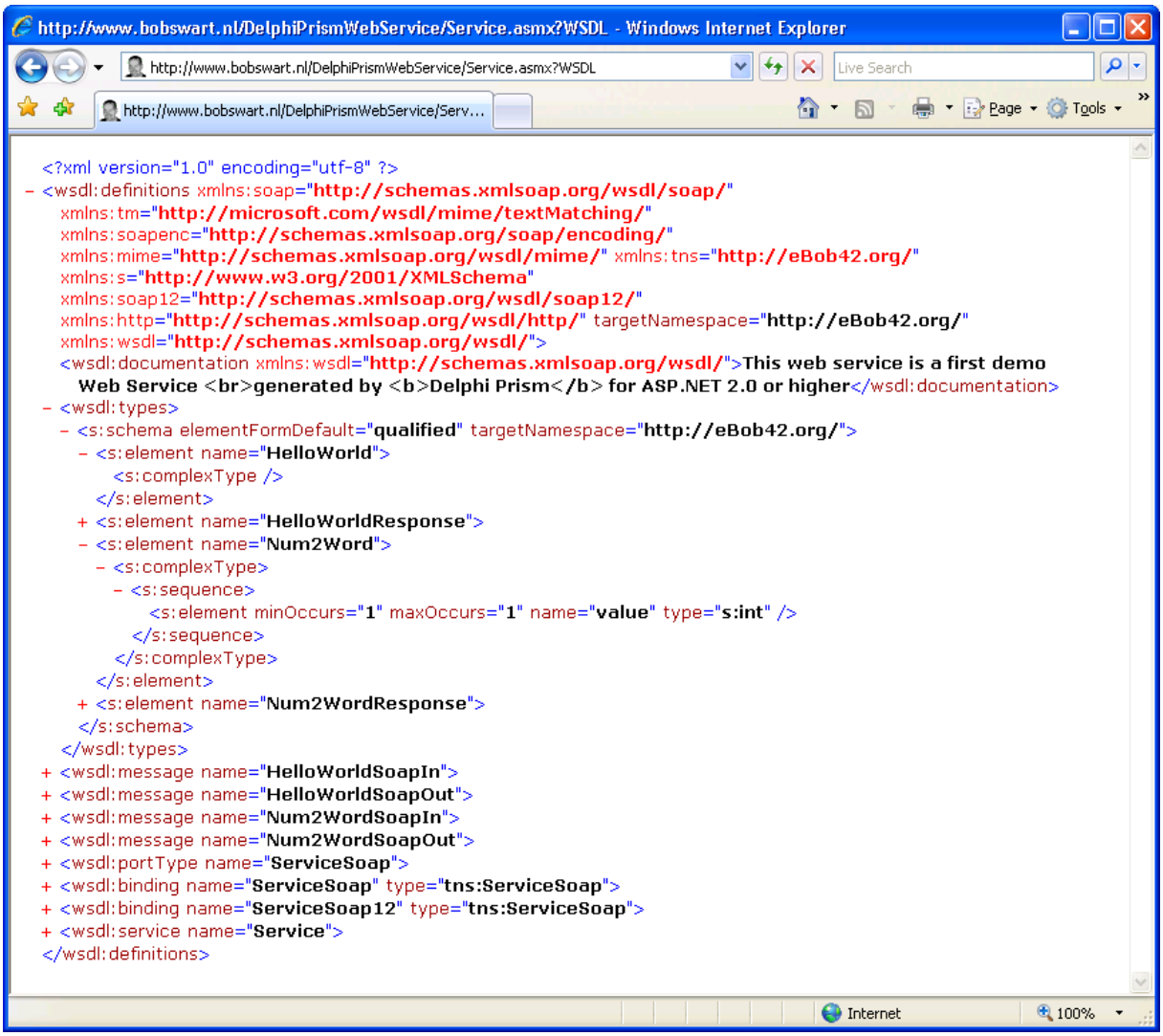


Belangrijk is de optie "Allow this precompiled site to be updatable". Alhoewel deze optie ons de mogelijkheid biedt om de site te updaten op de web server, moeten we dan ook de Delphi Prism (Oxygene) command-line compiler deployen op de web server. Dit zal ik deze keer niet doen, en is vaak ook niet nodig, maar dan moeten we wel de optie uit laten staan. Het resultaat is dan dat een virtual directory DeployedWebService onder de localhost op mijn ontwikkelmachine zal komen (te vinden in C:\Inetpub\wwwroot\DeployedWebService) met daarin de Service.asmx file, een PrecompiledApp.config en Web.config alsmede een bin directory met de gecompileerde App_Code.dll code behind assembly en een App_Code.compiled en service.asmx.cdcab7d2.gecompileerd bestand. Deze 6 files en ene directory zijn alles wat nodig is om de web service te deployen op een andere machine (zonder daarbij Delphi Prism of Oxyegen te moeten installeren op die betreffende machine).
Ter demonstratie heb ik een virtual directory genaamd DelphiPrismWebService op mijn www.bobswart.nl web server aangemaakt en daar de inhoud van de DeployedWebService naartoe gekopieerd.. Er is geen Delphi Prism (command-line compiler) op mijn web server ge:installeerd, maar zonder verdere aanpassingen kunnen we de Delphi Prism Web Services benaderen via http://www.bobswart.nl/DelphiPrismWebService/Service.asmx

Merk op dat Delphi Prism is niet nodig op mijn web server, maar ik moest wel zorgen dat het Microsoft .NET Framework 2.0 of hoger (zoals 3.5) geïnstalleerd is, afhankelijk van de keuze die in de eerste screenshot gemaakt is.

WSDL

Zodra de web service gedeployed is, kunnen we de zogenaamde "formale defnition" bekijken, die bestaat uit de Web Service Definition Language of WSDL. Voor ASP.NET web services krijgen we die door ?WSDL achter de oorspronkelijke URL zelf te plakken, zodat we in dit geval de nieuwe URL http://www.bobswart.nl/DelphiPrismWebService/Service.asmx?WSDL krijgen voor mijn voorbeeld. Het resultaat kunnen we in de browser bekijken als volgt:

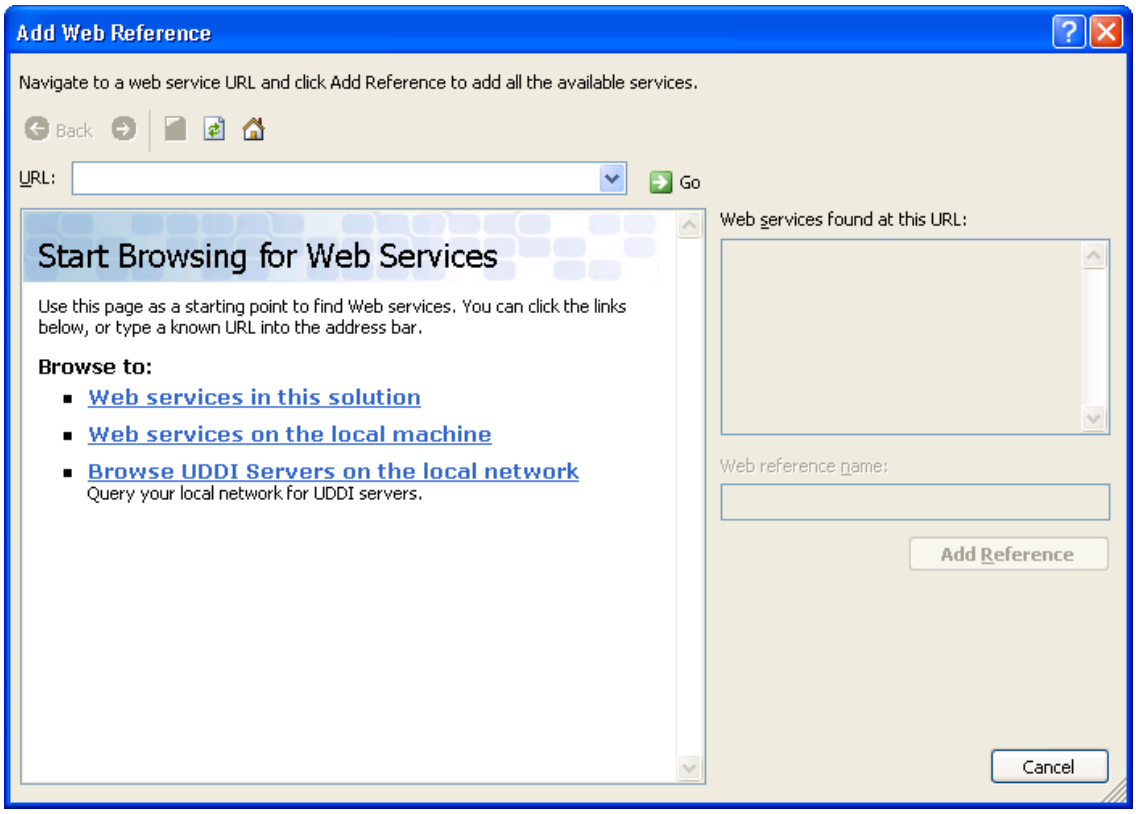


Merk op dat de description ook in de WSDL staat binnen de wsdl:documentation tag, inclusief de HTML tags die ik had toegevoegd.

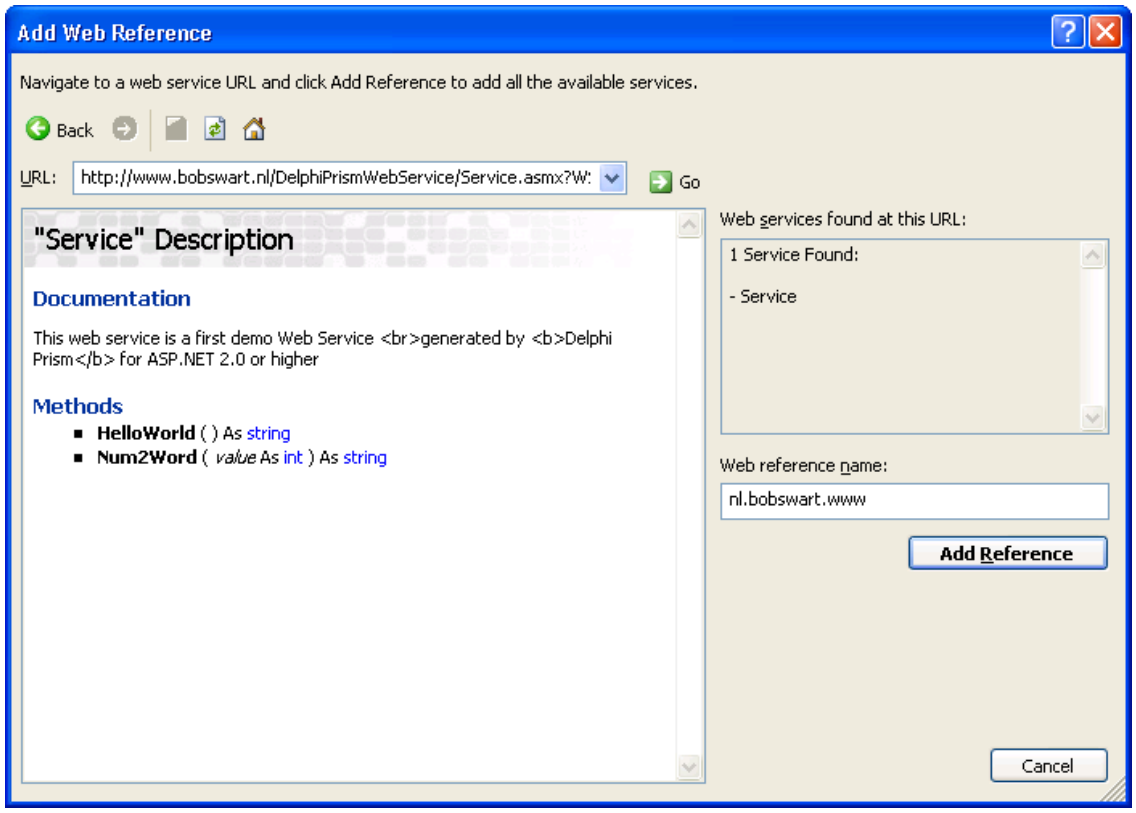
Web Service Client

Om een web service client te maken, kunnen we vanuit Delphi Prism de WSDL importeren (ook wel consumeren genoemd). Dit kunnen we doen voor ongeveer ieder Delphi Prism project type. Voor dit artikel kunnen we een WinForms GUI gebruiken. Dit kan door met de rechtermuisknop te klikken op de solution, Add | New Project te kiezen, en dan in de Delphi Prism, Windows category, te kiezen voor een Windows Application. Na het opgeven van MyWebServiceClient als Naam kunnen we in het nieuwe project een referentie aanmaken naar de web service. Hiervoor moeten we wederom met de rechtermuisknop klikken, deze keer op de References knoop, en dan kiezen voor Add Web Reference.

In de dialog die volgt kunnen we browsen naar web services in de huidige solution, op de local machine, of gebruik maken van UDDI servers op het locale netwerk.



In plaats daarvan geef ik echter meestal gewoon de URL op van de WSDL locatie, zodat de WinForms client echt werkt met de gedeployde web service (en niet alleen met de demo versie).
De URL was http://www.bobswart.nl/DelphiPrismWebService/Service.asmx en als ik die invoer dan zien we meteen de service description:



In deze dialoog zien we niet alleen de leesbare description (deze keer zonder de HTML effecten), maar ook de twee methods (en description als we die hadden toegevoegd). De web reference name zal de namespace van het project (MyWebServiceClient) worden,g evolgd door de URL van het domain in omgekeerde volgorde (nl.bobswart.www in dit geval).

Na een klik op de Add Reference knop wordt de nieuwe Web Reference knoop toegevoegd aan het project, met een subknoop genaamd nl.bobswart.www twee sub-subnodes genaamd Reference.map end Service.wsdl, met een file Reference.pas als blad van de Reference.map. Deze Reference.pas, bevat de Delphi Prism import onder de MyWebServiceClient.nl.bobswart.www namespace.

Om deze geïmporteerde web service te gebruiken hebben we alleen maar een Button nodig op de Main form en een paar regels code (let op dat de namespace ook moet worden toegevoegd aan de uses clause):

```
implementation
uses
  MyWebServiceClient.nl.bobswart.www;

method MainForm.button1_Click(sender: System.Object; e: System.EventArgs);
var
  MyService: Service;
begin
  MyService := new Service();
  MessageBox.Show( MyService.Num2Word(1964) );
end;
```

In de eerste regel code maken we de instantie aan van de Service class, en vervolgens kunnen we die gebruiken alsof het een lokaal object is. Iedere aanroep van een methode van MyService wordt echter als SOAP request over de lijn gestuurd en komt als SOAP response terug. En ook hierbij krijgen we de waarde van 1964 terug als resultaat.

Summary

In dit artikel heb ik laten zien hoe we met Delphi Prism een ASP.NET Web Service kunnen maken, en welke attributen daarbij van belang zijn. Ook heb ik laten zien hoe we deze web services kunnen testen en deployen. Vervolgens heb ik laten zien hoe we bestaande web services kunnen importeren en consumeren in Delphi Prism client toepassingen.
Vervolgartikelen over dit onderwerpen zijn te lezen via http://www.drbob42.com/examines/ , waarin ik o.a. laat zien hoe we SOAP security in de vorm van SOAP Headers kunnen toevoegen aan de server en client toepassing, en hoe we gebruik kunnen maken van het ASP.NET Session object.

Wie nog vragen of opmerkingen heeft, of Delphi Prism zelf een keer wil proberen, kan me altijd per e-mail bereiken of me een bezoek komen brengen in Helmond Brandevoort (zie www.eBob42.com voor details).

Tijdens de eerstvolgende gratis Delphi workshop in Helmond Brandevoort op vrijdag 11 september zal ik ook weer zowel Delphi for Win32 als Delphi Prism onderwerpen aan de orde laten komen. Zie mijn website of weblog te http://www.bobswart.nl/blog binnenkort voor de details.

Delphi Prism 2009 Update

In dit artikel heb ik ASP.NET Web Services gebouwd met Delphi Prism in de vorm van een ASP.NET Web Site. Sinds de Februari 2009 update van Delphi Prism is het echter ook mogelijk om een ASP.NET Project te starten, en daar de ASP.NET Web Service aan toe te voegen. Het grootste verschil tussen de ASP.NET Web Site en ASP.NET Web Project zit hem in de Code Beside vs. Code Behind benadering, en de daarmee gepaarde deployment mogelijkheden.
Tijdens mijn Delphi Prism ASP.NET sessie op de SDE van 26 juni zal ik hier ook dieper op ingaan!

Delphi Prism, Async, Futures en Parallelle mogelijkheden

Delphi Prism is de nieuwe .NET variant van Delphi, en bestaat uit drie onderdelen. Allereerst is daar de Oxygene compiler, gemaakt en onderhouden door RemObjects Software (destijds uitgebracht onder de naam Chrome). Deze Oxygene compileer draait als een plug-in in de Microsoft Visual Studio IDE (de gratis Shell of Express editie is voldoende), en levert daarmee als tweede onderdeel alle designers voor WinForms, WPF, ASP.NET, Silverlight, etc. Behalve een Compact Framework designer dan, omdat die nog steeds vastgebakken zit aan C# of VB. Het derde onderdeel van Dephi for .NET is het deel dat van CodeGear afkomstig is: de database connectivity in de vorm van de dbExpress drivers en DataSnap mogelijkheden. Op dit moment beperkt tot InterBase en BlackfishSQL voor dbExpress drivers, en alleen nog DataSnap clients, maar in toekomstige uitbreidingen kunnen we extra dbExpress drivers en DataSnap server functionaliteit verwachten.

Async

De Oxygene compiler van Delphi Prism ondersteung een taal die het async keyword bevat. Dit keyword kan gebruikt worden om bij een method aan te geven dat deze asynchroon moet worden uitgevoerd; in een aparte thread dus. De declaratie van een asynchrone method kan als volgt zijn:

```
method DoeIetsMetInputParameters(x,y,z: Integer); async;
```

Let op dat we in Delphi Prism geen procedure of function meer hoeven te schrijven, maar het generiekere method kunnen gebruiken. En voor een asynchrone method voegen we het keyword async toe aan het eind van de declaratie.
Zoals de naam van de method al doet vermoeden, heeft een async method een voordeel maar ook een aantal beperkingen. Het grote voordeel is uiteraard dat de main thread meteen terugkeert na de aanroep van de async method. De async method zal in een aparte thread uitgevoerd worden, waardoor de main thread meteen verder kan. Nadeel daarbij is dat de main thread niet zal weten wanneer de async method is afgelopen. Daarnaast kan de method alleen maar input parameters bevatten, en geen resultaat terugeven omdat deze natuurlijk pas bekend worden tijdens of na afloop vna het uitvoeren van de async method. En de main thread die de async method aanroept die zou niet kunnen weten wanneer het resultaat daadwerkelijke beschikbaar is.
Dit leidt al snel tot de vraag waar en wanneer een async method dan nuttig kan zijn. In praktijk gebruik ik het veel in situaties waar je iets wilt doen dat verder zonder interactie kan worden afgewerkt. Het genereren en/of printen van een report bijvoorbeeld, of het versturen van een e-mail. Dan is het ideaal dat de aanroep van de async method geen verdere tijd in beslag neemt en de main thread gewoon doorgaat.

Async Code

Behalve async methods, kunnen we het async keyword ook gebruiken om van code aan te geven dat deze asynchroon (in een aparte thread) moet worden uitgevoerd. Dit kan bijvoorbeeld als volgt:

```
class method ConsoleApp.Main;
var
  x: Integer;
begin
  x := 42;

  async begin
    x := x + x;
    Console.WriteLine('x = ' + x.ToString);
  end;

  x := x / 2;
  Console.WriteLine('x = ' + x.ToString);
  Console.ReadKey
end;
```

Ook hierbij geldt dat het async code block geen persistente wijzigingen kan aanbrengen in lokale variabelen. In bovenstaand voorbeeld wordt de variabele X echter wel aangepast. Om hiermee (concurrency) problemen te voorkomen, wordt voordat het async code block start eerst de gehele context gekopieerd. In dit geval is dat alleen de variabele X, maar dat kan in praktijk een stuk meer bevatten.
De output van het stuk code is dan ook x = 84, en x = 21. Misschien in deze volgorde, of wellicht andersom (de x = 21 eerst). Ze kunnen in theorie zelfs doorelkaar geschreven staan, maar dat heb ik zelf nog niet kunnen reproduceren.
Behalve het feit dat de variabele x in het async code block niet de echte x is, maar een kopie, is het ook niet gegarandeerd wanneer het async code block klaar is. Maar daar is wel een oplossing voor: we kunnen het gehele async code block toekennen aan een System.Action variabele en die gebruiken om op een bepaald moment deze variabele als statement aan te roepen waarna – indien het async code block nog niet klaar is – gewacht wordt tot deze thread afgerond is. Dat gaat dan als volgt:

```
class method ConsoleApp.Main;
var
  x: Integer;
begin
  x := 42;

  var xtask := async begin
    x := x + x;
    Console.WriteLine('x = ' + x.ToString); // 21
  end;

  x := x / 2;
  Console.WriteLine('x = ' + x.ToString); // 84
  xtask(); // wachten tot de thread klaar is
  Console.ReadKey
end;
```

De aanroep van xtask(); zal er nu voor zorgen dat er op dat moment gewacht wordt – indien nodig – tot het async code block dat we aan xtask hebben toegewezen is uitgevoerd. Die truc kunnen we helaas niet uithalen met async methods, maar alleen met async code blocks.

Futures

Nadeel van een async code block blijft het feit dat je wel iets kunt doen of uitrekenen, maar het resultaat niet meer kunt "terugzetten" naar de main thread. Het async code block wordt als het ware in een eigen wereldje uitgevoerd, met een kopie van de omgeving.
Als we wel nog iets terug willen geven, maar ook gebruik willen maken van de kracht van asynchrone code, dan moeten we een andere Delphi Prism feature gebruiken: het future keyword. Door gebruik te maken van dit keyword, geven we aan dat een variabele een waarde heeft die op dit moment nog niet bekend is (bijvoorbeeld omdat de toekomstige waarde ervan in een async code block of method wordt berekend), maar op het moment dat we hem nodig hebben, zal de waarde er zijn (of blijven we erop wachten).
Stel bijvoorbeeld dat we voor een hypotheekberekening twee maandbedragen bij elkaar moeten optellen: het rente bedrag en het premie bedrag. Beide bedragen zijn afkomstig uit een wellicht complexe berekening, en het zou zonde zijn om die op elkaar te laten wachten. Zeker als je een dual-core of multi-processor machine hebt en een deel ervan toch niks staat te doen.
De syntax van het gebruik van futures voor dit voorbeeld is als volgt:

```
method MainForm.BerekenMaandbedrag: Double;
var
  RenteBedrag: future Double;
  PremieBedrag: future Double;
  VerborgenKosten: Double;
begin
  RenteBedrag := async BerekenRenteBedrag;
  PremieBedrag := async BerekenPremieBedrag;
  VerborgenKosten := BerekenVerborgenKosten;
  Result := RenteBedrag + PremieBedrag + VerborgenKosten
end;
```

Zowel de waarde van RenteBedrag als die van PremieBedrag worden in aan aparte thread berekend. Door dat op deze manier te doen zijn de waardes waarschijnlijk nog niet bekend als we aan de verborgen kosten berekening gaan beginnen. Maar dat maakt niet uit, want op het moment dat we alles nodig hebben (bij het Result statement) zal blijken of de async code al klaar is, en de future variabelen hun waarde al hebben, of dat we daar nog even op moeten wachten. Je zou in de code eventueel een soort indicator op het scherm aan kunnen zetten ("waiting…") voordat je de waarde van een future variabele ophaalt (of erop blijft wachten), zodat de gebruiker in ieder geval door heeft wat er aan de hand is als we hier enige tijd moeten wachten tot de waarde van de future bekend is.

In feite is de xtask variabele die we eerder gebruikte om op een async code block te wachten ook een future. En het "aanroepen" van de variabele is niets anders dan het gebruiken van de waarde, waardoor we blijven wachten indien deze (het resultaat van het code block) nog niet bekend is.

PFX Framework: Threads vs. Tasks

De async voorbeelden runnen in een eigen thread, in de .NET thread pool. Echter, de volgende versie van het .NET Framework bevat een PFX Framework voor Parallelle Extensies, en als we dat erbij gebruiker zal de async method of code als een echte parallelle task draaien in plaats van alleen maar een nieuwe thread in the thread pool.

Parallel

Een nieuwe feature die verbonden is aan het PFX is het parallel keyword. Hiermee kunnen we aangeven dat een stuk code parallel uitgevoerd moet worden: dus bijvoorbeeld een fot-loop die in stukken geknipt zal worden.

```
method MainForm.Doe;
begin
  for parallel i: Integer := 0 to 10 do
  begin
    BerekenIetsMetI(i);
  end;
end;
```

Deze code zal alleen compileren als het PFX wordt aangetroffen (de benodigde assemblies zijn nodig), en zal ook alleen draaien op machines waar de PFX aanwezig is.
Meer experimenten met FPX en het parallel keyword (waarin we zullen zien hoe de loop in stukken wordt geknipt, en wat er met de i gebeurt) zullen volgen als ook de ontwikkeling aan .NET 4.0 en PFX wat meer gevorderd is. Het goede nieuws is dat Delphi Prism nu al deze komende features ondersteunt.

Conclusie

In dit artikel heb ik laten zien hoe enkele taalelementen van Delphi Prism werken – taalelementen zoals async en futures die overigens al enige tijd in de Oxygene compiler zaten, dus niet zo heel verschrikkelijk nieuw zijn. Het leuke is in ieder geval dat Delphi for .NET niet langer "achterloopt" op het gebied van .NET features, maar juist nu al gebruik maakt van functionaliteit die nog niet eens af is of volledig beschikbaar.
Wie nog vragen of opmerkingen heeft, of Delphi Prism zelf een keer wil proberen, kan me altijd per e-mail bereiken of me een bezoek komen brengen in Helmond Brandevoort (zie www.eBob42.com voor details).

Tijdens de eerstvolgende gratis Delphi workshop in Helmond Brandevoort op vrijdag 11 september zal ik ook weer zowel Delphi for Win32 als Delphi Prism onderwerpen aan de orde laten komen. Zie mijn website of weblog te http://www.bobswart.nl/blog binnenkort voor de details.

Delphi boeken op Lulu.com

De afgelopen tijd heb ik enkele Delphi cursusboeken omgezet tot een PDF formaat dat bruikbaar was om er door Lulu een boekje van te laten maken. Ik heb twee verschillende formaten aangemaakt: die van maximaal 88 pagina's met een nietje in de rug (de saddle-stitch versie) en de versie vanaf 68 pagina's (maximaal 680) met een zijkaft waarop de titel gedrukt kan worden.

De volgende Delphi titels zijn nu te koop van Lulu.com:

- Delphi 2009 Development Essentials (127 pagina's) – €24
- Delphi 2007 for Win32 Development Essentials (144 pagina's) - €24
- Delphi 2007 for Win32 VCL Database Development (132 pagina's) - €24
- Delphi 2007 for Win32 VCL for the Web Development (142 pagina's) - €32
- Delphi Win32 / .NET Component Development (175 pagina's) - €32
- Delphi 2006 Development Essentials (238 pagina's) - €32
- Delphi 2006 ADO.NET 1.1 Database Development (169 pagina's) - €24
- Delphi 2006 ASP.NET 1.1 Web Development (218 pagina's) - €24
- Delphi 2006 Advanced ASP.NET 1.1 Web Development (135 pagina's) -€24
- Delphi 2006 XML, SOAP and Web Serv ices (234 pagina's) - €32
- Delphi 2005 Development Essentials (187 pagina's) - €24

Zie http://stores.lulu.com/drbob42 voor een compleet overzicht van deze en de andere publicaties die ik op Lulu.com heb gezet, of http://www.eBob42.com/courseware voor de cursusboeken die nog niet op Lulu.com staan omdat ik er nog regelmatig training mee geef en uitbreidingen op aanbreng. Deze laatste cursusboeken verkoop ik wel in PDF formaat, maar voor een hogere prijs (99 Euro, ex. BTW), wat dan weer wel inclusief updates en eventuele e-mail ondersteuning is (over de onderwerpen die in het boek worden behandeld). Zie http://www.eBob42.com/courseware voor een recente lijst met titels.

Delphi support per e-mail

Naast het geven van training & consultancy (en het spreken op seminars en schrijven van artikelen), ben ik ook beschikbaar voor het geven van structurele ondersteuning per e-mail. Dit betekent dat Delphi ontwikkelaars mij per e-mail vragen en problemen kunnen voorlegen. De prijs is 100 Euro per (kalender)maand, met een korting voor wie meteen voor een kalenderjaar betaald; dan is de prijs 1000 Euro, ex. BTW. Voor die prijs kan een ontwikkelaar mij vragen per e-mail stellen die ik dan "snel" en in detail zal proberen te beantwoorden. Meestal zal "snel" betekenen binnen een paar uur, vrijwel altijd nog dezelfde dag. Ik kan helaas geen garantie geven dat ik alle vragen zal kunnen beantwoorden, maar zal wel altijd proberen om relevante informatie of verwijzingen op te zoeken. Mocht de "service" niet bevallen, dan kan er altijd aan het eind van een periode gestopt worden (maar er vind geen restitutie plaats van betaalde bedragen).
Indien een vraag extra uitzoekwerk zou verlangen, dan meld ik dat eerst (plus een schatting hoeveel tijd (en geld) dat extra uitzoeken met zich mee zou brengen). Je kan dan zelf bepalen of het de moeite waard is om mij dat uit te laten zoeken. Iemand heeft mij bijvoorbeeld wel eens laten uitzoeken hoe je een bepaald "vingerafdruk"-scan apparaat kunt gebruiken, waarbij alleen een DLL en een C header file beschikbaar was. Dat is in de vorm van een klein project van enkele dagen uitgevoerd. Tot volle tevredenheid van beide partijen overigens.

Marco Cantù is pas geleden ook met een vergelijkbare service begonnen. Alleen doet hij het niet per e-mail, maar in private nieuwsgroepen, en doet hij het niet alleen maar heeft hij een team van zo'n 10 personen die de vragen beantwoorden.
Bij mij krijgt u altijd alleen antwoord van mijzelf en per e-mail. Omdat ik het alleen doe zijn er echter maar een beperkt aantal "plaatsen" beschikbaar. Mail Bob@eBob42.com voor meer informatie.

Delphi bestellen bij Bob Swart 

Onderstaande prijzen zijn exclusief 19% BTW en gelden voor de Electronic Software Delivery (ESD) edities, die u kunt downloaden en kunt installeren met een bijbehorend serienummer dat u ontvangt. Een ESD licentie betreft een download – u kunt zelf indien nodig een ISO image op DVD branden.

Delphi Prism 2009 Enterprise

New User	€ 700,=
Subscription (verplicht 1e jaar)	€ 699,= per jaar

Delphi Prism 2009 Professional

New User	€ 250,=
Subscription (verplicht 1e jaar)	€ 249,= per jaar

Delphi 2009 Architect (Win32)

New User	€ 3.110,=
Upgrade	€ 2.011,=
Subscription Delphi for Win32	€ 933,= per jaar

Delphi 2009 Enterprise (Win32)

New User	€ 1.799,=
Upgrade	€ 1.199,=
Subscription Delphi for Win32	€ 540,= per jaar

Delphi 2009 Professional (Win32)

New User	€ 799,=
Upgrade	€ 349,=
Subscription Delphi for Win32	€ 240,= per jaar

CodeGear RAD Studio 2009 Architect (ESD)

New User	€ 4.299,=
Upgrade	€ 2.799,=
Subscription CodeGear RAD Studio	€ 1.290,= per jaar

CodeGear RAD Studio 2009 Enterprise (ESD)

New User	€ 2.799,=
Upgrade	€ 1.799,=
Subscription CodeGear RAD Studio	€ 840,= per jaar

CodeGear RAD Studio 2009 Professional (ESD)

New User	€ 1.399,=
Upgrade	€ 649,=
Subscription CodeGear RAD Studio	€ 420,= per jaar

Special JUNI 2009 Aanbieding: bij de aanschaf van een Delphi 2009 Upgrade krijgt u een RAD Studio 2009 geleverd (dus inclusief C++Builder 2009 en Delphi Prism 2009).

Zie http://www.eBob42.com/CodeGear of stuur e-mail naar Bob@eBob42.com voor meer informatie of om een bestelling te plaatsen