Delphi OplossingsCourant

Vol. 7. No. 2, Een gratis electronische publicatie van **Bob Swart Training & Consultancy** (**eBob42**) - http://www.eBob42.com

Helmond, 3 september 2005,

Welkom bij het nieuwe nummer van de Delphi OplossingsCourant; dat ook deze keer weer in het teken zal staan van nieuwe ontwikkelingen.
Voot het eerst in de geschiedenis heeft Borland informatie over niet één maar twee toekomstige versies van Delphi bekend gemaakt, inclusief de codenamen DeXter en Highlander.
DeXter is de codenaam voor Delphi 2006, die nog voor eind dit jaar (2005 dus) zal verschijnen. De X in DeXter staat overigens voor "10". DeXter zal als nieuwe persoonlijkheid nu ook C++(Builder) voor Win32 gaan bevatten. Op .NET gebied helaas nog steeds alleen .NET 1.1. ondersteuning. Voor .NET 2.0 support moeten we wachten op Delphi **Highlander**, die pas medio volgend jaar (2006) zal verschijnen.
Voor geregistreerde Delphi 2005 gebruikers is sinds kort ook de zgn. *.NET Compact Framework command-line compiler technology preview* te downloaden, maar die ondersteuning zit er bij DeXter meteen al in.

In dit nummer van de Delphi OplossingsCourant besteed ik aandacht aan een ander nieuwtje: Skype! Ik laat zien hoe je met Delphi (Win32) gebruik kunt maken van de Skype Client en de Skype API.
Het tweede artikel gaat over het automatisch verzenden van e-mail vanuit Delphi for .NET.
Het laatste artikel gaat over ASP.NET en DB Web toepassingen in Delphi 2005. Wie overigens enkele praktijkvoorbeelden van Delphi 2005 en ASP.NET wil zien kan bijvoorbeeld mijn cursusboek verkoop toepassing te http://www.eBob42.com/courseware of weblog te http://www.eBob42.com/blog bekijken.

Veel leesplezier, en schroom niet om op- of aanmerkingen te mailen of Skype-en!

Groetjes,

Nieuwe Delphi 2005 Clinics

Ook na de zomervakantie heb ik enkele Delphi 2005 Clinics gepland, waar iedereen zich voor kan inschrijven.

8 september
ADO.NET Database Development Essentials
http://www.eBob42.com/training/2005ado.htm

22 september
ASP.NET Web Development Essentials
http://www.eBob42.com/training/2005asp.htm

6 oktober
Advanced ASP.NET Web Development
http://www.eBob42.com/training/2005asp2.htm

27 oktober
XML, SOAP and Web Services Essentials
http://www.eBob42.com/training/2005xml.htm

Daarnaast is het altijd mogelijk om een Workshop bij mij in Helmond, of een Custom Clinic bij u op lokatie te laten organiseren. De onderwerpen die daarbij aan de orde komen bepalen we in overleg. Zie http://www.eBob42.com/workshop voor details. Voor wie geen trainingen kan bijwonen, is er ook de mogelijkheid om de individuele cursusboeken te bestellen. Zie http://www.eBob42.com/courseware voor informatie en on line bestelmogelijkheden.

Samenwerking met Brian Long

Alhoewel ik als eenmanszaak vrijwel alles alleen doe, ben ik recent een samenwerking aangegaan met Brian Long (http://www.blong.com) op het gebied van training en consultancy, waardoor we samen een nog breder aandachtsgebied kunnen bestrijken. Brian is nu ook de technisch reviewer van mijn Delphi 2005 cursusboeken, en gebruikt deze ook bij zijn eigen Delphi 2005 trainingen (net als enkele andere trainingsbedrijven zoals DTS).

Samenwerking met de SDN

Ook met de Software Development Network (SDN) ben ik een samenwerking aangegaan op het gebied van trainingen en cursusboeken voor SDN-leden. Zie http://www.sdn.nl/Default.aspx?tabid=134

Conferenties en Events

Behalve de Delphi Clinics ben ik in de tweede helft van 2005 ook als spreker aanwezig bij de SDE van de SDN op 16 september, bij de EKon 8 conferentie in Frankfurt Duitsland op 26-30 september, en de Borland Developers Conferentie op 6-10 november in San Francisco, USA (Marco Cantù en ik zijn daar de enige Europese sprekers) alsmede de Europese Borland Conferentie op 28-30 november 2005 in ons eigen Amsterdam. Voor deze laatste conferentie zit ik wederom in de Advisory Board, en zal mijn best doen om weer een goed en evenwicht programma samen te stellen.

Kort samengevat zal ik de volgende Delphi sessies verzorgen:

SDE, 16 september
- Using the Skype API with Delphi (for Win32)
- SharePoint 2003 Web Parts with Delphi 2005

EKon 8, 26 september
- ADO.NET DB Development Power Workshop

Ekon 8, 27-28 september
- ASP.NET Web Forms and Databases
- SharePoint 2003 Web Parts with Delphi 2005
- XML Document Programming in Win32

Borland DevCon 2005, 6-10 november
- ASP.NET Development with Delphi for .NET
- ASP.NET Web Forms and Databases
- Multi-Platform Web Services with Delphi 2005

Zie http://www.drbob42.com/events voor de laatste updates en de sessies voor de Europese BorCon die op dit moment nog niet bekend zijn..

De 24 uur van Delphi

Op woensdag 13 juli 2005 nam ik deel aan de internationale 24-uurs Delphi marathon op BDN Radio. De hele dag zijn daarbij verschillende Delphi sprekers aan het woord gekomen.
Deelnemers waren o.a. Danny Thorpe, Allen Bauer, Steve Trefethen, Ramesh Theivendran, John Keegan, Jim Tierney, het Borland ECO team, Robert Love, Marco Cantù, Michael Swindell, Malcolm Groves, Jason Vokes, Daniel Magin, Bernd Ua, Ray Konopka, Nick Hodges, en ook ikzelf was op 13 juli van 10:45 tot 11:30 te horen in een interview over training, customer stories, en migraties, maar ook mijn gebruik van de Skype API met Delphi en m'n Skype antwoordapparaat.
Je kan de interviews nu downloaden als mp3 van http://bdn.borland.com/article/0,1410,33109,00.html

Delphi ECO II Weblog

Ik heb nu eindelijk ook een (engelstalige) weblog, geschreven in Delphi 2005 met behulp van ASP.NET en ECO II. Sinds begin augustus is de weblog te lezen op http://www.eBob42.com/blog, waar ook verwijzingen staan naar de artikelen die ik schreef voor Bitwise Magazine, en beschrijven hoe de ASP.NET weblog toepassing na te bouwen en te deployen is.

Bitwise Magazine

Over Bitwise Magazine gesproken, dat is een gratis on-line magazine met artikelen over Delphi, C#, VB.NET, Java, etc. Te vinden op de voor de hand liggende url http://www.bitwisemag.com.
Met mijn Delphi artikelen heb ik een kleine bijdrage kunnen leveren aan de lancering, en het is dan ook met groot genoegen dat ik een speciale wedstrijd kan aankondigen waarbij drie personen een doos met Delphi 2005 Architect kunnen winnen. Je hoeft alleen maar een simpele vraag over Delphi te beantwoorden (voor 16 september).
Zie http://www.bitwisemag.com en doe mee!

Delphi 2005 Compact Framework

De Delphi *Compact Framework Technology Preview* bestaat uit een command-line compiler voor het .NET CF. Hiermee kun je een bestaande WinForms toepassing met enkele aanpassingen (properties die niet door CF ondersteund worden in commentaar zetten) opnieuw compileren tot een toepassing die op een PocketPC kan werken. De volgende keer zal ik hier met een voorbeeld dieper op ingaan!

Delphi Skype Antwoordaparaat

Ik ben de afgelopen tijd bezig geweest met het gebruiken van de Skype API in Delphi. Een van de resultaten is een antwoordapparaat verbonden aan Skype handle "drbob42". Wie via Skype met drbob42 belt krijgt een welkomstbericht en kan vrijblijvend een boodschap achterlaten.
Ik gebruik overigens de message-based Skype API, en niet de COM of ActiveX laag. En alhoewel Skype zelf ook VoiceMail aanbiedt, is deze niet gratis. Mijn "antwoordapparaat" kost alleen een "male-male" audio kabeltje om je speaker output weer naar je microfoon input door te lussen als het antwoord-apparaat aan staat.
Zie ook http://www.drbob42.com/Skype of lees het artikel elders in dit nummer over details om met de Skype Client en Skype API te werken vanuit een Delphi toepassing.
Overigens is Microsoft al bezig met een Skype-killer, die in de nieuwe MSN zal komen. Daarover een volgende keer meer.

Bellen met Delphi en de Skype API

In dit artikel beschrijf ik de eerste stappen die je moet zetten om een Delphi toepassing te laten integreren met Skype. Een eerdere versie van dit artikel verscheen ook in SDN Magazine, en mijn Skype antwoordapparaat is in meer detail beschreven in The Delphi Magazine.

Skype

Skype is een dienst die telefonie over een breedband internet verbinding mogelijk maakt. Je moet hiervoor de Skype software downloaden (van http://www.skype.com) en op je machine installeren.
Gratis telefonie alleen mogelijk met andere Skype gebruikers. Daarnaast is het mogelijk om een SkypeOut beltegoed te kopen, om daarmee naar externe telefoonnummers te bellen. Tot slot is er ook een SkypeIn (momenteel in beta), waarmee je een "vast" nummer kunt aanvragen om daarop gebeld te worden – ook hiervoor moet je betalen echter.
Behalve de Skype software, is het voor ontwikkelaars ook mogelijk om de Skype API te gebruiken, en die bijvoorbeeld te integreren in eigen software (een snelle en goedkope manier om vanuit je Delphi toepassing de helpdesk te laten bellen bijvoorbeeld). Daarvoor moet de gebruiker uiteraard wel Skype zelf op zijn machine hebben staan.

Skype API

In dit artikel zal ik laten zien hoe je een begin kan maken met het gebruik van de Skype API. Deze API is geen laag om de Skype telefonie/communicatie library, maar een laag om de Skype client zelf. Wie de Skype client op zijn machine heeft, die heeft dus automatisch ook alle benodigde APIs beschikbaar.
Behalve de platte Skype API, is het ook mogelijk om via een COM interface met Skypy te praten (voor oude VB6 ontwikkelaars bijvoorbeeld). De Skype COM server is te vinden in de system32 directory als skypeapi.dll. Het gebruik van de COM server is in principe eenvoudiger (vanuit de ontwikkelaar gezien), maar betekent wel dat je dan een extra installatie op de client moet doen – of moet afdwingen dat de eindgebruiker zelf de COM server geïnstalleerd heeft. Ik zal me in dit artikel beperken tot de niet-COM manier, en daartoe moeten we met de Skype Client communiceren via Windows messages.

Skype Client Detectie

De Skype API is met name bedoeld om binnenkomende telefoontjes buiten de Skype client zelf af te handelen (je kunt hierbij denken aan een antwoordapparaat) - maar de Skype client zelf moet dus aanwezig zijn (en zal indien nodig gestart moeten worden).
Het detecteren van de aanwezigheid van Skype kan in de registry, door te zoeken naar de key HKEY_LOCAL_MACHINE\SOFTWARE\Skype\Phone die een String value SkypePath heeft met de waarde het pad naar de Skype.exe (bij mij is dat C:\Program Files\Skype\Phone\Skype.exe).

```
function SkypeClient: String;
var
  Reg: TRegistry;
begin
  Result := '';
  Reg := TRegistry.Create(KEY_READ);
  try
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    if Reg.OpenKey('SOFTWARE\Skype\Phone', False) then
      Result := Reg.ReadString('SkypePath')
  finally
    Reg.Free
  end
end;
```

Als het resultaat leeg is, dan is de Skype Client niet geïnstalleerd, en de Skype API niet beschikbaar.

Protocol versies

Als de Skype Client gevonden is, dan is de volgende stap het bepalen van de compatibiliteit tussen wat ons programma verwacht van de Skype API, en wat de Skype Client op de huidige machine daadwerkelijk ondersteunt.
Er zijn op dit moment meerdere verschillende versies van de Skype API beschikbaar, onder de naam PROTOCOL 1, 2, etc.. Op het moment van schrijven is PROTOCOL 4 geïnstalleerd op mijn ontwikkelmachine, maar is PROTOCOL 5 en hoger al niet ver meer.
Als ontwikkelaar kun je gebruik maken van het nieuwste protocol, maar dan zul je wel moeten vragen of de Skype Client op de uiteindelijke deployment machine ook dit protocol ondersteunt. We zullen aan de Skype Client moeten vertellen welk protocol versie we ondersteunen (bijvoorbeeld "PROTOCOL 4"), en de Skype Client zal daarop antwoorden door zijn versie terug te geven. Nieuwere Protocol versies bouwen voort op eerdere versies, dus wie protocol 4 ondersteunt, ondersteunt ook protocol 1, 2 en 3. Op onze melding "PROTOCOL 4" zal de Skype Client kunnen antwoorden met "PROTOCOL 1", 2, 3 of 4. Het is uiteraard aan te raden om vervolgens het Protocol versie te gebruiken waar de Skype Client mee terugkwam (in ieder geval niet hoger).
De Skype Client zal nooit met een hoger PROTOCOL nummer komen dan onze aanroep (omdat onze software die niet ondersteunt). Een leuke truc om het daadwerkelijke PROTOCOL nummer van de geïnstalleerde Skype Client te vinden is dus om PROTOCOL met een hogo nummer aan te roepen, zoals PROTOCOL 9999, waarop de Skype Client dan zal antwoorden met het hoogst mogelijke PROTOCOL nummer beschikbaar op de betreffende machine. Veel heb je er overigens niet aan als je te horen krijgt dat die machine PROTOCOL 42 ondersteunt, als je zelf niet meer weet (en dus kan) dan PROTOCOL 4...

Windows Message Communicatie

De communicatie tussen onze Delphi toepassing en de Skype Client gaat via speciale Windows Messages. We moeten dus eerst zorgen dat onze zelfgeschreven Delphi toepassing deze messages kent, zodat we ze kunnen versturen en ontvangen. Daarvoor gebruiken we de RegisterWindowMessage Win32 API, die als argument de string van de Windows Message meekrijgt. De twee Windows Messages die we moeten registreren zijn SkypeControlAPIAttach en SkypeControlAPIDiscover.
In de FormCreate kun je deze messages registreren, waarbij ik het resultaat in twee DWord velden van het Form zelf opsla. Dus het TfrmSkype form krijgt het volgende erbij:

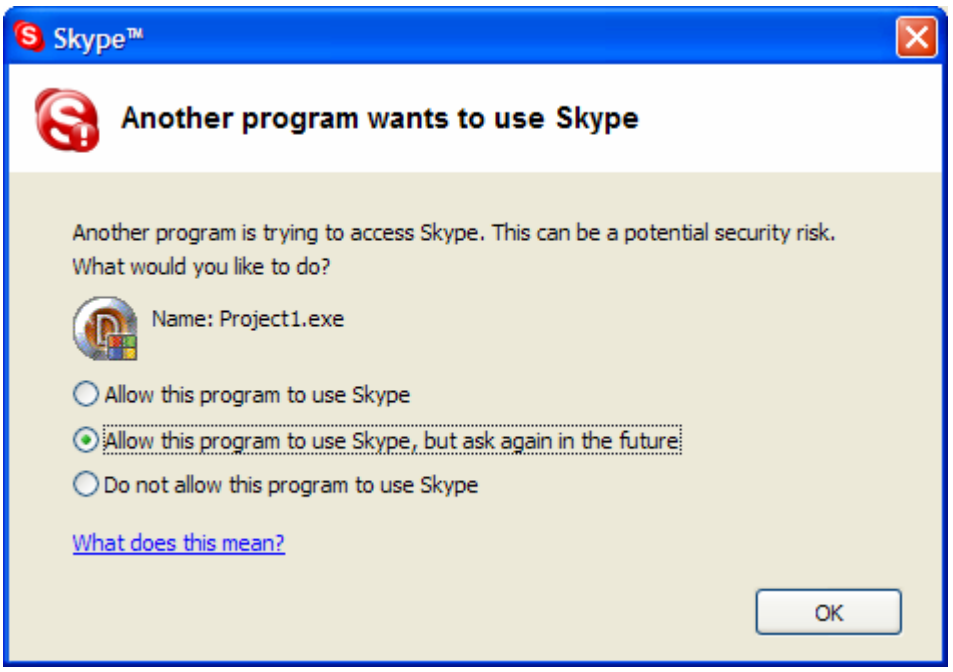
```
  private
    WM_SkypeControlAPIDiscover: DWord;
    WM_SkypeControlAPIAttach: DWord;
```

En in de FormCreate kunnen we deze twee als volgt een waarde geven:

```
procedure TfrmSkype.FormCreate(Sender: TObject);
begin
  if SkypeClient <> '' then // zie vorige pagina
  begin
    WM_SkypeControlAPIDiscover := RegisterWindowMessage('SkypeControlAPIDiscover');
    WM_SkypeControlAPIAttach := RegisterWindowMessage('SkypeControlAPIAttach');
    SendMessage(HWND_BROADCAST, WM_SkypeControlAPIDiscover, Handle, 0)
  end
  else
  begin
    ShowMessage('Error: De Skype client is niet gevonden op deze machine.'#13#10#13#10 +
       'Indien nodig kan Skype gedownload worden van http://www.skype.com');
    Application.Terminate
  end
end;
```

Let op dat we na het registreren van de messages meteen een Broadcast message versturen van de WM_SkypeControlAPIDiscover, om te zien of de Skype Client daarop reageert.

Als de Skype Client draait, dan zal de WM_SkypeControlAPIDiscover Broadcast message ook bij hem aankomen, en als resultaat zal de Skype Client een WM_SkypeControlAPIAttach bericht terugsturen naar ons, om te vertellen of de poging om een verbinding te maken (al) is gelukt. Soms is dat niet meteen het geval, als het namelijk om een nieuwe toepassing gaat, of een toepassing die nog niet bekend is bij de Skype Client. In dat geval krijgt de gebruiker op de betreffende machine eerst een dialoog te zien waarin toestemming gegeven moet worden alvorens de verbinding tussen onze toepassing en de Skype Client wordt toegestaan.



De default keuze bestaat uit het toestaan van het gebruik van de Skype Client, maar dit opnieuw vragen en laten bevestigen door de eindgebruiker in de toekomst. Alternatieven zijn het onconditioneel toestaan of niet toestaan. De Skype Client onthoudt dit, en koppelt dit aan de informatie over "Project1.exe". Als dezelfde Project1.exe in de toekomst nog een keer langskomt, dan is hij in ieder geval bekend. Als Project1.exe in gewijzigde vorm langskomt, dan zal de Skype Client dit aangeven. Zo van "let op, je gaf in het verleden eens toestemming, maar inmiddels is Project1.exe gewijzigd.". Prettig om te weten voor het geval er iets of iemand met Project1.exe gaat knoeien.
Terwijl de Skype Client wacht op input van de gebruiker, komt er bij onze Delphi toepassnig een message van type SkypeControlAPIAttach binnen, waarbij de status is weergegeven in het Msg.LParam deel. Als dat 0 is, dan is de permissie gegeven. Is het 1, dan moeten we nog even wachten (dan zal de dialoog zichtbaar zijn bijvoorbeeld). Als Msg.LParam 2 is, dan is de toegang geweigerd.
Als we uiteindelijk een SkypeControlAPIAttach message met Msg.LParam gelijk aan 0 binnenkrijgen, dan zit in de Msg.WParam de Windows Handle van de Skype Client waar we dan vanaf nu direct Windows Messages naartoe mogen sturen, en terug van kunnen verwachten. De communicatie is dan gereed. Al deze berichten komen binnen bij de WndProc van onze main form, en kunnen we als volgt afhandelen:

```
procedure TfrmSkype.WndProc(var Message: TMessage);
begin
  if Message.Msg = WM_SkypeControlAPIAttach then
  begin
    if Message.LParam = 0 then
    begin
      HWND_SkypeAPIWindowHandle := Message.WParam;
      StatusBar1.SimpleText := 'Attached';
    end
```

```
    else
      case Message.LParam of
        1: StatusBar1.SimpleText :=
          'SkypeControlAPIAttach: Waiting for confirmation from the user.';
        2: StatusBar1.SimpleText :=
          'SkypeControlAPIAttach: User has denied access to Skype client!';
        3: StatusBar1.SimpleText :=
          'SkypeControlAPIAttach: Skype API is not available at this time.'
      else
        StatusBar1.SimpleText := 'Not Attached';
      end;
    Message.Result := 1
  end
  else
    inherited
end;
```

Let op dat we de Msg.Result altijd op 1 (of in ieder geval niet 0) moeten zetten als we een bericht voor ons ontvangen hebben, anders denkt de Skype Client dat de verbinding intussen al verbroken is.

SendMessageToSkype

Zodra we attached zijn kunnen we vragen wel prorocol er ondersteund wordt. Daartoe moeten we een SendMessage doen naar de SkypeAPIWindowHandle. Hiervoor heb ik een speciale SendMessageToSkype method geschreven, met de volgende code:

```
procedure TSkypeForm.SendMessageToSkype(Str: String);
var
  CopyData: CopyDataStruct;
begin
  if Str <> '' then
  begin
    Log.Lines.Add('SEND: ' + Str);
    CopyData.dwData := 0;
    CopyData.lpData := PChar(Str);
    CopyData.cbData := Length(Str)+1;
    SendMessage(SkypeAPIWindowHandle, WM_COPYDATA, Application.Handle,
      LPARAM(@CopyData))
  end
end;
```

En deze kunnen we als volgt aanroepen vanuit de WindowHook method (vlak na de assignment naar de SkypeAPIWindowHandle):

```
      SendMessageToSkype('PROTOCOL 4')
```

WM_COPYDATA

Vanaf nu zijn alle berichten die van de de Skype Client ontvangen van het WM_COPYDATA type. Hiervoor hoeven we niet langer de WndProc te gebruiken, maar kunnen we gewoon de WM_COPYDATA message handler gebruiken. Ga met de cursor in de class definitie van je form staan, druk op Control+Spatie, en een window met de lijst met alle methodes zal verschijnen. Zoek hierin naar de WM_COPYDATA en druk op Enter om de declaratie aan je class toe te voegen.

```
    procedure WMCopyData(var Message: TWMCopyData); message WM_COPYDATA;
```

Control+Shift+C zorgt er vervolgens voor dat de implementatie ook beschikbaar komt. Het voordeel van de WMCopyData methode is dat we nu een argument van type TWMCopyData hebben in plaats van een generieke TMessage, zodat we gericht in het ontvangen bericht kunnen kijken.

Wel moeten we eigenlijk nog even controleren of het WM_COPYDATA bericht wel afkomstig is van de HWND_SkypeAPIWindowHandle (die we een waarde gaven in de WndProc.

```
procedure TfrmSkype.WMCopyData(var Message: TWMCopyData);
begin
  if (Message.From = HWND_SkypeAPIWindowHandle) and
     (HWND_SkypeAPIWindowHandle > 0) then
  begin
    // doe iets...
    Message.Result := 1
  end
end;
```

Een van de dingen die we nu kunnen maken is een Skype Trace toepassing, waarbij we alle uitgaande berichten (in de SendMessageToSkype) en binnenkomende berichten (in de WMCopyData) laten zien in een TMemo component.
Als dit allemaal achter de rug is, dan kunnen we een SendMessage met een van de Skype API commandos sturen,. Zoals het CALL commando, waarbij we als argument de Skype ID van de ontvanger moeten opgeven. Ik heb zelf een aantal Skype ID's, waaronder een Delphi antwoordapparaat onder "drbob42" dat je kunt testen met "CALL drbob42".

Wie met de Skype API aan de slag wil en er echte integraties mee wil bouwen, doet er goed aan de Skype API documentatie op de Skype website door te nemen, en de verschillende voorwaarden te lezen. Mijn eigen experimenten hebben nog niet tot commerciële toepassingen geleid, maar dienen voornamelijk om anderen te laten zien hoe de integratie met Skype mogelijk is. Wie daarbij een helpende hand zou willen gebruiken kan bij mij altijd terecht. Stuur een e-mailtje naar bob@eBob42.com, of Skype even naar "eBob42" (het Skype account met een echt persoon aan de andere kant) voor een vrijblijvend advies.

Automatisch E-mail versturen vanuit Delphi for .NET

Nee, die artikel gaat niet over het versturen van spam, alhoewel dit wel ter sprake zal komen (hoe je je juist als niet-spammer moet gedragen). Het gaat over de nieuwsbrief die je nu aan het lezen bent: de "Delphi OplossingsCourant", waar Delphi ontwikkelaars zich op kunnen abonneren door een mailtje te sturen naar doc@eBob42.com. Bij het verschijnen van een nieuw nummer krijgen deze abonnees automatisch een e-mailtje met de inhoudsopgave, een korte beschrijving en meestal als reclame ook het nieuwe overzicht van mijn trainingsaanbod.
Toen ik in 2000 begon met het publiekelijk verspreiden van de Delphi OplossingCourant waren er een paar dozijn abonnees, en kon ik iedereen nog persoonelijk een mailtje sturen. Inmiddels staan er een paar honderd e-mail adressen op de lijst (ik sla overigens alleen het e-mail adres op, en weet dus niet welke naam of andere persoonlijke informatie erbij hoort). Een paar honderd mailtjes versturen is niet meer handmatig te doen. Maar één mailtje naar een paar honderd email adressen proberen te versturen werkt ook niet, want dan wordt ik door mijn ISP als spammer gezien, en loop ik bovendien het risico afgesloten te worden van internet.
Het probleem is geboren, en de oplossing ligt voor de hand. Waarom moeilijk doen als het makkelijk kan? In de rest van dit artikel beschrijf ik een kleine Delphi 2005 toepassing die ik schreef om een e-mail bericht (in het ene bestand) te sturen naar een lijst met e-mail adressen (in een andere bestand). Volledig automatisch – eindelijk weer eens een plek waar de automatisering echt van nut kan zijn.

Delphi 2005

Ik besloot de nieuwste versie van Delphi te gebruiken, Delphi 2005. En dan komt meteen het tweede keuzemoment: wordt het een Windows 32-bit of een .NET toepassing? In beide gevallen was het grootste probleem om e-mailtjes te versturen, de rest van de functionaliteit was niet zo complex. In Win32 kun je de Indy componenten te gebruiken, en die zijn ook beschikbaar in VCL for .NET. In plaats van de Indy componenten te gebruiken – die prima geschikt zijn – besloot ik echter om te kijken of er in het .NET Framework zelf al componenten zijn opgenomen met de functionaliteit om e-mailtjes te versturen.

System.Web.Mail

E-mailtjes versturen kun je doen vanuit zowel een Windows GUI als een Web toepassing, en zelfs een console toepasing als je zou willen. Dat klinkt mij tenminste logisch in de oren. Toch was ik een beetje verbaasd dat de enige plek waar ik een MailMessage class kon vinden, de System.Web.Mail namespace was. Toegegeven, het versturen van e-mail heeft met het internet te maken, maar niet zozeer met het world-wide-web (ik verstuurde mijn eerste e-mail al via Multix eind jaren 70, toen je nog ! in plaats van @ moest gebruiken, en de eerste websites kwamen toch echt pas eind jaren 80 van de grond).
Het .NET Framework biedt visuele componenten aan voor Windows GUI Forms (ook wel WinForms genoemd) in de System.Forms namespace hierarchy. Het alternatief zijn ASP.NET Web Forms componenten, die in de System.Web namespace hierarchy te vinden zijn. Je zou dan ook op het eerste gezicht geneigd zijn te denken dat de MailMessage class uit de System.Web namespace alleen voor ASP.NET toepassingen geschikt is. Gelukkig is dat niet het geval: je kunt deze class ook in WinForms, console en zelfs VCL for .NET toepassingen gebruiken! De MailMessage is een niet-visuele class, ook niet terug te vinden in deTool Palette (net zo min als je ooit een StringList in de Tool Palette zal vinden).

MailMessage

De MailMessage class uit System.Web.Mail werkt samen met het globale SmtpMail object uit dezelfde namespace. Via SmtpMail stel je de SmtpServer in, en roep je de Send methode aan. De Send methode krijgt als argument een instantie van de MailMessage class aan (die instantie moet je dus voor ieder mailtje zelf maken, invullen en dan meegeven). Het SmtpMail object bestaat dus altijd al binnen je toepassing.

De MailMessage class heeft een aantal properties die je een waarde moet geven. Zo is er de From property, die net als de Subject property van type String is. Dat ligt vrij voor de hand. Iets lastiger is de To property. Deze is ook van type String, maar de naam van de property is ook een keyword in Delphi. Daar hebben ze in C# en VB.NET natuurlijk geen last van, maar een Delphi ontwikkelaar krijgt een foutmelding bij de volgende regel code (waarbij Msg van het type System.Web.Mail.MailMessage is):

```
Msg.To := 'bob@eBob42.com';
```

De foutmelding is "Identifier expected but TO found". Om toch aan te geven dat we de property To willen gebruiken zonder het als keyword te zien, moeten we er een & voor zetten. Dus dat wordt dan:

```
Msg.&To := 'bob@eBob42.com';
```

Dit geldt ook voor classes die eventueel methodes hebben met de naam Begin of End. Die kun je ook niet aanroepen zonder er een & voor te zetten.
Overigens is het aardig te zien dat de & er automatisch voor komt als je de property To via Code Insight kiest (dus Msg. intikken, wachten op de lijst met properties en events, en dan To kiezen om vervolgens Msg.&To als resultaat te krijgen).

Aan de slag!

Genoeg theorie. Tijd om de toepassing te bouwen – hands-on, dus wie mee wil doen kan zich nu naar de computer begeven. Zoals ik al zei is de MailMessage class uit de System.Web.Mail namespace te gebruiken WinForms, WebForms, VCL for .NET en zelfs Console toepassingen. Omdat het eindresultaat toch een toepassing is die ik aanzet om een paar honderd e-mailtjes te versturen zonder dat ik daar verder bij nodig ben, lijkt een console toepassing mij voldoende. Kan er altijd nog een foutboodschap op de console geschreven worden mocht er iets mis gaan.
Start Delphi 2005 (of Delphi 8 for .NET als je die nog hebt), en do File | New – Other, ga naar de Delphi for .NET Projects categorie in de Object Repository, en kies voor de Console Application. Dit levert een nieuwe console toepassing op voor .NET (ook te zien aan het grijze helmpje dat je nu rechts van de menubalk zult ziet). Ik bewaar het project als MailDoc.dpr.
Om de MailMessage class te kunnen gebruiken, moeten we nu eerst de System.Web.Mail namespace aan ons project toevoegen. Dat is niet hetzelfde als de System.Web.Mail unit in de uses clause zetten, want dan weet Delphi nog niet waar de System.Web.Mail unit informatie vandaan gehaald moet worden. We moeten de bijbehorende .NET assembly toevoegen aan de lijst met referenties van ons project. In de project manager moet je met de rechtermuisknop op de MailDoc.exe klikken, en dan in het pop-up menu voor "Add Reference" kiezen. Je krijgt nu een dialoog te zien met daarin drie tabs (voor .NET Assemblies, COM Imports en Project References). In de eerste tab staan alle .NET Assemblies waar we uit kunnen kiezen,en hier zit ook de System.Web assembly tussen. Merk op dat de .NET Assembly System.Web.Mail er niet is. Maar omdat System.Web er wel is, zal de namespace System.Web.Mail zich in de System.Web assembly bevinden. Het is belangrijk om te onthouden dat er in het .NET Framework geen één-op-één koppeling tussen een assembly en namespace hoeft te zijn. Een .NET Assembly kan meerdere namespaces in zich bevatten (en in theorie kan een namespace verspreid zijn over meerdere .NET Assemblies, ahoewel we dat minder vaak zien).
In de Add Reference dialoog moet je dus op zoek naar de System.Web assembly, en daarop dubbelklikken zodat hij in het lijstje New References onderaan de dialoog komt te staan. Als je daarna op OK klikt zal de System.Web namespace worden toegevoegd aan je console toepassing. Dit kun je behalve in de project manager, ook terugzien in je source file zelf, die de volgende regel erbij krijgt:

```
{%DelphiDotNetAssemblyCompiler
'$(SystemRoot)\microsoft.net\framework\v1.1.4322\System.Web.dll'}
```

En nu kun je veilig de System.Web.Mail namespace aan je uses clause toevoegen (en de SysUtils unit verwijderen, die hebben we toch niet nodig).

Voor het versturen van een mailtje moeten we een nieuwe instantie van de MailMessage class maken, de benodigde properties invullen, inclusief de Body (daar kom ik zo nog op terug), en dan via het SmtpMail object de SmtpServer instellen en Send aanroepen. In code is dat als volgt:

```
var
  Msg: System.Web.Mail.MailMessage;
begin
  Msg := MailMessage.Create;
  Msg.&To := 'drbob@chello.nl';
  Msg.From := 'b.swart@chello.nl';
  Msg.Subject := 'Delphi OplossingsCourant vol.7 no.2 - Delphi 2005';
  Msg.Body := 'Dit is een test';
  SmtpMail.SmtpServer := 'mail.chello.nl';
  try
    SmtpMail.Send(Msg);
    writeln('Bericht verzonden');
  except
    on E: Exception do
      writeln('Error: ',E.Message)
  end;
  readln
end.
```

De readln op het eind is om te zorgen dat ik het programma vanuit de IDE kan runnen, en dan nog de kans krijg om het bericht in het console window te lezen voor ik op <enter> kan drukken om de toepassing weer af te sluiten.
Voor je dit programma draait hoop ik dat je het e-mail adres aanpast – zowel van de afzender als de ontvanger – zodat ik niet nog meer (ongewenste) mail krijg. Bij voorbaat dank.
Vul je eigen e-mail adres en smtp server van je internet provider in, en probeer dan uit of het werkt. Bij sommige internet providers moet je een username en password invullen voor je mag mailen, maar bij de meeste providers is het voldoende om op hun netwerk ingelogd te zijn (via de verbinding door je modem – of dat nou een kabel, ADSL, ISDN of ouderwets 14K4 modem is).

Gebruikerslijst

Als het versturen van één mailtje lukt, dan lukt het ook wel om honderden mailtjes te versturen. De vraag is even hoe we de lijst met e-mail adressen zullen opslaan. Er zijn genoeg programmeurs die meteen naar een database grijpen daarvoor, maar dat lijkt me persoonlijk een beetje overkill. Ik hoef alleen maar het e-mail adres te bewaren, en niet de naam of overige gegevens van de betreffende persoon (het gratis abonnement is wat dat betreft volledig anoniem). Voor mijn cursistenbestand gebruik ik wel een database, maar voor mijn wekelijkse boodschappenlijstje pak ik gewoon een potlood en papiertje.
In het februari 2005 nummer van The Delphi Magazine heb ik in een artikel beschreven hoe .ini files als alternatief voor databases gebruikt kunnen worden (met name handig als je een soort database tabellen wilt gebruiken op een web server, maar geen echte database mag of kan installeren). Echter, omdat we hier maar één veld hoeven op te slaan is zelfs dat overkill in mijn ogen. De lijst met e-mail adressen heb ik uiteindelijk opgeslagen in een gewoon tekstbestand, genaamd *email.txt*, met op iedere regel één e-mail adres. Die kan in inlezen in een TStringList uit de Borland.Vcl.Classes namespace, en vervolgens door de lijst met strings lopen in de TStringList. In versies van Delphi voor Delphi 2005, moet dat nog als volgt, met een index:

```
var
  Abonnees: TStringList;
  Adres: String;
  i: Integer;
begin
```

```
  Abonnees := TStringList.Create;
  Abonnees.LoadFromFile('email.txt');
  for i:=1 to Abonnees.Count do
  begin
    Adres := Abonnees[i];
  end;
```

Maar in Delphi 2005 kan dat ook als volgt, met de nieuwe for-in-do loop:

```
var
  Msg: System.Web.Mail.MailMessage;
  Abonnees: TStringList;
  Adres: String;
begin
  Abonnees := TStringList.Create;
  Abonnees.LoadFromFile('email.txt');
  for Adres in Abonnees do
  begin
    // doe iets…
  end;
```

Geen index meer nodig, en je hoeft ook niet meer te kijken hoeveel elementen er in de StringList zitten. Op eenzelfde manier kan het hele e-mail bericht ook in een textbestand staan, bijvoorbeeld in *message.txt*, dat we ook via een TStringList in één keer kunnen inlezen, en dan in één keer aan de Body van de Msg toekennen via de Text property. Ons programma ziet er nu al als volgt uit, en is bijna klaar om ingezet te worden:

```
var
  Msg: System.Web.Mail.MailMessage;
  Abonnees,Inhoud: TStringList;
  Adres: String;
  i: Integer;
begin
  Inhoud := TStringList.Create;
  Inhoud.LoadFromFile('message.txt');
  Abonnees := TStringList.Create;
  Abonnees.LoadFromFile('email.txt');
  SmtpMail.SmtpServer := 'mail.chello.nl';
  Msg := MailMessage.Create;
  Msg.From := 'b.swart@chello.nl';
  Msg.Subject := 'Delphi OplossingsCourant vol.7 no.1 - Delphi 2005';
  Msg.Body := Inhoud.Text;
  for Adres in Abonnees do
  begin
    Msg.&To := Adres;
    try
      SmtpMail.Send(Msg);
      writeln('Bericht verzonden naar ',Adres);
    except
      on E: Exception do
        writeln('Error: ',E.Message,' bij ',Adres)
    end
  end;
  readln
end.
```

Wat ontbreekt er nog aan? De anti-spam voorzieningen!

Spik en Spam

Als ik het programma zoals het nu is inzet om een paar honderd mailtjes te versturen, dan is de kans groot dat mijn internet provider dat detecteert en voortijdig ingrijpt om de mailtjes te onderscheppen. Zeker als het gaat om een paar honderd mailtjes in enkele minuten met allemaal hetzelfde onderwerp – dat ruikt naar spam, ziet eruit als spam en smaakt naar spam (alleen weet mijn provider niet dat het geen spam is). Om te zorgen dat mijn provider geen onraad ruikt kan ik twee voorzorgsmaatregelen treffen: allereerst kun je het onderwerp van het mailtje een klein beetje aanpassen, bijvoorbeeld door de naam van de ontvanger er in te zetten (da's jammer, die heb ik niet – alleen het e-mail adres), of door er iets anders achter te plakken. Ik krijg zelf regelmatig spam waarbij het me opvalt dat aan het eind van de subject regel er een aantal spaties staan gevolgd door enkele willekeurige tekens. Op die manier houden de echte spammers ook hun internet provider voor het lapje, denk ik.
Een andere – en veiliger – manier is om te voorkomen dat er tientallen mailtjes tegelijk worden verstuurd, door na ieder mailtje even een pauze in te lassen. Als je bij 600 mailtjes na ieder mailtje 10 seconden wacht dan duur het ten minste 6.000 seconden oftewel 100 minuten voor alle mailtjes verzonden zijn (de tijd die het versturen van de mailtjes zelf kost is hierbij te verwaarlozen). Ruim anderhalf uur dus. Het is maar goed dat ik daar niet bij hoef te zijn: ik zet het MailDoc programma gewoon aan, en ga lekker verder werken (of pak een goed boek – ik zie morgen wel hoe het afgelopen is).

Sleep?

En dan nu de honderduizend punten vraag: hoe las je een pauze in onder .NET? In Win32 kun je Sleep aanroepen uit de SysUtils unit, en ook onder .NET zit er een Sleep routine in Borland.Vcl.SysUtils. Waar de Win32 sleep gebruik maakt van de Windows sleep API (en onder Linux van de usleep library), daar maakt de sleep onder .NET gebruik van de System.Threading namespace. Dus de Sleep uit SysUtils zit eigenlijk in de System.Threading namespace.
Als we de SysUtils unit toevoegen aan de uses clause, en dan het programma compileren krijg je een interessante hint te zien: *Inline function sleep has not been expanded because unit System.Threading is not specifed in the USES list.*
Dit heeft te maken met het feit dat de sleep routine uit de SysUtils unit (voor .NET) eigenlijk een inline routine is, dus die ziet er als volgt uit:

```
procedure Sleep(milliseconds: Cardinal); inline;
```

Met als implementatie gewoon het aanroepen van Sleep uit de System.Threading namespace, wat als volgt gaat voor de huidige thread:

```
procedure Sleep(milliseconds: Cardinal);
begin
  System.Threading.Thread.Sleep(milliseconds);
end;
```

Het inline keyword wil zeggen dat de code die in de sleep routine staat (dus de code uit de SysUtils unit) letter moet worden opgenomen bij de aanroeper. Dus als we in ons MailDoc programma de aanroep Sleep(10000) neerzetten – een aanroep naar SysUtils' Sleep, dan zou dat eigenlijk vervangen moeten worden door System.Threading.Thread.Sleep(10000);
Helaas kan dat niet, want de System.Threading namespace is niet bekend in het MailDoc programma (deze namespace staat niet in de uses clause), en dus kan de compiler dan System.Threading.Thread.Sleep niet compileren. Niet binnen MailDoc. Maar wel binnen Borland.Vcl.SysUtils, want daar staat de System.Threading unit gewoon in de uses clause.
Toch is het een beetje overkill om de SysUtils unit te gebruiken alleen maar om sleep aan te kunnen roepen. Zeker als die sleep ook eens afkomstig is uit een hele andere unit, van het .NET Framework zelf. Dus stel ik voor om de SysUtils unit uit de uses clause te verwijderen, en System.Threading daarvoor in de plaats te zetten. De aanroep naar Sleep(10000) moet dan wel gewijzigd worden in Thread.Sleep(10000).

De System.Threading namespace zit in de .NET system unit zelf (mscorlib), en we hoeven dus niet een extra assembly aan de references lijst van het project toe te voegen (zoals we wel met de System.Web assembly moesten doen, om de System.Web.Mail namespace te kunnen gebruiken). De uiteindelijke – complete – versie van de toepassing is nu als volgt:

```
program MailDoc;
{$APPTYPE CONSOLE}
{%DelphiDotNetAssemblyCompiler
'$(SystemRoot)\microsoft.net\framework\v1.1.4322\System.Web.dll'}
uses
  System.Web.Mail,
  System.Threading,
  Classes;
var
  Msg: System.Web.Mail.MailMessage;
  Abonnees,Inhoud: TStringList;
  Adres: String;
begin
  Inhoud := TStringList.Create;
  Inhoud.LoadFromFile('message.txt');
  Abonnees := TStringList.Create;
  Abonnees.LoadFromFile('email.txt');
  SmtpMail.SmtpServer := 'mail.chello.nl';
  Msg := MailMessage.Create;
  Msg.From := 'b.swart@chello.nl';
  Msg.Subject := 'Delphi OplossingsCourant vol.7 no.2 - Delphi 2005';
  Msg.Body := Inhoud.Text;
  for Adres in Abonnees do
  begin
    Thread.Sleep(10000);
    Msg.&To := Adres;
    try
      SmtpMail.Send(Msg);
      writeln('Bericht verzonden naar ',Adres);
    except
      on E: Exception do
        writeln('Error: ',E.Message,' bij ',Adres)
    end
  end;
  readln
end.
```

Niet meer dan 35 regels code om – spamvrij – het uitkomen van de nieuwe Delphi OplossingsCourant te vermelden aan de lijst met abonnees.
Met op dat ik de twee TStringLists wel creeer, maar niet vrijgeef. Dat had ik wel kunnen doen, door Inhoud.Free en Abonnees.Free aan te roepen, maar het is onder het .NET Framework niet langer noodzakelijk om zelf je geheugen op te ruimen: dat doet de garbage collector voor je (het heeft zelfs geen direct effect om Free aan te roepen overigens, maar dat is een ander verhaal).

Samenvatting

In dit artikel heb ik een Delphi 2005 toepassing geschreven die enkele onderdelen van het .NET Framework gebruikt (zoals de MailMessage class, SmtpMail object, en de Sleep methode van het Thread object) alsmede enkele nieuwe Delphi 2005 taalelementen zoals de for-in-do loop, inline routines, en de & (die ook al in Delphi 8 for .NET zat overigens).
Uitbreidingen zijn natuurlijk mogelijk, maar voor mijn wensen voldoet het op dit moment prima, en het neemt me een hoop werk uit handen.

Delphi ASP.NET & DB Web

This article covers some of the enhancements in Delphi 2005 for building ASP.NET Web Applications (like the Deployment Manager), as well as the enhancements that Delphi 2005 brings to the ASP.NET world in the first place (like the Borland Data Provider and the DB Web controls).

Delphi 2005 and ASP.NET

With Delphi 2005 you can use Internet Information Server or the Cassini Personal Web Server as hosting web server for your ASP.NET projects. You can specify the default web project options in the ASP.NET page of the Tools Options dialog, as shown in Figure 1.

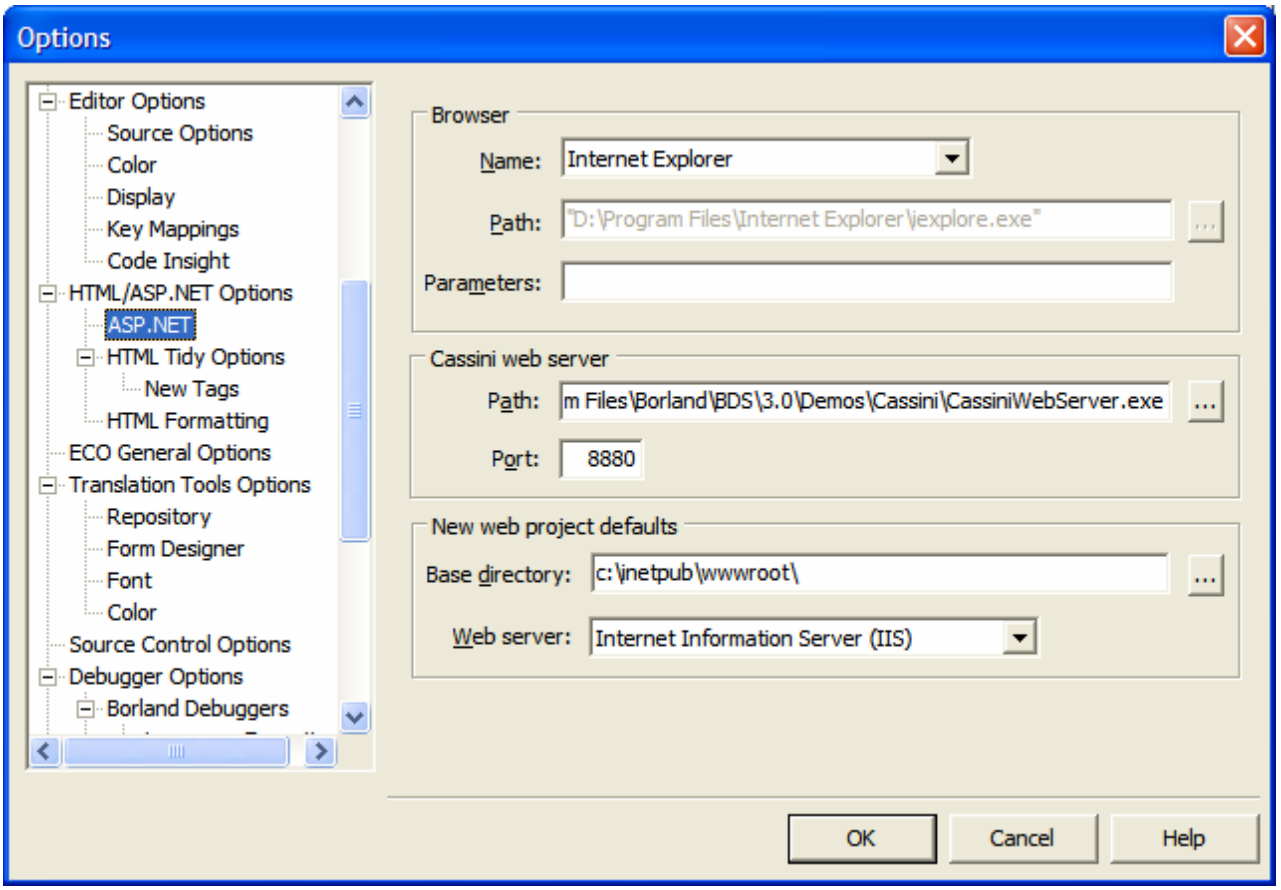


Figure 1. Delphi 2005 ASP.NET Options

Note that Cassini is only meant for local testing and debugging purposes – it's an Open Source Web Server, written in C#, and can be found in the BDS\3.0\Demos\Cassini directory.

There are a number of ASP.NET projects we can build with Delphi 2005 – from ASP.NET Web Applications to ASP.NET Web Services and ASP.NET Control Libraries. And that's not even counting the ECO editions of the ASP.NET Web Applications or Web Services, or mentioning the fact that all ASP.NET projects can be build using Delphi or C# as language. In this article, I want to build an example ASP.NET Web Application using the Delphi language, so we should do File | New – Other, go to the Delphi for .NET Projects page in the Object Repository, and select the ASP.NET Web Application icon (see Figure 2).

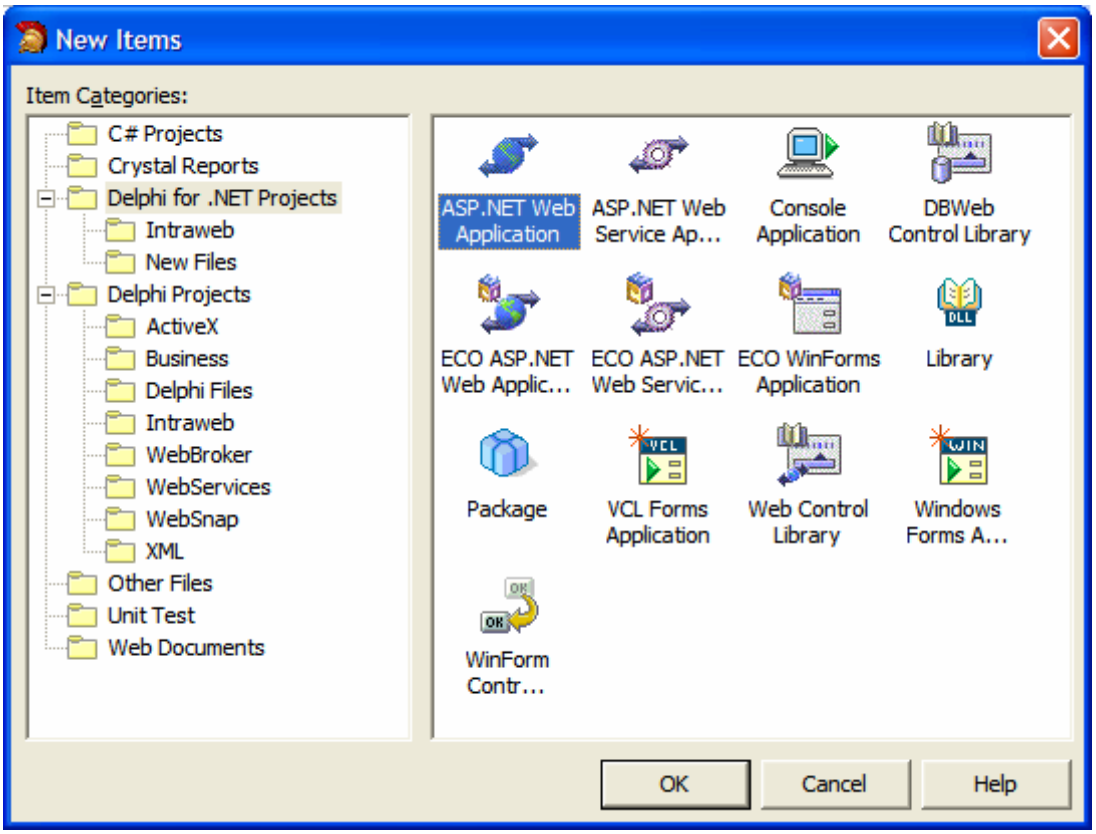


Figure 2. Delphi 2005 Object Repository – ASP.NET Web Application

When you click on OK, the New ASP.NET Web Application dialog is shown (see Figure 3). Here you can specify the name of the application as well as the location. If you use Internet Information Server (IIS) as hosting web server, then the location will also be registered with IIS as a virtual directory with read and scripting rights (but no execute rights, since ASP.NET Web Applications do not need that). You can view these detailed options if you click on the "View Server Options" button.
Note that even if Internet Information Server (IIS) is your default hosting server, you can still decide to use Cassini as local testing and debugging server. This can be useful if you're running Windows XP Home edition, which does not include Internet Information Server.

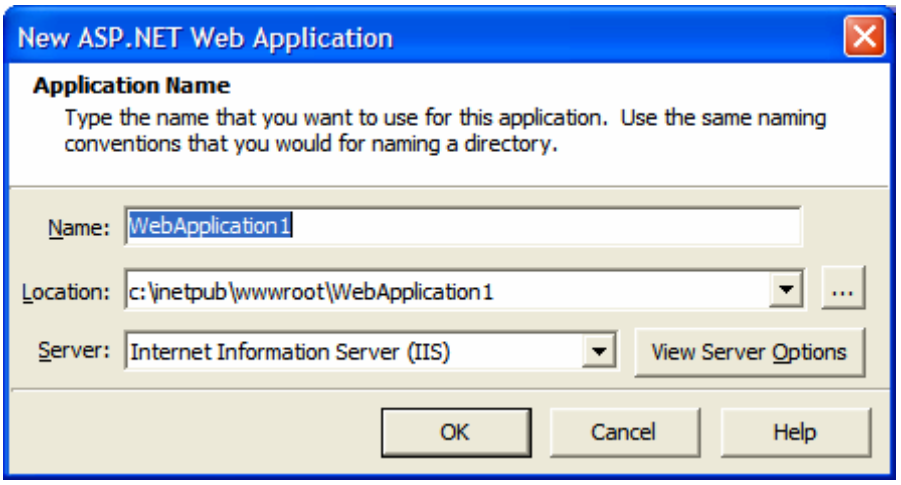


Figure 3. New ASP.NET Web Application

When you click on OK in the New ASP.NET Web Application dialog, a new project is created for you, in the WebApplication1 virtual directory.

ASP.NET Web Forms and Data Access

The name of the project is the name that you entered in the dialog of Figure 3, but the initial web form will be called WebForm1.aspx. You can rename that in the Project Manager, or do File | Save As and save it under a new name, such as Home.aspx.
We will use this first page to display a DataGrid with overview information from the Employees table of the SQL Server Northwind example database. The SQL scripts to generate this example database can be found on the MSDN website.

We can either use the BdpConnection or the SqlConnection components to connect to the SQL Server database/ With Delphi 8 for .NET or C#Builder, the use of SqlConnection would not lead to a situation where live data at design-time was possible. There was also no easy way to build the connection string for the SqlConnection component. This has all been enhanced in Delphi 2005.

Place a SqlConnection component from the Data Components category of the Tool Palette on the non-visual component area of the Web Forms designer. The SqlConnection component has a ConnectionString property, and Delphi 2005 now offers the helpful ConnectionString dialog (see Figure 4).
The SQL Server instance on my machine is called NX03\NX3, and for an ASP.NET web application you must ensure to specify a specific user name and password to login to the database, otherwise the data will be shown at design-time, but not at run time (since the default ASP.NET internet user will not get permission to access the SQL Server database – especially not if you're running on Windows 2003).

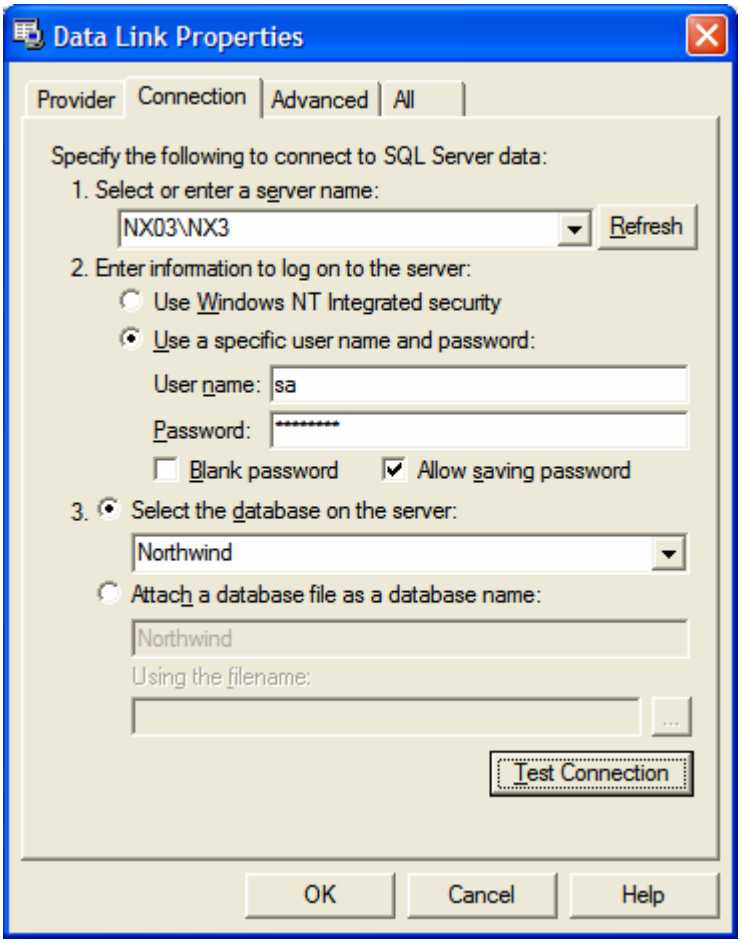


Figure 4. SqlConnection ConnectionString dialog

After you've specified the server name and user name with password, you can open up the drop-down combobox to select a database on the server. For the example in this article, we should use the Northwind database, but any other database will also do fine.
Make sure to click on the Test Connection button to verify that you can connect to the specified database on the server using the user name and password settings.
Once the ConnectionString is ready, you can close the dialog and place a SqlDataAdapter component next to the SqlConnection component. An ADO.NET Data Adapter has four properties that correspond to the four basic SQL statements: SELECT, INSERT, UPDATE, and DELETE. The obvious names for these properties are SelectCommand, InsertCommand, UpdateCommand and DeleteCommand. At this time, we only need the SelectCommand, so in the Object Inspector open up the SelectCommand property of the SqlDataAdapter. Point the Connection sub-property of the SelectCommand property to the SqlConnection component. We can now specify a value for the CommandText property, which should be an SQL SELECT statement on the Northwind database, such as:

```
SELECT EmployeeID, FirstName, LastName, BirthDate FROM Employees
```

DataSync and DataHub

It's time to add two new components of Delphi 2005. From the Borland Data Provider category in the Tool Palette, place both a DataSync and a DataHub component next to the SqlConnection and SqlDataAdapter components.
The DataSync component maintains a list of data providers, where each data provider is associated to a DataAdapter – which can be any ADO.NET DataAdapter, including the Bdp DataAdapter or the SqlDataAdapter of course. Double-click on the Providers property of the DataSync component, to start the Providers Collection Editor (see Figure 5). Here, you need to add a DataProvider, point its DataAdapter property to the SqlDataAdapter component. You can optionally change the TableName property as well. If you have a list of several Data Providers, make sure that each DataProvider uses a unique TableName to hold the result of the SelectCommand from the associated DataAdapter.

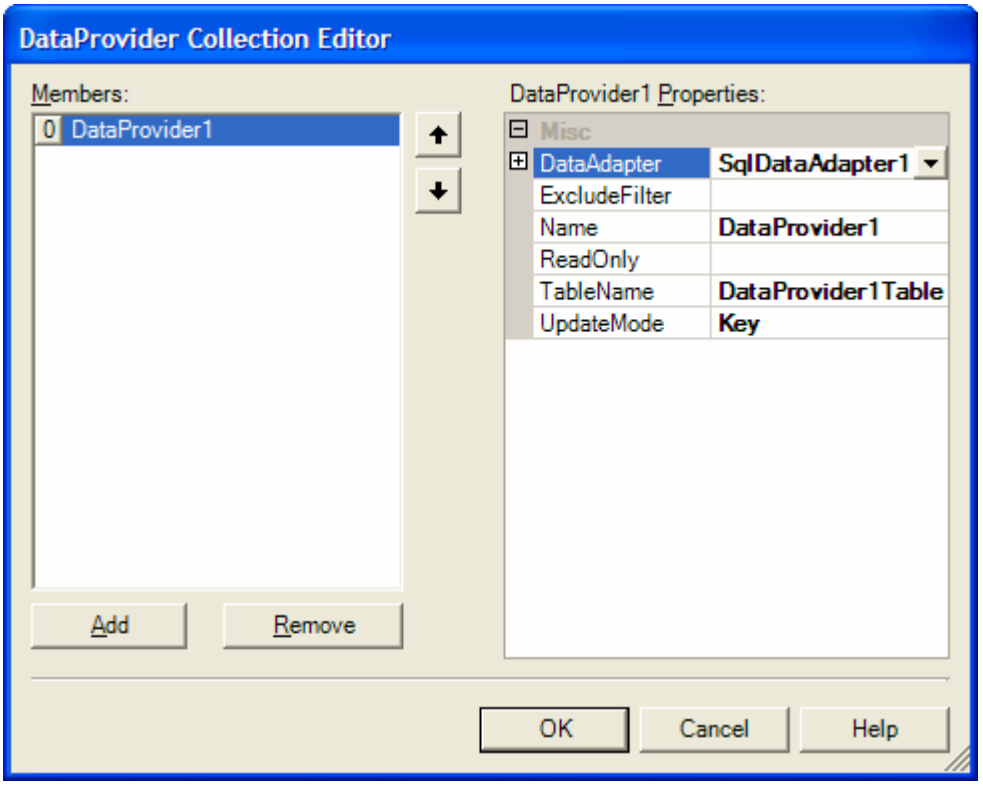


Figure 5. DataSync DataProvider Collection Editor

After you've finished the Providers Collection Editor, turn to the DataHub component. Point the DataPort property to the DataSync component, so it can retrieve the result from the SQL Select commands from the DataProviders. In order to store the results, we need a DataSet and hook it up to the DataSet property of the DataHub component. We can find a DataSet component in the Data Components category of the Tool Palette. Once you've placed one in the Web Forms designer, you can hook it up to the DataSet property of the DataHub component. The last step consists of setting the Active property of the DataHub to True. The DataHub will now send a request to the DataSync component, asking it to send the DataTables for all its associated DataProviders. Time to add the user interface controls.

DB Web Controls

Delphi 2005 contains a special set of visual and data-aware ASP.NET controls, called the DB Web Controls. They are based on the design pattern that we could find in Delphi version 1 already (now over 10 years ago), with a DataSource component as gateway between the data access (or DataSet) component and the visual controls.

The most important DB Web control is the DBWebDataSource – this one acts as the gateway between a .NET DataSet or DataTable, and the other DB Web visual controls. From the DB Web category, place a DBWebDataSource component on the Web Form designer. Assign its DataSource property to the .NET DataSet (that's also connected to the DataHub component).

DBWebGrid

Now, place a DBWebGrid on the Web Form. You can best do this by double-clicking on the DBWebGrid control in the Tool Palette, so it will be placed in flow-layout. Make the DBDataSource property of the DBWebGrid point to the DBWebDataSource component, and select DataProvider1Table (the only choice) as value for the TableName property. This will show live data at design-time in the DBWebGrid, something that was not possible before Delphi 2005 (while using the regular SqlConnection component to talk to SQL Server).

You can now format the DBWebGrid, by clicking on the Auto Format verb at the bottom of the Object Inspector. Set the color scheme to Colorful 1, for example. Using the Property Builder, you can customize the DBWebGrid even further.

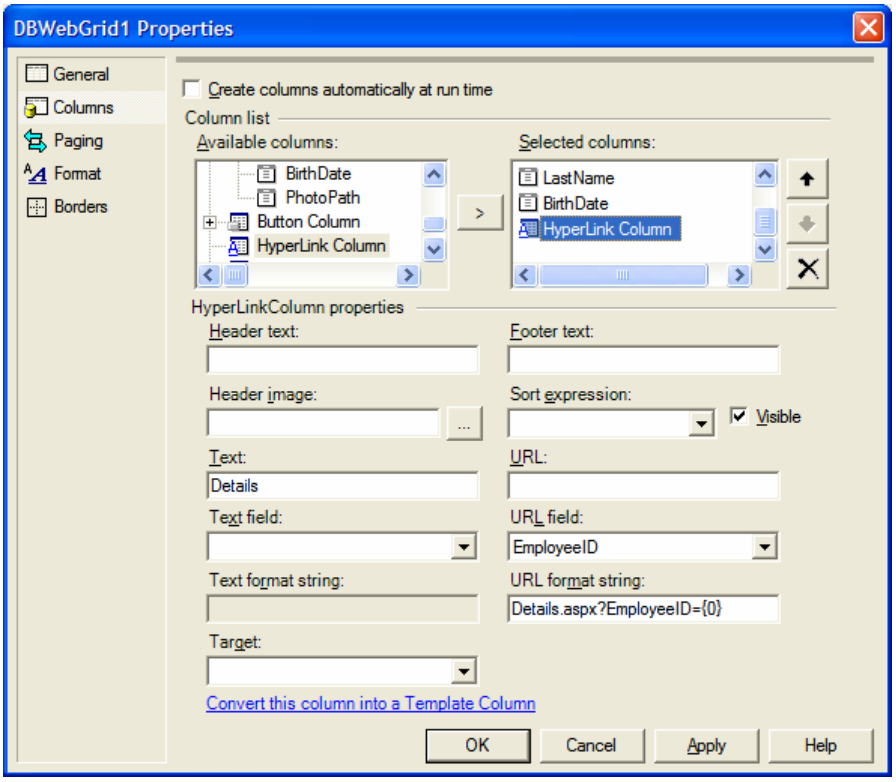


Figure 6. DBWebGrid Properties Dialog

In the General page of the DBWebGrid Properties dialog, you need to select the DataProvider1Table as DataSource property, to make sure we can connect to columns at design-time. In the Columns page, uncheck the "Create columns automatically at run time" option, and instead explicitly add all columns through the (All Fields) option. This will add all fields into the Selected columns listbox.
We can specify additional property settings, like the Data formatting expression of the BirthDate field, by setting it to {0:d}.

The DBWebGrid will show only a subset of all fields from the Employees table, and we should add a HyperLink column that can be used to jump to a detail table with all fields. Set the Text property of the HyperLink column to Details, the URL field to EmployeeID, and the URL formatting string to Details.aspx?EmployeeID={0}, as shown in Figure 6.

At this time, we should have a nice DBWebGrid, filled with four data fields and a hyperlink column, as can be seen in Figure 7 (the HyperLink isn't visible in Figure 7, but we'll see that in a next screenshot of Figure 8).

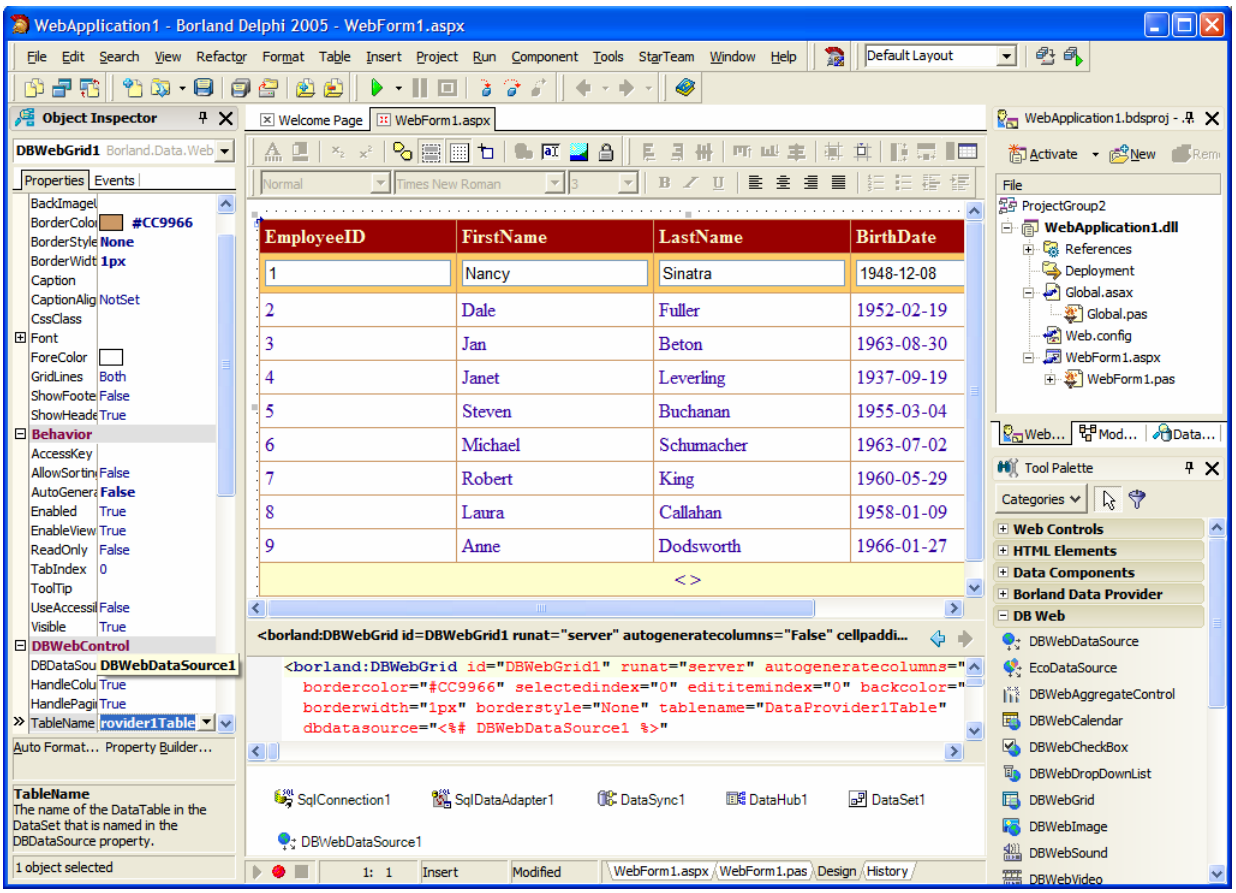


Figure 7. Live Data at Design-Time

There are two more things I want to change before we continue with the Details.aspx page. First of all, the DBWebGrid is set in edit-mode by default, but I want it to be used for an overview of the Employees table only – without the option to edit the contents. For that, we need to set the Readonly property of the DBWebGrid to True.

The other change involves adding a DBWebNavigator – placing it above the DBWebGrid – so we can navigate through the Employees table if you wish. Make sure to remove all buttons from the DBWebNavigator except for the First, Prev, Next and Last buttons.

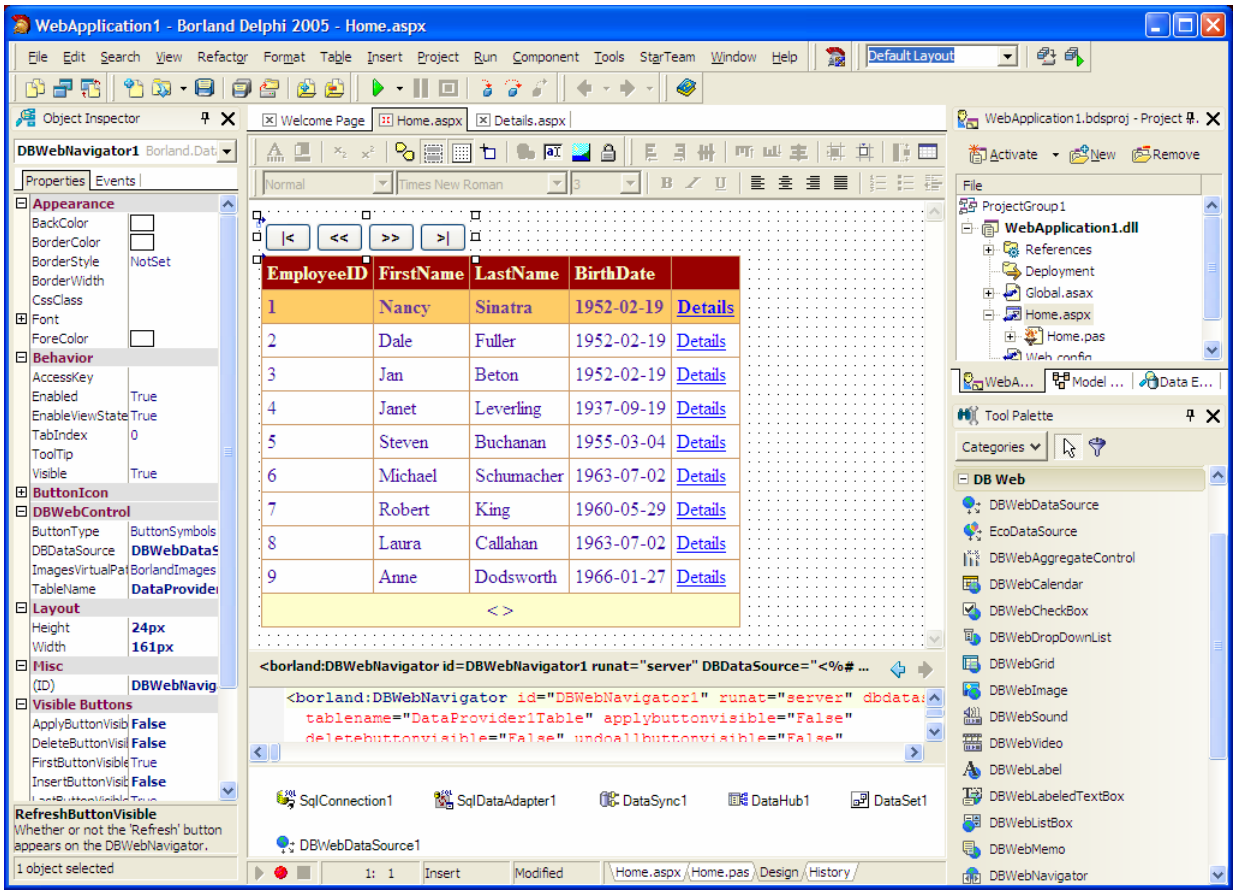


Figure 8. Read-only DBWebGrid with DBWebNavigator

This results in the overview of Figure 8, where we can navigate through the Employees records, and click on the Details hyperlink to get redirected to a new web page called Details.aspx that we should build now.

Details Page

To add a new ASP.NET Web Page to the project, do File | New – Other, and double-click on the ASP.NET Page in the New ASP.NET Files category (see Figure 9).

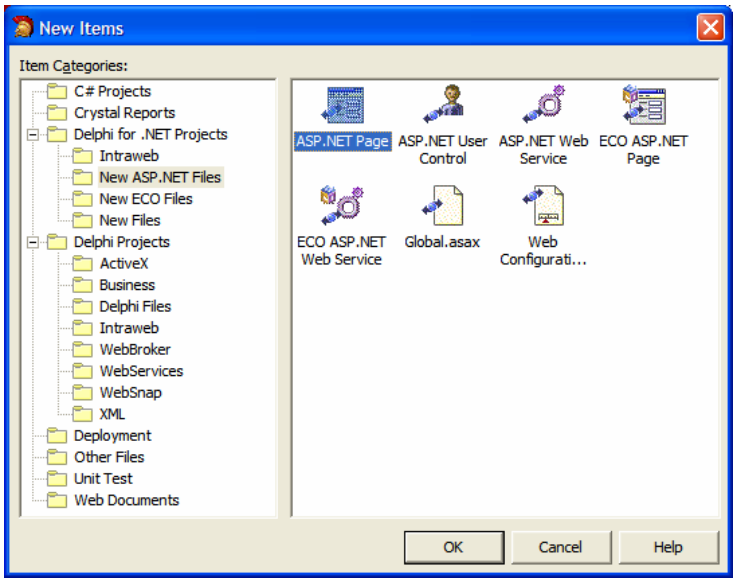


Figure 9. Delphi 2005 Object Repository – New ASP.NET Files

Instead of using the SqlConnection ADO.NET components, we can also use the Borland specific Borland Data Provider components. These consists of a single set of data access components that can be used to work with InterBase, MS SQL Server, IBM DB2, Oracle, and more.
For our example, we can go to the Data Explorer and right-click on the MS SQL Server connection node to configure the connection to the Northwind database. Note the differences between Figure 10 (BDP) and Figure 4 (ADO.NET), the main difference being the fact that the ADO.NET dialog allows us to select a database from a list of items, and in the BDP dialog we need to enter the name of the database manually.

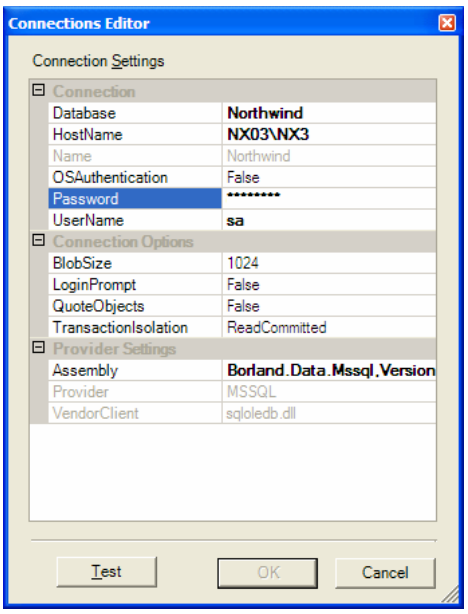


Figure 10. BdpConnections Editor

Once the connection works – which can be verified with the Test button – we can place both a BdpConnection component and a BdpDataAdapter component in the non-visual components area of the Web Forms designer in the Details.aspx page.
Right-click on the BdpDataAdapter to start the Data Adapter Configuration dialog, which can be seen in Figure 11.

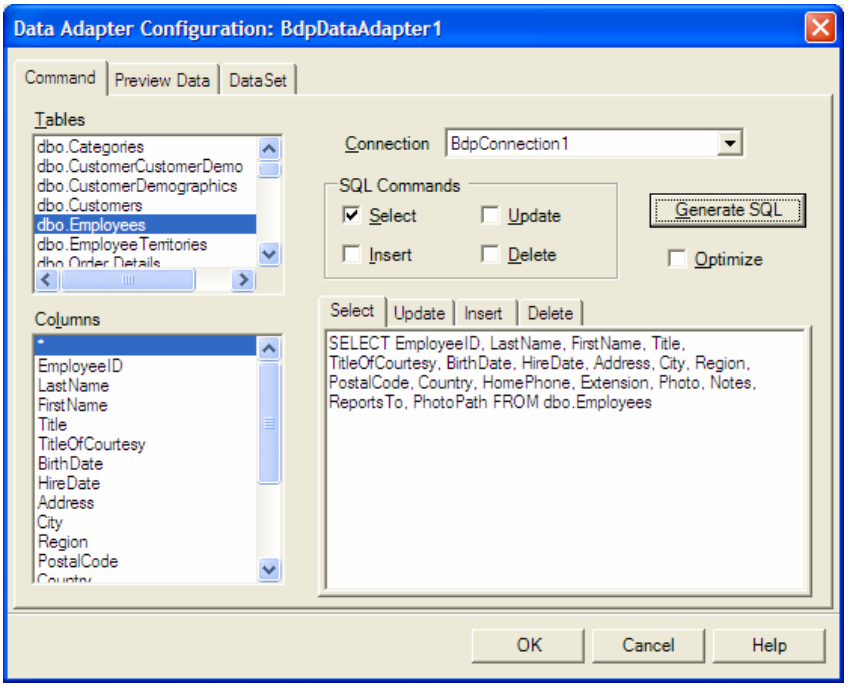


Figure 11. BDP Data Adapter Configuration Dialog - Command

Although we want to use the Details.aspx page to edit and modify the contents of the Employees records, we only need to specify the Select command in the BDP Data Adapter Configuration dialog. So uncheck the Update, Delete and Insert options (as seen in Figure 11). These SQL commands will be generated automatically when we call the AutoUpdate method later.

The SQL Select command is as follows:

```
SELECT EmployeeID, LastName, FirstName, Title, TitleOfCourtesy,
  BirthDate, HireDate, Address, City, Region, PostalCode, Country,
  HomePhone, Extension, Photo, Notes, ReportsTo, PhotoPath
FROM dbo.Employees
```

In the Preview Data tab of the Data Adapter Configuration dialog, we can get a preview of the results from the SQL Select command. In case you're connecting to a really big database table, with lots of records, it's good to know that you can limit the number of records, so the Delphi 2005 IDE won't have to spend a long time waiting for the dataset.

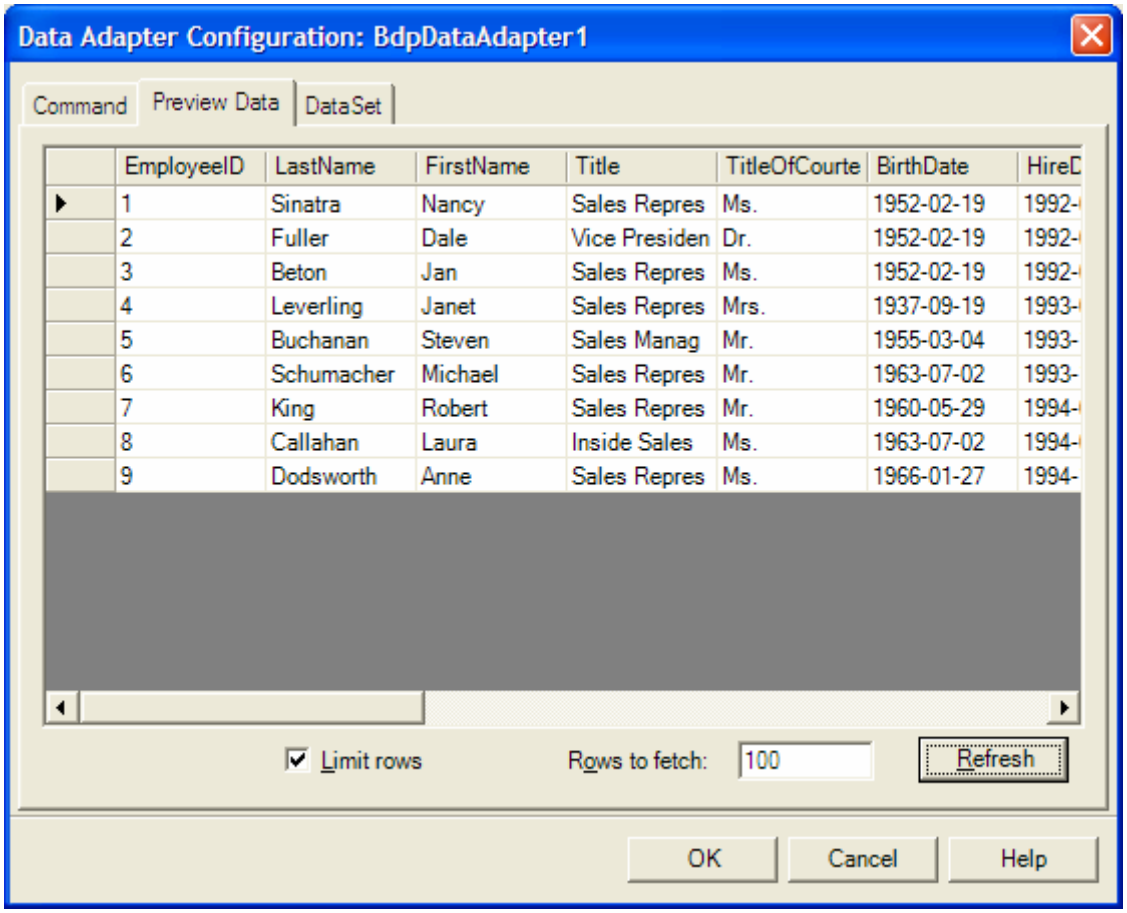


Figure 12. BDP Data Adapter Configuration Dialog – Preview Data

A big feature of the BdpDataAdapter is the support for live data at design-time. This is supported by a .NET DataSet property which can be filled with a DataTable (holding the result of the SQL Select command) at design-time. The .NET DataSet should be assigned to the DataSet property of the BdpDataAdapter, and in the DataSet tab you can either select an existing DataSet component to connect to, or explicitly create a new .NET DataSet and connect the DataSet property to it automatically.

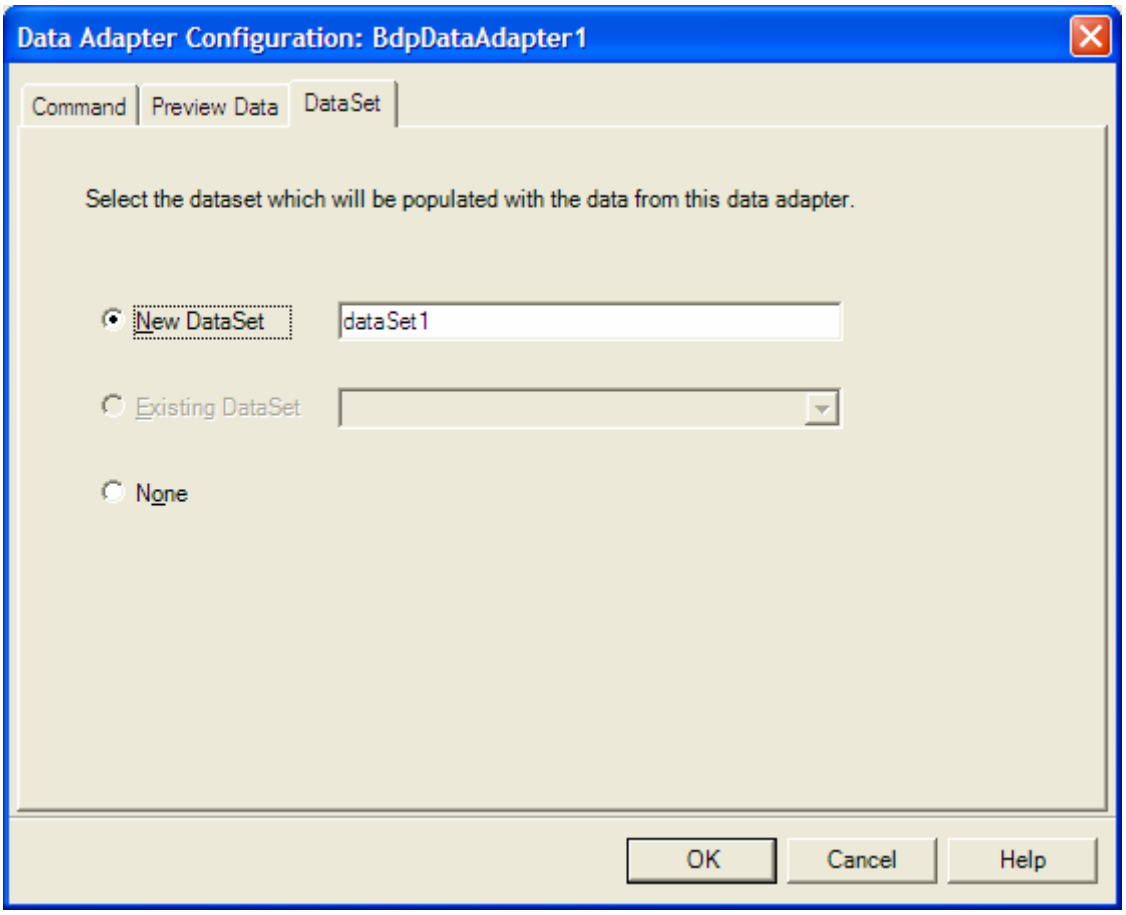


Figure 13. BDP Data Adapter Configuration Dialog - DataSet

As you can see in Figure 13, in our case, there is no .NET DataSet yet, so we should create a new .NET DataSet, which will be assigned to our DataSet property as well.
Now, close the Data Adapter Configuration Dialog. The new .NET DataSet will be added to the non-visual component area of the Web Forms designer. But it's still empty at this time. In order to "fill" it with a DataTable, we need to set the Active property of the BdpDataAdapter to True. This will execute the SQL Select command, and place the result in a DataTable called Employees (this name is based on the table name in the Select command).

The Right EmployeeID

Note that the SQL Select command still retrieves all Employees from the database – and not the specific Employee based on the EmployeeID we passed to the Details.aspx page. We need to ensure that we only get the right Employee, of course, and this can be done in the Page_Load event of the Web Form. Double-click on the Web Form designer area, and write the following code in the Page_Load event handler:

```
procedure TWebForm1.Page_Load(sender: System.Object;
  e: System.EventArgs);
begin
  BdpDataAdapter1.Active := False;
  if Assigned(Request.QueryString['EmployeeID']) then
    BdpDataAdapter1.SelectCommand.CommandText :=
      BdpDataAdapter1.SelectCommand.CommandText +
      ' WHERE EmployeeID = ' +
      Request.QueryString['EmployeeID'].ToString;
  BdpDataAdapter1.Active := True;
end;
```

Note that it seems unnecessary to set the BdpDataAdapter's Active property to False and then back to True again, but the last assignment will actually (re)fill the DataSet with the Employees table. This time with only one record inside. If it makes you feel better, you can set the Active property of the BdpDataAdapter to False at design-time, so the SQL Select command doesn't get executed unnecessarily (before the Page_Load).

We can now place a DBWebDataSource component next to the DataSet component, and connect its DataSource property to this DataSet. In the overview page, the DBWebDataSource would cache its contents, but this time I want it to refresh the contents and always show the latest version. For that, we must set the AutoRefresh property of the DBWebDataSource to True.

We can then add several DB Web controls to show the contents of the selected Employee record. In order to format these controls nicely, we should start by inserting a HTML table on the form, using Table | Insert - Table or just the Insert | Table menu option. As you can see in Figure 14, we can define the number of rows, columns, width (in pixels or percentage) as well as background colors, border size and more. It's important – in my view – to set the Border Size to 0, which will still show the border at design-time, but no longer at runtime.

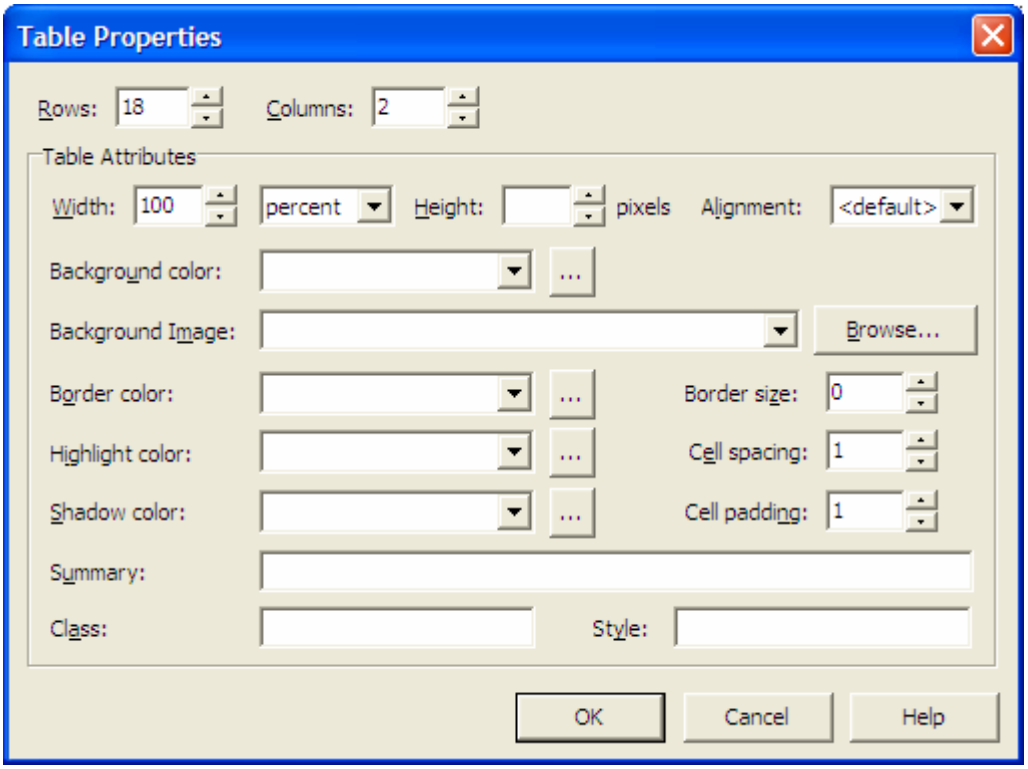


Figure 14. HTML Table Properties Dialog

In the left column of the HTML Table, you can enter the name of the field – just type some text in it. You can use the Object Inspector to set the align property of the table cell, for example to right in order to right-align the left column. In the right column, you can place DB Web controls. For the read-only EmployeeID field (which is the key field), I use a DBWebLabel control, and for the two date fields I use a DBWebCalendar, but for the other fields I use a DBWebTextbox control, as can be seen in Figure 15.

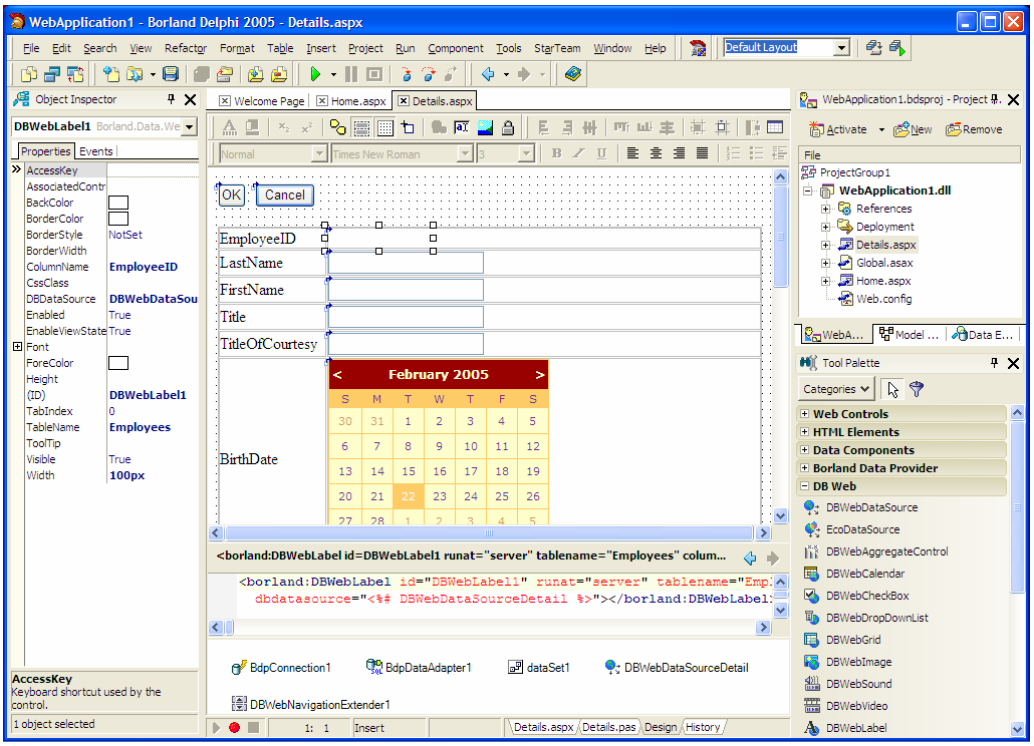


Figure 15. Details Design with DB Web Controls

Since the details page is only used to display and edit one record, I have no need to use the DBWebNavigator control. However, with Delphi 8 for .NET, the DBWebNavigator control was the only component that held the "Apply To Server" button in order to apply the changes back to the database. With Delphi 2005, there is a new component called the DBWebNavigationExtender that we can use instead of the DBWebNavigator. The DBWebNavigationExtender is not a navigator itself, but it will extend normal ASP.NET buttons with three design-time properties to connect to a DBWebDataSource, a TableName and a specific Action. One of these actions is Apply, so we only have to place a regular ASP.NET button on the Web Form, point it to the DBWebDataSource and Employees tablename, and set its DataSourceAction property to Apply.
Calling Apply in itself is not enough: we must tell the DBWebDataSource what to do when the updates are applied. We can do this in the OnApplyChangesRequest event handler, and implement it as follows:

```
procedure TWebForm1.DBWebDataSource1_OnApplyChangesRequest(
  sender: System.Object;
  e: Borland.Data.Web.WebControlEventArgs);
begin
  BdpDataAdapter1.AutoUpdate
end;
```

As you see, this is the place where we call the BdpDataAdapter.AutoUpdate method, which will use the SQL Select command and information about the table (the primary key mainly) to construct the SQL Update, Insert and Delete commands (with the help of a BdpCommandBuilder component), and then performs the updates.
A second button can be used to return to the overview page – Home.aspx – and cancel the updates, so set the DataSourceAction property to Cancel. This button's Click event is implemented as follows:

```
procedure TWebForm1.Button2_Click(sender: System.Object;
  e: System.EventArgs);
begin
  Session.Remove('Employees_CurrentObj'); // important!!
  Response.Redirect('Home.aspx');
end;
```

Note that the Session.Remove statement is required in order to ensure that the DBWebDataSource will no longer "remember" which Employee record we are working on when we return to the overview page. Without removing the Employees_CurrentObj from the session, you get problems if you go to the Details.aspx page to modify a record, return to the overview page, and then go to the Details.aspx page again to try to modify a different record.

Deployment

Before Delphi 2005, we had to manually deploy the ASP.NET Web Application. But Delphi 2005 now includes a Deployment Manager, that can be used for ASP.NET projects, IntraWeb projects or other projects (with some customization). For our ASP.NET project, there already is a Deployment node in the Project Manager (as can be seen in the upper right corner of Figure 16). If you right-click on this node, you can select the Deployment choice. This will give you the Deployment Manager window that you can see fully in Figure 16. The left side of the window shows the files in your project that need to be deployed. The right side shows the files in the deployment target location – by default this is empty. You can select a specific target: either a folder on your local machine or network, or an FTP connection. When the deployment target is created, you can click on the second button from the left to actually copy the files from the project to the deployment location. This is a great help, especially since the Deployment Manager automatically detects and displays the files that have been modified since the last time you deployed the project. There's no way you can forget to deploy a file or deploy an older version.

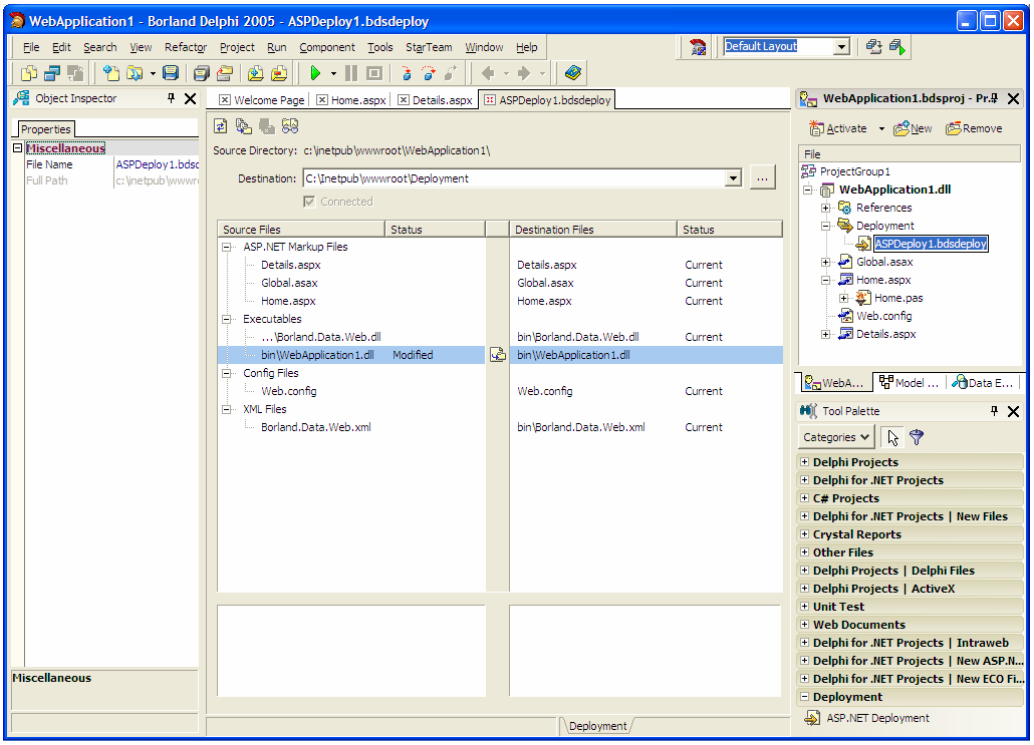


Figure 16. Deployment Manager

Summary

In this article, I have developed and deployed an example, but non-trivial ASP.NET Web Application, using Delphi 2005 and some of the new ASP.NET enhancements in Delphi 2005. We've used both the ADO.NET and Borland Data Provider to connect to and work with SQL Server, and examined the Borland DB Web controls to build overview as well as data-entry forms. Finally, I've demonstrated the Borland Deployment Manager to quickly deploy ASP.NET Web Applications.

Note that deployment of the BDP assemblies may not be easy, and is in fact the topic of an article of its own, available at http://www.drbob42.com/examines/examin66.htm.

Finally, as mentioned before, some of my own Delphi 2005 ASP.NET applications are can be seen online at http://www.eBob42.com/courseware and http://www.eBob42.com/blog