Delphi OplossingsCourant

Vol. 6. No. 1, Een gratis electronische publicatie van **Bob Swart Training & Consultancy** (**eBob42**) - http://www.eBob42.com

Helmond, 9 januari 2004,

Welkom bij het nieuwe nummer van de Delphi OplossingsCourant. Het is misschien al wat aan de late kant, maar ik wens iedereen toch nog de het allerbeste voor het nieuwe jaar. En dat geldt ook voor de mensen die vorig jaar hun baan of andere dierbare zaken zijn kwijtgeraakt. Ook ik heb in 2003 een minder jaar meegemaakt vergleken met 2002, al had dat met name te maken met de daling van de dollar. En ook het feit dat er in 2003 geen nieuwe versie van Delphi is verschenen, waardoor de deelname aan de trainingen terugliep, speelde mee. Gelukkig werd ik wat vaker gevraagd voor advies en ontwerp & bouw (met name op het gebied van web toepassingen en web services).
Formeel heb ik overigens geen gelijk met de bewering dat er in 2003 geen nieuwe versie van Delphi is verschenen, want in de week voor Kerst is Delphi 8 for .NET dan toch uitgekomen. Dit nummer van de Delphi OplossingsCourant staat dan ook geheel in het teken van deze nieuwe versie.
Is Delphi 8 for .NET het wachten waard geweest? Mijn eerste ervaringen zijn in ieder geval positief (maar ik ben nu eenmaal een optimist, en zie ook 2004 met vertrouwen tegemoet).

Nieuwe Clinics in 2004

Het uitkomen van Delphi 8 for .NET betekent in ieder geval ook een nieuw aanbod van Delphi Clinics, en een aanpassing van de bestaande Clinics (waarin ik nu desgewenst zowel de onderwerpen met Delphi 7 als Delphi 8 for .NET laat zien).
Het begint op **22 januari** (herhaling **27 mei**) met de Delphi 8 for .NET Clinic waarin ik alle hoeken van de IDE laat zien (ook waar nog geen documentatie over is), de overgang van VCL naar VCL for .NET, het maken en gebruik van .NET assemblies, WinForms, en alle verschillende database benaderingen (BDE, IBX en dbExpress voor VCL for .NET, en BDP en ADO.NET voor niet-VCL). Ook Enterprise Core Objects: het "model driven" design met object persistence raamwerk komt aan de orde.

Op **19 februari** (herhaling **24 juni**) volgt de speciale ASP.NET dag, waarin ik laat zien hoe we web toepassingen voor het .NET Framework maken met Delphi 8 for .NET. Ook ASP.NET Web Services komen aan de orde, en als alternatief voor het bouwen van multi-tier toepassingen laat ik zien hoe .NET Remoting werkt in Delphi 8 for .NET.
Later in 2004 zal ik nog trainingen verzorgen over Delphi en databases: dbExpress, DataSnap en BDP / ADO.NET (op **25 maart**), en over IntraWeb voor Win32 alsmede IntraWeb for .NET (**22 april**)

Wie liever zelf een curriculum met onderwerpen wil samenstellen, heeft de mogelijkheid om mij een maatwerk training / workshop te laten organiseren. Dat kan bij mij thuis (voor twee tot maximaal drie personen), maar ik kan ook op bezoek komen.
Zie http://www.eBob42.com/workshop of de laatste pagina van de OplossingsCourant voor details.

Delphi OplossingsCourant

In voorgaande jaren is de Delphi OplossingsCourant altijd drie keer per jaar verschenen, maar dat is me in 2003 niet meer gelukt. Vanaf nu zal nog maar twee keer per jaar een nummer verschijnen: rond de jaarwisseling (na de kerstvakantie), en aan het eind van de zomervakantie. Een andere aanpassing is dat de artikelen langer zullen zijn (meer details dus), en niet meer in twee kolommen, maar gewoon in één kolom opgemaakt zijn. Dat heeft als voordeel dat de source code beter leesbaar wordt, en ik ook grotere screenshots kan gebruiken (zonder dat iedereen meteen een vergrootglas nodig heeft).

Wie na lezen van de Delphi OplossingsCourant nog opmerkingen heeft, kan gerust een mailtje sturen naar bob@eBob42.com- ik stel alle feedback op prijs (dat geldt ook voor vragen, opmerkingen of verzoek om andere ondersteuning die ik kan bieden).

De Delphi OplossingsCourant (DOC) is een gratis productie van Bob Swart Training & Consultancy (eBob42).

Eindredactie: Bob Swart e-mail: doc@eBob42.com

Delphi 8 for .NET

Net als Kylix (Delphi for Linux) hebben we heel lang moeten wachten op Delphi 8 for .NET. Maar vlak voor Kerst 2003 kwam het er eindelijk van: met de officiële naam Borland Delphi 8 for the Microsoft .NET Framework is Delphi - na de preview command-line compiler - nu ook een volwaardige ontwikkelomgeving onder .NET geworden.

Wie de overstap van Delphi 6 of 7 naar Delphi 8 for .NET maakt zal merken dat er heel wat veranderd is, met name op het gebied van databases. Delphi voor Windows kent de BDE voor dBASE, Paradox en FoxPro, en SQL Links (inmiddels "deprecated") voor grote databases als Oracle, Microsoft SQL Server, IBM DB2, Informix, Sybase en InterBase. Deze laatste kun je ook met IBExpress benaderen, en er is sinds Delphi 6 ook een verzameling cross-platform componenten onder de naam dbExpress (om met InterBase, Oracle, SQL Server, DB2, Informix en MySQL te werken).
Delphi 8 for .NET biedt ondersteuning voor de BDE (maar niet voor SQL Links), IBExpress en dbExpress - voor VCL for .NET toepassingen - en ADO.NET alsmede de Borland Data Providers for .NET voor alle andere .NET toepassingen (zoals WinForms of Web Forms).

VCL for .NET of niet?

Wie met Delphi 8 for .NET gaat werken zal merken dat er twee mogelijkheden zijn. Enerzijds kun je VCL for .NET toepassingen bouwen, en anderszijds niet-"VCL for .NET" toepassingen (en dat zijn dan WinForms, Web Forms of alles wat in ieder geval niks met de VCL for .NET te maken heeft). Dit is een grove tweesplitsing, die wel wat gevolgen heeft, en waarvoor je aan het begin van een nieuw project een bewuste keuze moet maken. Het voordeel van VCL for .NET is dat je zonder al te veel moeite deze toepassingen ook met Delphi 7 tot een Win32 toepassing kunt compileren. Daarnaast maakt de VCL for .NET gebruik van GDI voor het grafische werk, terwijl WinForms gebruik maakt van het langzamere GDI+. Het verschil is niet zo heel groot, maar is er wel.
De VCL for .NET toepassingen hebben de beschikking over de database componenten die we kennen uit Delphi voor Windows, namelijk met een TDataSet en een TDataSource. Alle niet VCL for .NET toepassingen werken net even anders, en gebruiken een .NET DataSet - die nog het meest te vergelijken is met een TClientDataSet - en helemaal geen TDataSource. De data-aware componenten die we kennen uit Delphi voor Windows zijn onder .NET dan ook uitsluitend beschikbaar voor VCL for .NET toepassingen, net als de Data Module overigens, wat ook een specifieke Borland oplossing is, en niet standaard onder .NET beschikbaar (maar dus wel voor VCL for .NET).

.NET Data Access

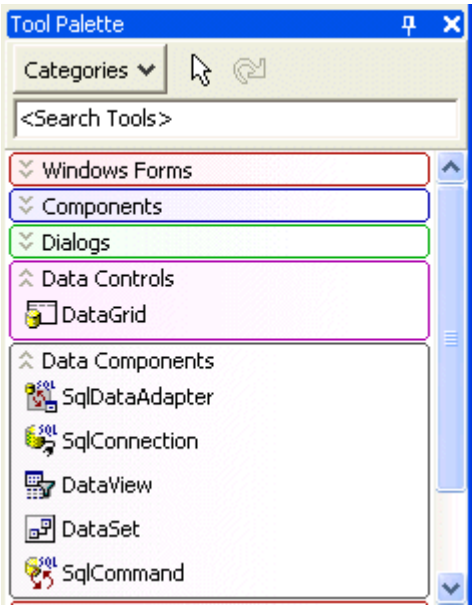
Wat biedt het .NET Framework dan op database gebied? Welnu, Microsoft heeft pasgeleden een speciale "uitgekleedde" editie van de SQL Server beschikbaar gemaakt voor gratis gebruik. Het betreft hier MSDE, en deze wordt ook bij C#Builder en Delphi 8 for .NET bijgeleverd. Met deze database kun je onder .NET je eigen database toepassingen bouwen, en kun je daarbij gebeuik maken van de .NET specifieke database componenten genaamd ADO.NET. Delphi 8 for .NET bevat naast ADO.NET ondersteuning ook een verzameling Borland Data Providers for .NET bevatten, met drivers voor SQL Server / MSDE, Oracle, IBM DB2, Informix en zelfs MS Access. De Borland Data Providers zijn een uitbreiding van ADO.NET, en bieden meer design-time ondersteuning.
De rest van dit artikel wil ik besteden aan het introduceren van de ADO.NET manier om met databases te werken, en dan met name met de gratis MSDE. Op mijn website zijn al verschillende artikelen te vinden over het gebruik van de Borland Data Providers for .NET, zie o.a. http://www.drbob42.com/IBM voor artikelen over Delphi 8 for .NET en DB2.

ADO.NET

De onderstaande tabel laat nog eens in het kort zien wat op dit moment de database ondersteuning is in Delphi 8 for .NET. Ten opzichte van Delphi 7 missen we SQL Links (maar die was deprecated) en dbGo for ADO. Wat betreft deze laatste is het niet uitgesloten dat Borland t.z.t. nog met een update voor Delphi 8 for .NET komt waarin dbGo for ADO alsnog in is opgenomen. De tijd zal het leren.

VCL for .NET			.NET (WinForms & ASP.NET)	
BDE	**IBExpress**	**dbExpress**	**BDP for .NET**	**ADO.NET**
dBase	InterBase	InterBase	InterBase	ODBC.NET
Paradox		SQL Server	SQL server	SQL Server
FoxPro		Oracle	Oracle	Oracle
		DB2	DB2	OLEDB
		Informix	MS Access	...
		MySQL		
		SQL Anywhere		

Om gebruik te maken van ADO.NET met Delphi 8 for .NET kun je naast Delphi 8 for .NET (die zelf het .NET Framework 1.1, .NET SDK 1.1, en ook de Visual J# runtime nodig heeft), ook de MSDE sp3a installeren.
Zoals de tabel laat zien, kunnen we ADO.NET het beste gebruiken in een WinForms toepassing, dus we starten Delphi 8 for .NET en beginnen een nieuwe WinForms toepassing. Hier zal ik de ADO.NET componenten aan toevoegen, die overigens te vinden zijn in de Data Components categorie van het Tool Palette.



Maar voor ik daadwerkelijk componenten ga toevoegen, eerst nog een korte uitleg over hoe ADO.NET in elkaar zit, en welke rol de SqlConnection, SqlCommand, SqlDataReader, SqlDataAdapter, DataSet, DataTable, DataView, en SqlCommandBuilder spelen (niet allemaal zichtbaar in de Tool Palette overigens).

Connected vs. Disconnected

Je kunt namelijk op twee manieren met ADO.NET werken: direct verbonden (connected) met de database, of niet verbonden (ook wel disconnected genoemd). Dit laatste heeft de grootste kracht, omdat je hiermee een dataset kunt vullen met het resultaat van een zoekopdracht, en deze data naar een .NET DataSet kunt "sturen" waar hij vervolgens verwerkt kan worden. En dat heeft zelfs voordelen op je eigen

lokale machine, want het alternatieve connected model heeft ook als eigenschap dat het resultaat niet gewijzigd mag worden, en alleen van begin (eerste record) tot eind (laatste record) gelezen mag worden.
De SqlConnection is het component dat je gebruikt om een verbinding met de database te maken. Met de SqlCommand kun je daar vervolgens SQL statements en commando's naartoe sturen, al dan niet met een resultaat (na een select statement), welke in een SqlDataReader geplaatst kan worden. Dat is dan wel een read-only unidirectional resultaat. De .NET DataSet wordt gebruikt in het disconnected model, en wordt dan gevuld worden met behulp van een SqlDataAdapter, die properties bevat voor de vier basis commando's: SelectCommand, UpdateCommand, InsertCommand en DeleteCommand. Op basis van een SelectCommand kan de SqlCommandBuilder de andere drie voor je genereren - mits er een index of primary key informatie aanwezig is.

ADO.NET SqlConnection

Laten we een voorbeeld bouwen, en dan kun je vanzelf zien wat het verschil is tussen de conenected en disconnected benaderingen (en leer je meteen een beetje hoe je ADO.NET met Delphi 8 for .NET kunt gebruiken, wat nooit weg is).
In ga ervanuit dat je een WinForms project bent gestart, dus zet nu een SqlConnection component op het WinForms - dit component kun je vinden in de Data Components categorie van het Tool Palette. Met dit component kunnen we de verbinding met een SQL Server of MSDE opzetten, en doen dat door een waarde in de ConnectionString te zetten. Voor iedere database is die weer anders, maar het daaropvolgende gebruik van de componenten is exact hetzelfde. Dus wie meer wil spelen zal even moeten kijken naar de juiste ConnectionString, maar als die eenmaal werkt dan is de rest niet zo'n probleem meer.
Op mijn machine kan ik als ConnectionString naar MSDE het volgende gebruiken: "Data Source=.; Initial Catalog=master; Integrated security=SSPI". Als ik deze waarde toeken aan de ConnectionString property van de SqlConnection component, kan ik die vervolgens proberen te openen met Open. Als dat lukt, dan is het moeilijkste al achter de rug. Anders zul je even moeten "proberen" tot je de juiste ConnectionString te pakken hebt. Je kan de ConnectionString property een waarde geven in de Object Inspector, maar het ook in Delphi source code doen, zoals ik hieronder in de Click event handler van een Button doe:

```
procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
begin
  try
    SqlConnection1.ConnectionString :=
     'Data Source=.; Initial Catalog=master; Integrated security=SSPI';
    try
      SqlConnection1.Open // open connection
    finally
      SqlConnection1.Close
    end
  except
    on E: Exception do
      MessageBox.Show(E.Message)
  end
end;
```

Let op dat je de SqlConnection ook weer afsluit met Close, anders blijft de verbinding open staan.

ADO.NET SqlCommand

Tijd om eens wat met de database te doen. Voor de lezers van deze gratis Delphi OplossingsCourant verstuur ik twee tot drie keer per jaar een mailtje met daarin de aankondiging van het nieuwe nummer, plus een kort overzicht van de inhoud. Om dat mailtje te versturen, heb ik een database met e-mail adressen - zonder bijbehorende namen of andere informatie (ik benader de DOC-abonnees per e-mail uitsluitend voor de Delphi OplossingsCourant, dus meer dan het e-mail adres heb ik niet nodig).

Als ik met ADO.NET een tabel genaamd "abonnee" wil maken, met één veld genaamd "email" van type nvarchar met lengte 42 (dat tevens het keyfield van de unieke index is), dan kan ik dat doen door een SQL "create table" statement via de SqlConnection naar de database te sturen. Een dergelijk statement kun je in de CommandText property van een SqlCommand component zetten, en dan via de ExecuteNonQuery uitvoeren, zoals hieronder te zien is:

```
procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
var
  Command: SqlCommand;
begin
  try
    SqlConnection1.ConnectionString :=
      'Data Source=.; Initial Catalog=master; Integrated security=SSPI';
    try
      SqlConnection1.Open // open connection
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end;
    Command := SqlCommand.Create;
    Command.Connection := SqlConnection1;
    Command.CommandText := 'create table abonnee (email nvarchar(42))';
    try
      Command.ExecuteNonQuery; // create table
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end;
    Command.CommandText := 'create unique index lijst on abonnee (email)';
    try
      Command.ExecuteNonQuery; // create index
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end;
    Command.CommandText := 'insert into abonnee values(''b.swart@chello.nl'')';
    try
      Command.ExecuteNonQuery; // insert record
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end
  finally
    SqlConnection1.Close
  end
end;
```

Deze methode ExecuteNonQuery verwacht geen resultaat terug van de database, dus is bij uitstek geschikt voor SQL create table, drop table, insert, update en delete statements.

ADO.NET SqlDataReader

Als er een tabel is die al gevuld is met records, dan kunnen we daar een select statement op loslaten. Weer met behulp van een SqlCommand, maar deze keer krijgen we wel een resultaat terug, en moeten we geen ExecuteNonQuery maar de methode ExecuteReader aanroepen. Die geeft als resultaat en SqlDataReader terug, waarin het resultaat van de SQL query in staat. Maar dan wel als read-only, uni-directional dataset, dus je mag niks wijzigen en er alleen van boven naar beneden doorheen lopen. Da's goed genoeg voor mij, want ik wil toch alleen maar door de lijst met abonnees heen lopen om iedereen een mailtje te sturen dat het nieuwe nummer van de Delphi OplossingsCourant klaar staat.

De SqlDataReader heeft een methode Read die True teruggeeft als het gelukt is (en dan meteen het volgende record in de buffer heeft gezet), of False als het einde van de dataset is bereikt. Erg handig in een while-loop dus, om daarbinnen dan de records te verwerken. De SqlDataReader heeft daar speciale Get- methoden voor, waarbij je eigenlijk van te voren al moet weten wat het type is van het veld waarvan je de waarde wilt ophalen. Dus als het eerste veld van type string is, dan roep je GetString aan. En zo zijn er ook GetInt32, en nog vele andere methoden (gelukkig heeft Delphi 8 for .NET ook nog steeds Code Insight die je alle mogelijkheden kan laten zien). Als argument aan de GetString moet je dan het volgnummer van het betreffende veld meegeven, waarbij de teller bij 0 begint. In mijn geval wordt dat dus GetString(0), zoals hieronder te zien is in een klein stukje code dat door de abonnee tabel heen loopt en ieder e-mail adres ophaalt (en ze in dit voorbeeld allemaal achterelkaar zet in een lange string, zonder er veel bijzonders mee te doen):

```
procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
var
  Command: SqlCommand;
  DataReader: SqlDataReader;
  Str: StringBuilder; // System.Text
begin
  try
    SqlConnection1.ConnectionString :=
      'Data Source=.; Initial Catalog=master; Integrated security=SSPI';
    try
      SqlConnection1.Open; // open connection
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end;
    Command := SqlCommand.Create;
    Command.Connection := SqlConnection1;
    Command.CommandText := 'select * from abonnee';
    try
      DataReader := Command.ExecuteReader;
      Str := StringBuilder.Create('DOC abonnees:');
      while DataReader.Read do
      begin
        Str.Append(#13#10 + DataReader.GetString(0)) // 1e veld, type string
      end;
      MessageBox.Show(Str.ToString)
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end;
  finally
    SqlConnection1.Close
  end
end;
```

Het resultaat is als volgt:

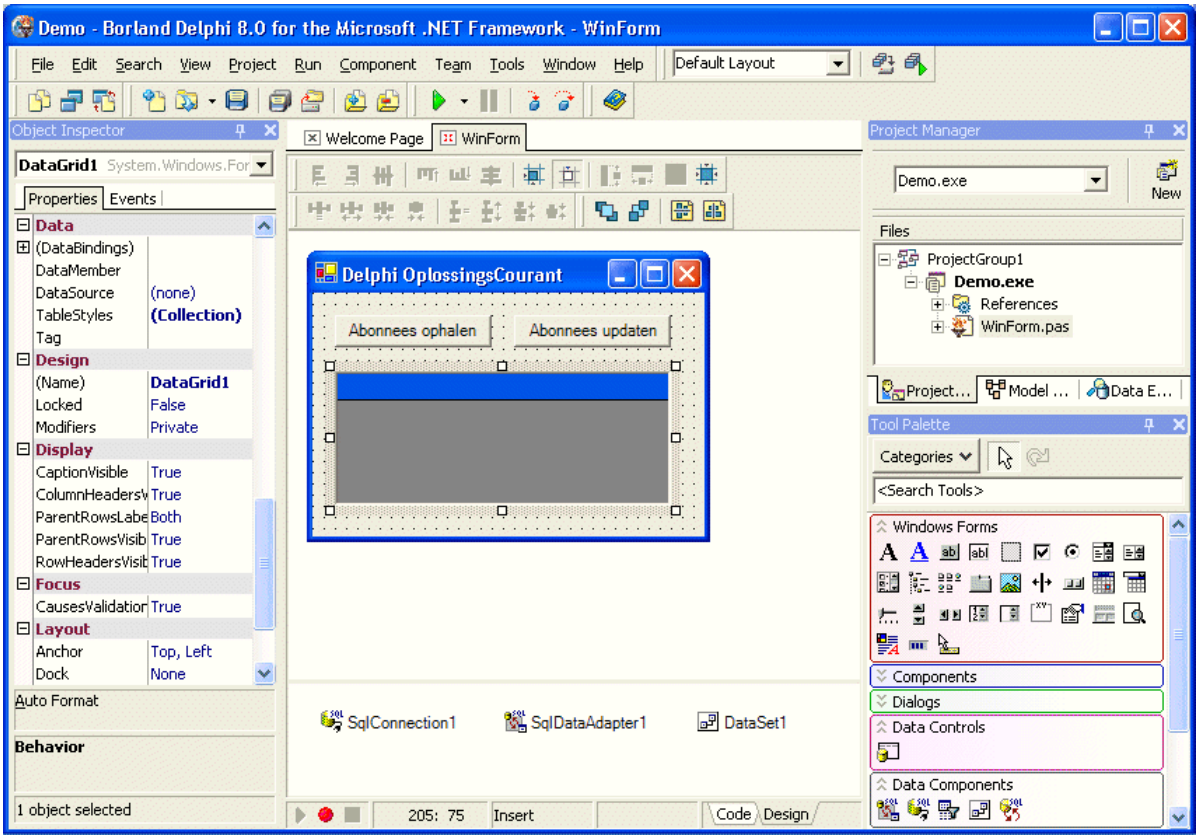
Het is nu een kleine moeite om in de while-loop iets anders te doen dan het alleen maar "optellen" van de waarde van het email field in een lange string (iets waar je in .NET beter geen String voor kan gebruiken, maar een StringBuilder uit de System.Text namespace, zoals ik hierboven heb gedaan);

ADO.NET Disconnected

Hoe kan je de inhoud van de abonnee tabel aanpassen? Dat kan op verschillende manieren. Je kan een SQL update commando in de CommandText property van een SqlCommand component stoppen, en dan weer met ExecuteNonQuery dit SQL commando naar de database sturen (vergelijkbaar met het SQL insert commando van twee listings geleden). Echter, meestal wil je de updates (automatisch) laten uitvoeren als resultaat van een bewerking door een eindgebruiker, bijvoorbeeld in een DataGrid. Hierbij gebruiken we een .NET DataSet, en betreft dan de disconnected benadering van ADO.NET.
Ga naar de Data Controls category van de Tool Palette, en zet een DataGrid control op de form. In dit DataGrid gaan we de lijst van abonnees neerzetten, zodanig dat ik er ook nieuwe e-mail adressen aan kan toevoegen, en updates kan maken en kan opsturen. Voor dat laatste heb ik wel een tweede button nodig. Ik heb er dan eentje om de data op te halen, en eentje om de updates terug te sturen.

ADO.NET SqlDataAdapter

Tot slot hebben we een SqlDataAdapter nodig, alsmede de .NET DataSet. Dit zijn allebei non-visuele componenten, die dus onderaan de Designer komen te staan. Het WinForm ziet er nu als volgt uit:

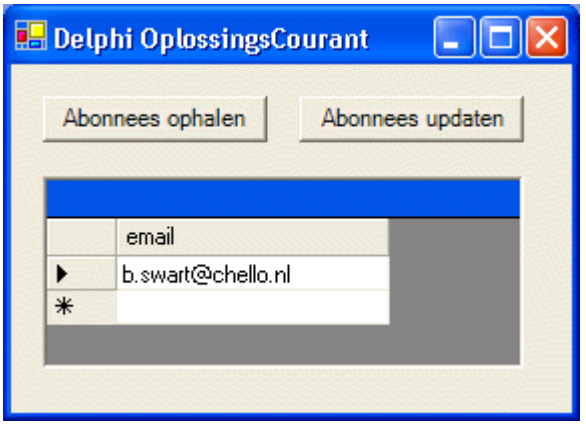


In de eerste button moet ik de code schrijven om een select statement uit te voeren, maar deze keer niet om het resultaat niet in een read-only unidirectional SqlDataReader, maar om het resultaat in een .NET DataSet te stoppen, dat we vervolgens gebruiken om het DataGrid naar te laten wijzen.
Voor het eerste deel hebben we de SqlDataAdapter nodig, en van dit component gebruiken we de ingebouwde SqlCommand property om daarvan de SQL query in de CommandText te zetten. Waar een "losse" SqlCommand via ofwel ExecuteNonQuery ofwel ExecureReader de query kan uitvoeren, daar kan een SqlDataAdapter de query uitvoeren en het resultaat automatisch in een .NET DataSet stoppen met behulp van de Fill methode. Fill krijgt als eerste argument de .NET DataSet mee, en als tweede argument de naam die we aan de resultaatset willen geven. In tegenstelling namelijk tot de dataset componenten die we in Delphi gewend zijn, kan een .NET DataSet meerdere tabellen (of resultaten van queries) bevatten, die allemaal DataTables zijn. In de aanroep die ik gebruik, geef ik de nieuwe DataTable de naam 'Abonnees'.

Nadat we een DataTable in de .NET DataSet hebben gevuld, kunnen we de inhoud hiervan afbeelden in een DataGrid. Daarvoor moeten we eerst de DataSource property van de DataGrid laten wijzen naar de .NET DataSet, en daarna aangeven welke van de potentieel vele DataTables in de .NET DataSet we willen laten zien. Dat laatste kunnen we aangeven met de DataMember property, die ik de waarde 'Abonnees' geef (dezelfde waarde die ik bij de Fill methode meegaf dus).

```
procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
begin
  try
    SqlConnection1.ConnectionString :=
      'Data Source=.; Initial Catalog=master; Integrated security=SSPI';
    try
      SqlConnection1.Open // open connection
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end;
    SqlDataAdapter1.SelectCommand.Connection := SqlConnection1;
    SqlDataAdapter1.SelectCommand.CommandText := 'select * from abonnees';
    try
      SqlDataAdapter1.Fill(DataSet1, 'Abonnees');
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end;
    dataGrid1.DataSource := DataSet1;
    dataGrid1.DataMember := 'Abonnees'
  finally
    SqlConnection1.Close
  end
end;
```

Als de deze toepassing compileren en runnen, dan kunnen we op de eerste button drukken om het DataGrid te vullen met de gegevens uit de abonnees tabel:



In het DataGrid kunnen we wijzigingen maken, bijvoorbeeld mijn e-mail adres wijzigen van b.swart@chello.nl naar drbob@chello.nl (die komen allebei aan), of een nieuw e-mail adres toevoegen aan de lijst met abonnees, als iemand weer een mailtje heeft gestuurd naar doc@eBob42.com.
In beide gevallen zal ik de inhoud van het DataGrid - die verbonden is met de .NET DataSet - naar de onderliggende database moeten sturen, en daarvoor kan ik de Update methode van de SqlDataAdapter aanroepen. Die gaat er wel vanuit dat er dan ook iets in de UpdateCommand van de SqlDataAdapter staat. Omdat we dat niet met de hand hebben gedaan - of willen doen - kunnen we een speciale SqlCommandBuilder gebruiken om de SQL update, delete en insert statements automatisch te laten

genereren. Het was daarvoor wel noodzakelijk dat de tabel een unieke index heeft - en daarom hebben we destijds de "create unique index" aangeroepen bij het aanmaken van de tabel zelf. Zonder een unieke index is de SqlCommandBuilder niet in staat om de juiste where-clause bij de update en delete commando's te genereren.
Met deze commando's erbij, kan ik nu de Update method van de SqlDataAdapter aanroepen, met daarbij als argumenten de .NET DataSet alsmede de naam van de DataTable (die met het DataGrid verbonden is), zodat de juiste Update, Delete en Insert commando's verstuurd kunnen worden. Mocht er iets mis gaan, dan zorgt de code in de except-clause van de try-except ervoor dat we de foutmelding in beeld krijgen.

```
procedure TWinForm.Button2_Click(sender: System.Object; e: System.EventArgs);
var
  CB: SqlCommandBuilder;
begin
  try
    SqlConnection1.ConnectionString :=
      'Data Source=.; Initial Catalog=master; Integrated security=SSPI';
    try
      SqlConnection1.Open; // open connection
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end
      CB := SqlCommandBuilder.Create(SqlDataAdapter1);

      SqlDataAdapter1.UpdateCommand := CB.GetUpdateCommand;
      SqlDataAdapter1.InsertCommand := CB.GetInsertCommand;
      SqlDataAdapter1.DeleteCommand := CB.GetDeleteCommand;

      SqlDataAdapter1.Update(DataSet1,'Abonnees')
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end
  finally
    SqlConnection1.Close
  end
end;
```

Hiermee kunnen we alle mogelijke aanpassingen in het DataGrid automatisch doorsturen naar de onderliggende database tabel voor de lijst met DOC abonnees.

Natuurlijk konden een heleboel properties ook al met de hand in de Object Inspector ingesteld worden, maar door het met Delphi 8 for .NET source code te laten zien hoop ik de werking van ADO.NET een beetje duidelijker te hebben gemaakt.
Voor meer informatie over Delphi 8 for .NET verwijs ik graag naar de artikelen op mijn websites te http://www.eBob42.com of http://www.drbob42.com of naar de Delphi 8 for .NET Clinic op donderdag 22 januari of 27 mei 2004, waarin zowel ADO.NET als de Borland Data Providers voor .NET uitgebreid aan de orde komen. Zie http://www.eBob42.com/training voor meer details.

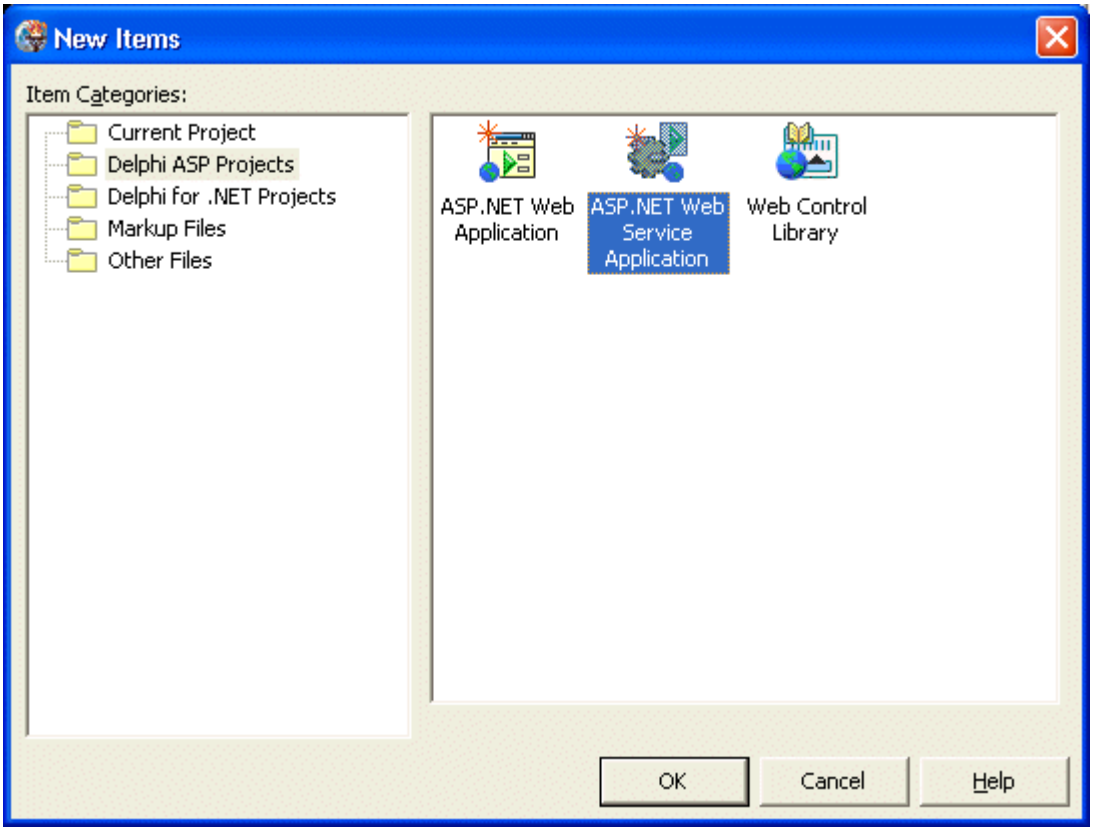
Delphi 8 ASP.NET Web Services

SOAP and Web Services are hot, and used just about everywhere these days. The Microsoft .NET Framework and SDK contain built-in support for web services (with the System.Web.Services.dll assembly as one of the core pieces).

This article will show in detail how we can build ASP.NET web services using Borland's Delphi 8 for .NET, as well as how we can import web services (in this case the same one) and use them in WinForms applications - also using Delphi 8 for .NET.

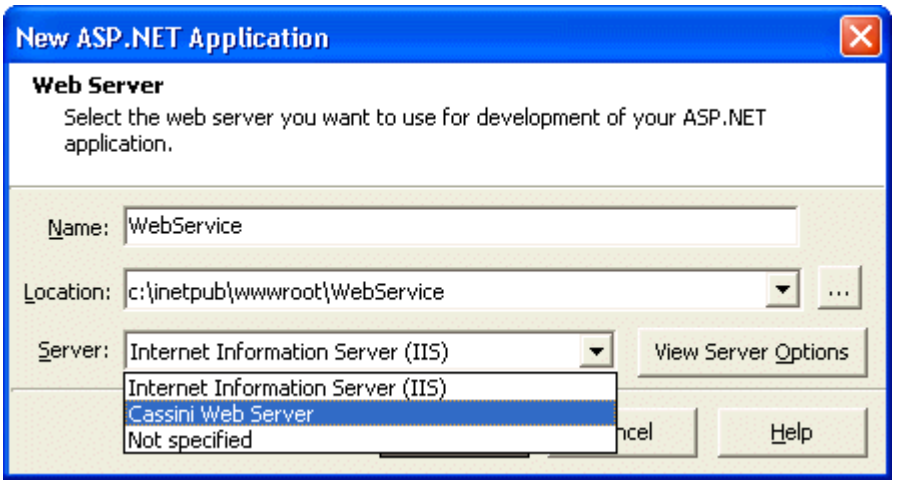
Web Service Engine

The example Web Service that we will be making and using in this article is the Centigrade to Fahrenheit Temperature Conversion Web Service, also known by the name Celsius. This Web Service is available on the web as http://www.ebob42.com/cgi-bin/Celsius42.asmx, but we'll build a fresh one using Delphi 8 for .NET now.
Start Delphi 8 for .NET, and create a new project using *File | New - Other*. This will give us the Object Repository, which has three new project targets in the Delphi ASP Projects category. In time, I'll cover all possibilities of this category, but today, we'll only create an ASP.NET Web Service Application.



Double-click on the ASP.NET Web Service Application icon which starts the NEW ASP.NET Application wizard.

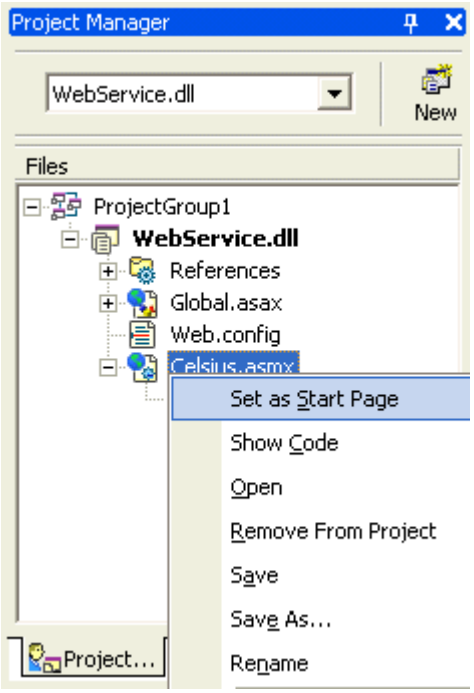
Like ASP.NET Web Forms, we can use the ASP.NET Web Service Application wizard to select a web server, with a choice between Microsoft's Internet Information Server (IIS) and the Cassini web server.



We have to specify a number of options here. First of all, the name of our project (which can contain more than one actual web service), which will also determine the namespace of the web service classes inside the project. As second option, you can specify the location of the project. This directory - by default a subdirectory of your wwwroot directory, although you can change this setting in the ASP.NET Options page of the Tools Options dialog - will actually contain all your project files including the .asmx and .asmx.pas code-behind files.

Click on OK to start the new ASP.NET web service project. It consists of a project file WebService.bdsproj, WebService.cfg and WebService.dpr, and contains the files WebService1.asmx, WebService1.asmx.pas, Global.asax, Global.pas, and Web.config. The later three file will not be covered here, since right now we only need to concern ourselves with the WebService1.asmx and .asmx.pas file, which are already opened in the code editor now. Do *File | Save As* to save then under a new name, such as Celsius. Note that both files will be renamed at once - so I end up with Celsius.asmx and Celsius.asmx.pas.

The Delphi 8 for .NET project options contain a reference to the .asmx page of the project to use as start page when debugging the application. In order to specify that the renamed Celsius.asmx has to be used as starting page, right-click on Celsius.asmx in the Project Manager and select "Set as start page" (this option is only used by the IDE, and not by the web server where you deploy the web service).



Now it's time to do *File | Save All* and make sure all files are saved on disk before we continue.

TCelsius Web Service

Apart from renaming the WebServices1 files to Celsius, I also want to rename the generated name of the web service class (by default TWebService1) to TCelsius. This is something we can try to do in two files: both in the new Celsius.asmx and in the Celsius.asmx.pas source file.
In the code editor, click on the tab for Celsius.asmx (which contains a single line) and replace WebService1 with TCelsius. The new contents should now be as follows:

```
<%@ WebService Language="c#" Debug="true" Codebehind="Celsius.pas"
    Class="Celsius.TCelsius" %>
```

Note that Language="c#" here, which just makes sure that the actual page that is used as generated ASP.NET web form will be a C# page, using our new assembly (in the file bin\WebService.dll) as host for the Celsius.TCelsius class.
However, if you save this file, then the name TWebService1 is put back instead of the TCelsius. This is a reminder that the .asmx file does not need to be edited by us, we should use the .asmx.pas file instead.
So, click on the tab for the Celsius.asmx.pas code behind source file. This file contains the web service class in the namespace WebService as specified in Celsius.asmx. By default that class is called TWebService1, and you have to rename it to TCelsius now (a global search and replace is the fastest way).
As soon as you save the .asmx.pas file, the .asmx file will be updated as well.

Namespace

Apart from the new name of the web service, there is one thing that you may want to add: the web service namespace information (otherwise a web service will use the default namespace http://www.tempuri.org which is not a good idea since your web service and types should be unique - hence the need for a unique namespace). In my case, I always use http://www.eBob42.org as namespace, so just before the line which declares the web service class, I add a WebService attribute as follows:

```
[WebService(Namespace='http://www.eBob42.org',
 Description='Dr.Bob''s Celsius Web Service written in Delphi for .NET using ASP.NET')]

TCelsius = class(System.Web.Services.WebService)
```

This is enough to make sure that the TCelsius web service uses a non-default namespace. It's still an empty web service, however. If you browse through the Celsius.asmx.pas source file, you'll notice some helpful comments at the bottom of the file that contain some sample methods that you can uncomment to use. The most important thing that they should tell you is that any method that must be exported (as web service method) must be preceded by an attribute [WebMethod] in the definition of the method (i.e. inside the TCelsius class definition).

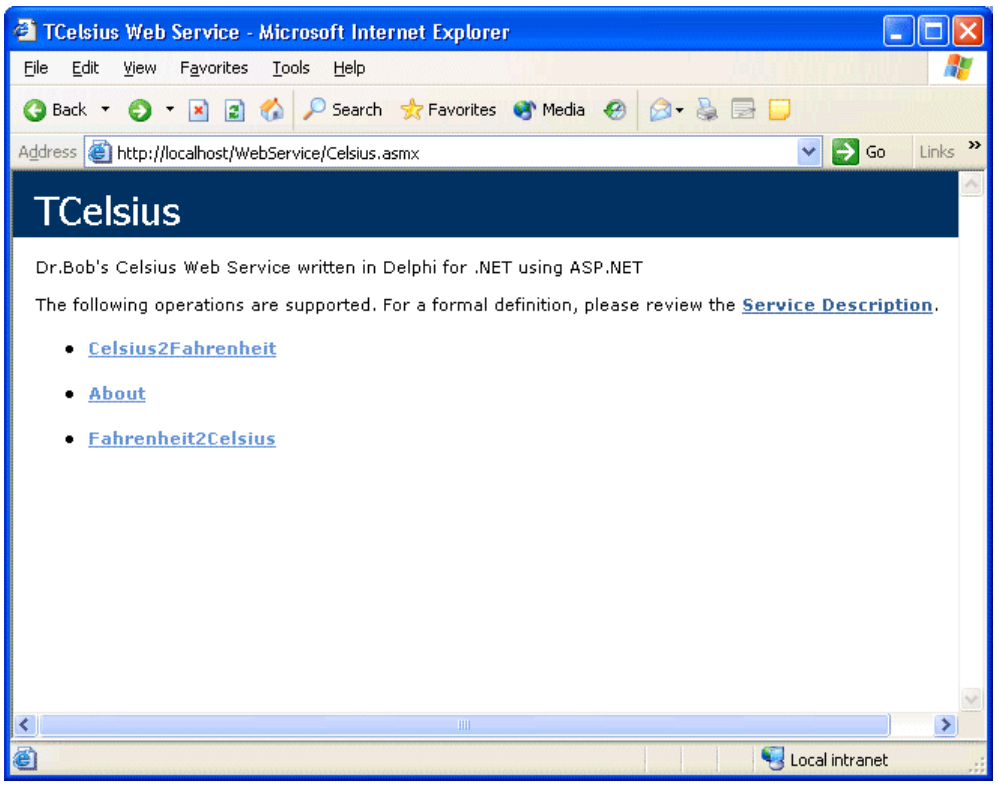
The two sample methods are HelloWorld and EchoString, and since I don't need them, just delete them and instead write three new methods: About, Celsius2Fahrenheit and Fahrenheit2Celsius. The first one has the same signature as the HelloWorld; no arguments and returning a string. The later two have both a double argument and a double result type, converting one temperature value to another.
The class definition will be as follows:

```
[WebMethod]
function About: String;

[WebMethod]
function Celsius2Fahrenheit(degrees: Double): Double;

[WebMethod]
function Fahrenheit2Celsius(degrees: Double): Double;
```

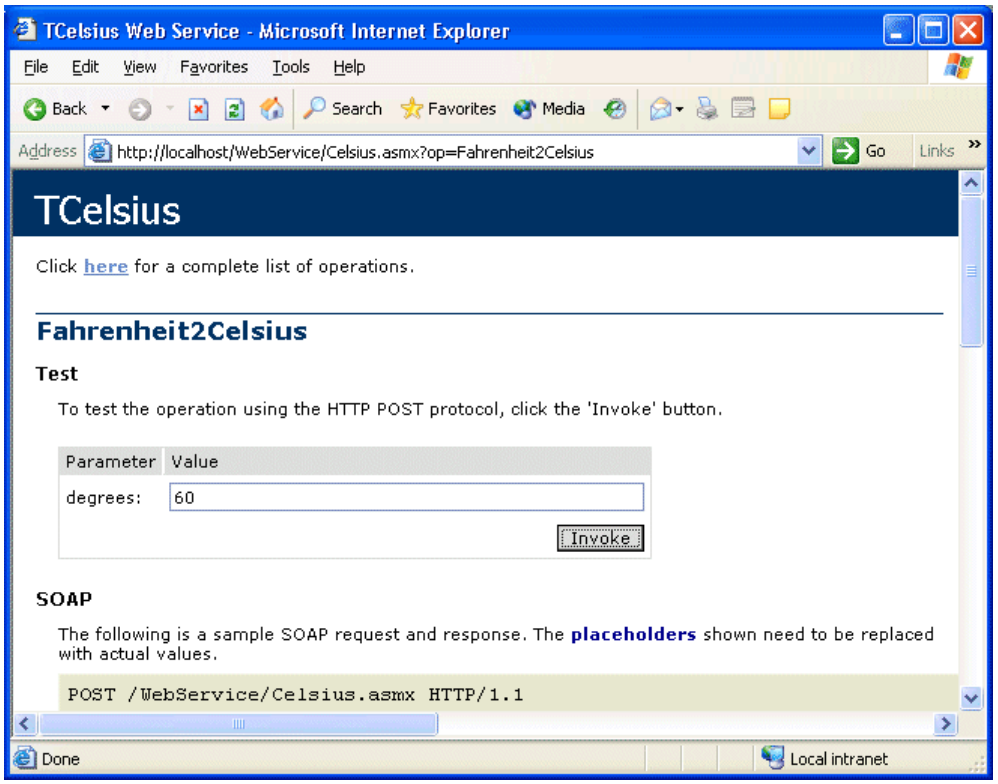
The implementatation code that you can add to the bottom of the Celsius.asmx.pas file is as follows:

```
function TCelsius.About: string;
begin
  Result := 'Dr.Bob''s Celsius Web Service written in Delphi for .NET using ASP.NET'
end;

function TCelsius.Celsius2Fahrenheit(degrees: Double): Double;
const
  AbsoluteZeroCelsius = -273.15;
begin
  if (Degrees < AbsoluteZeroCelsius) then
    raise ApplicationException.Create('Invalid Temperature');
  Result := 32 + (9 * degrees / 5)
end;

function TCelsius.Fahrenheit2Celsius(degrees: Double): Double;
const
  AbsoluteZeroFahrenheit = -459.67;
begin
  if (Degrees < AbsoluteZeroFahrenheit) then
    raise ApplicationException.Create('Invalid Temperature');
  Result := 5 * (degrees - 32) / 9
end;
```

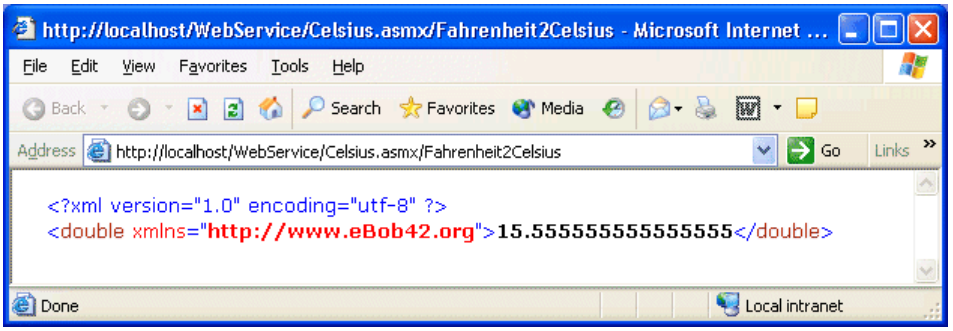
Time to save your work and compile the project. Note that you can run it from the IDE, which means that a browser will be started and the project options will be used to determine the starting page (which we set to Celsius.asmx, remember?). To run the project without actually debugging it, do *Run | Run Without Debugging*. This will launch Internet Explorer and show you the information about the Celsius web service.



You can even test the methods of the web service in the browser (all facilitated by the ASP.NET environment).



With the following result:

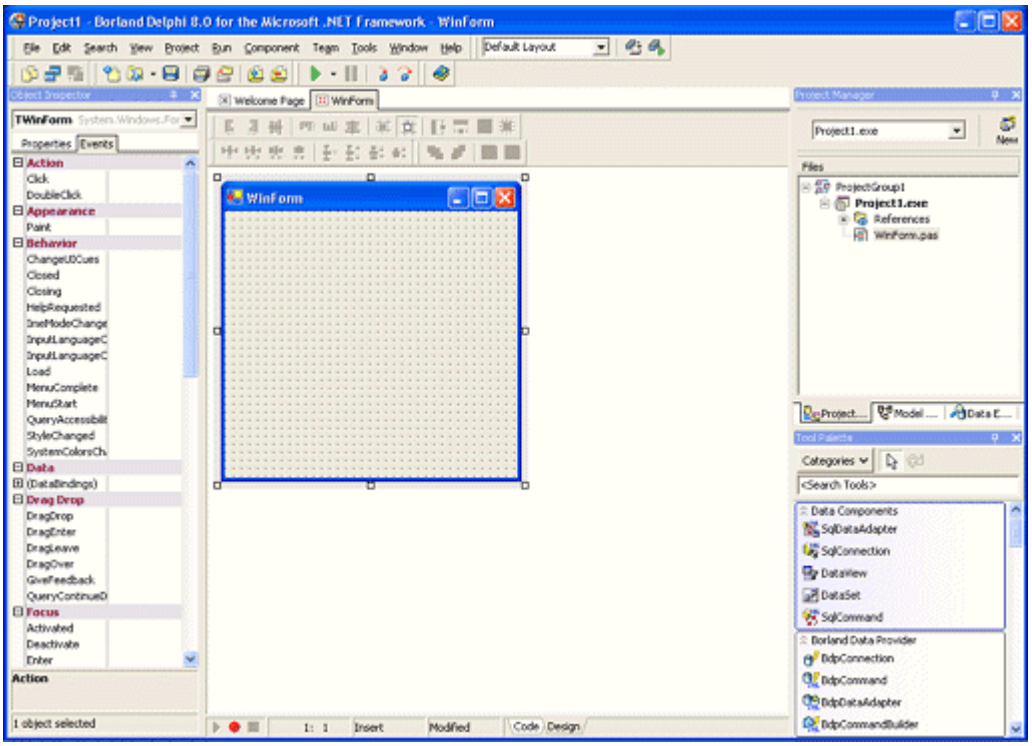


An even more useful test of the web service will be to actually import and use it - in another Delphi 8 for .NET application, of course, which is done in the second part of this article.
So close the browser, close the project (save everything if you haven't done that before), and prepare to start another Delphi 8 for .NET project.

Web Services Client

In this second part of this article, I want to show you how Delphi 8 for .NET can import and use (also called consume) web services. You can use the example web service that was made in the first part of this article, or use another web service available on the web. For your convenience (if you didn't build the web service from the first part), I have made some web services available on the internet, including the Celsius conversion web service itself, so you can still play along if you want.
Anyway, start Delphi 8 for .NET and build a new project by doing *File | New - Windows Form Application*.

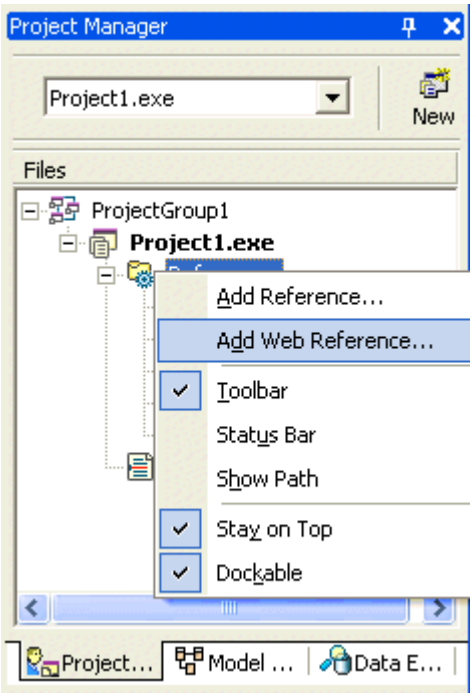
The result is a new project with an empty form that can be saved where you want, and looks as follows:



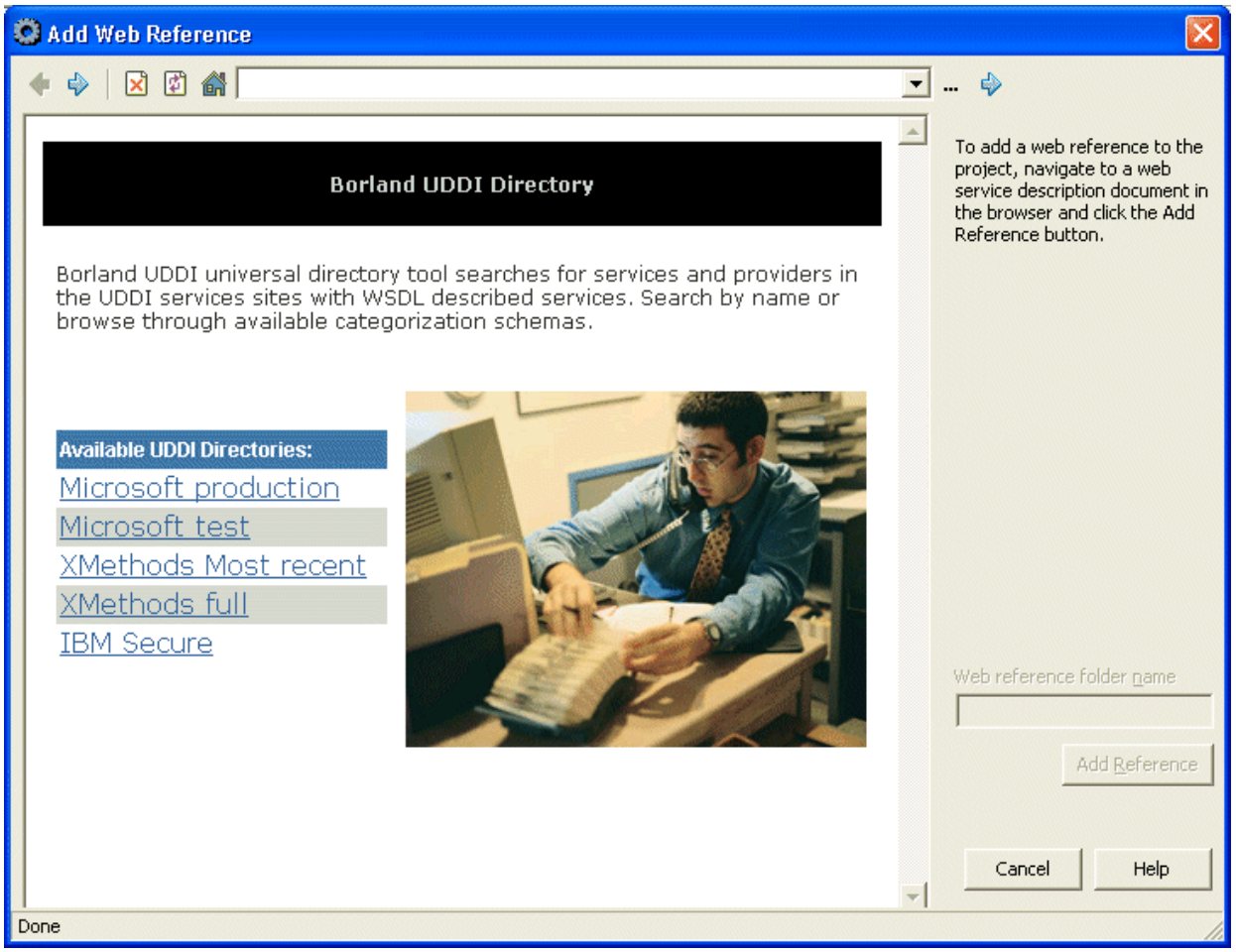
I assume you know how to place a control on a form (although this may feel a little bit different if you come from a Delphi/C++Builder background and are using Delphi 8 for .NET for the first time). Before we add visual components to the form, however, I first want to import the web service and add it to our project (so at least we know what we should build upon).

Add Web Reference

In order to import a web server, we have to add a web reference (that's how it's called in Delphi 8 for .NET), using the *Project | Add Web Reference* dialog.



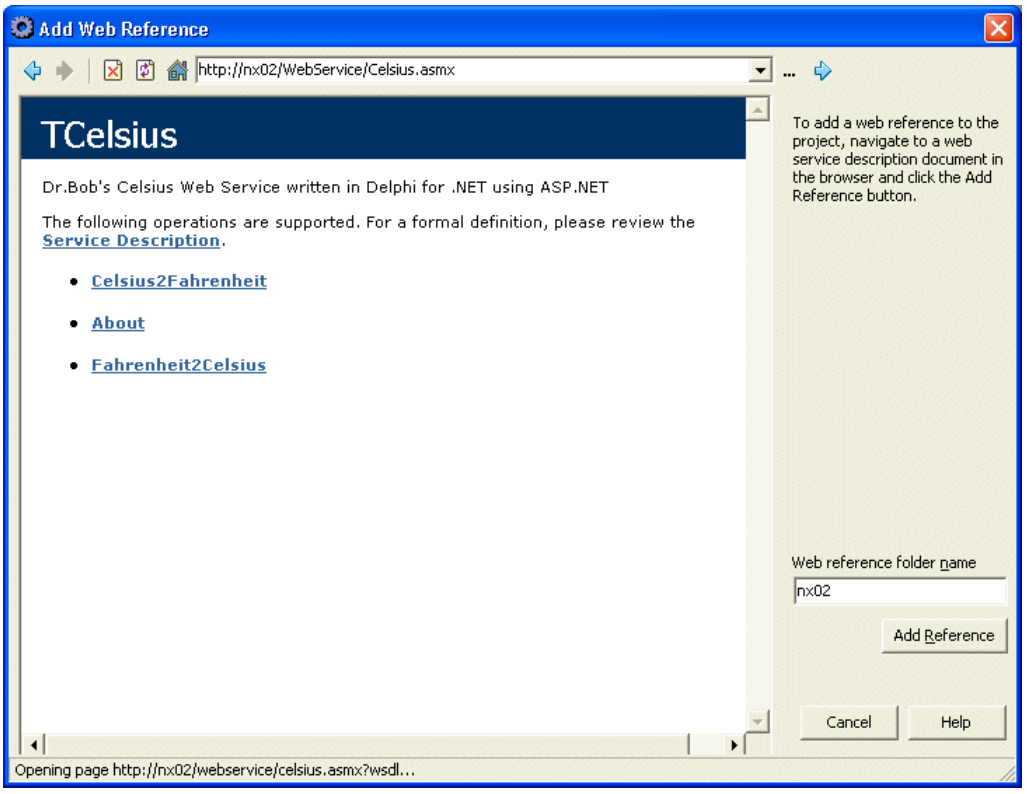
This dialog consists of a URL editbox and browser window, and starts with the Borland Delphi 8 for .NET UDDI (Universal Description Discovery and Integraion) Browser to UDDI Services - in case you want to import a web service from one of the available UDDI directories.



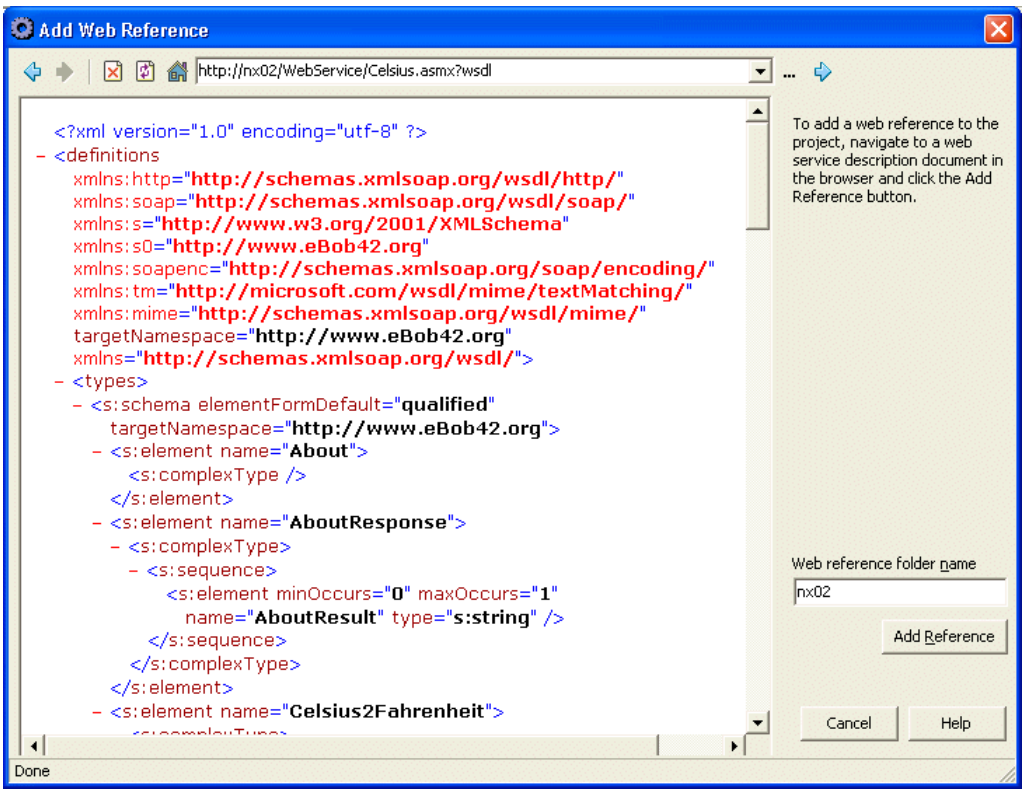
Note that the list of "known" UDDI Directories is stored inside the HTML file for this page, which is file BorlandUDDI.html in the Objrepos directory (where you can also find and change the searcher2.jpg image if you want). You can edit the BorlandUDDI.html file and add other UDDI Directories (for example the ones in your local intranet or extranet) or remove the UDDI Directories that you do not want to work with when importing Web Services with Delphi 8 for .NET.

Although some of my web services are registered with XMethods, I want to use the Celsius web service that was written in the first part of this article (and for which I know the URL, but which isn't registered with a UDDI registry, yet).

In case you have the Celsius web service available on your local machine, the URL to enter in the editbox is probably similar to http://localhost/eBob42/Celsius.asmx, otherwise you can use the version that I've deployed on the internet as http://www.eBob42.com/cgi-bin/Celsius42.asmx (hosted by TDMWeb).
In both cases, the dialog should now show the information of the Celsius web service, as can be seen in the next screenshot:

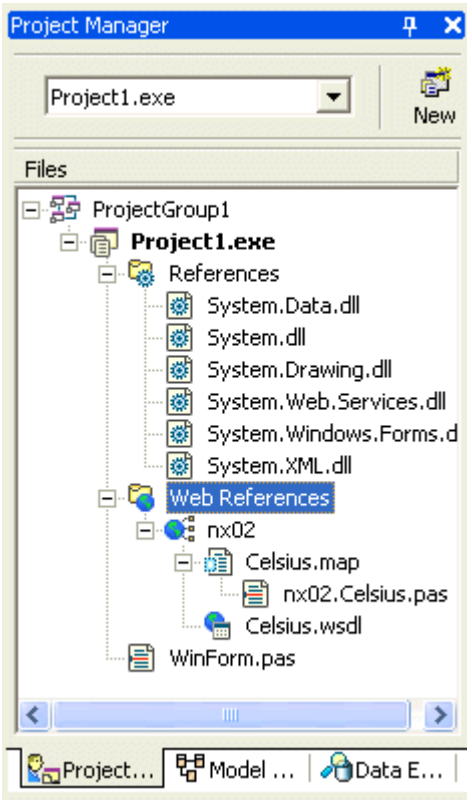


The Add Reference button is still disabled, and that's because this button can only use a formal WSDL (Web Service Description Language) definition of a web service. For that, click on the Service Description link to get an enabled Add Reference button.



Now we can click on the Add Reference button, which will take a few seconds to analyze the WSDL and produce additional files that are added to the project.

Since I've imported the Celsius web service from my nx02 example, we get a new node nx02, with subnodes for the Celsius.wsdl file and a Celsius.map file that refers to the imported web service itself. The import code is placed in nx02.Celsius.pas, where the class Celsius is defined in the nx02.Celsius namespace.



The imported source code is interesting to view, but the only interesting part is the class TCelsius with a default constructor, and the methods About, Celsius2Fahrenheit and Fahrenheit2Celsius that we can use very easily.

Client Form

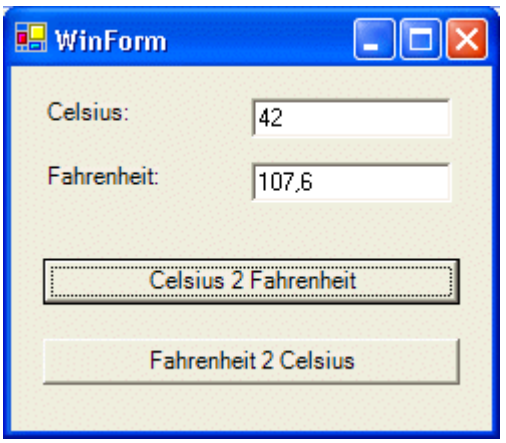
Time to start some visual programming on the WinForm. Drop two Label, two TextBox, and two Button components on the form. Call the first TextBox tbCelsius and the second one tbFahrenheit, and call the first Button btnCelsius2Fahrenheit and the second one btnFahrenheit2Celsius. Now, put "Celsius: " in the Text property of the first Label, and "Fahrenheit: " in the Text property of the second Label. Similarly, put "Celsius 2 Fahrenheit" in the Text property of the first Button, and "Fahrenheit 2 Celsius" in the Text property of the second Button. If you want everything to appear clean, you can also clear the Text property of the two TextBoxes.

Time to write some event handling code for the two Buttons. First, add both the SysUtils and nx02.Celsius units to the uses clause. Now we can implement the two event handlers. In both cases, we need to create a new instance of the TCelsius class, and then call the required methods.

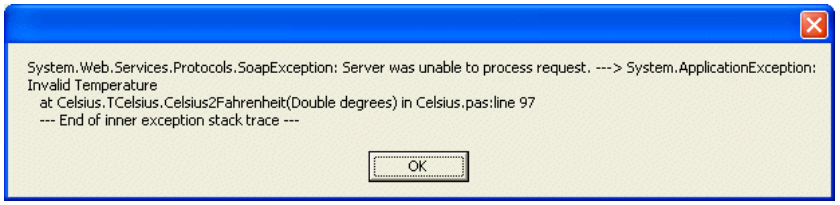
```
procedure TWinForm.btnFahrenheit2Celsius_Click(sender: System.Object;
  e: System.EventArgs);
var
  C2F: nx02.Celsius.TCelsius;
begin
  C2F := nx02.Celsius.TCelsius.Create;
  tbCelsius.Text := FloatToStr(
    C2F.Fahrenheit2Celsius(StrToFloatDef(tbFahrenheit.Text,0)))
end;
```

```
procedure TWinForm.btnCelsius2Fahrenheit_Click(sender: System.Object;
  e: System.EventArgs);
var
  C2F: TCelsius;
begin
  C2F := TCelsius.Create;
  tbFahrenheit.Text := FloatToStr(
    C2F.Celsius2Fahrenheit(StrToFloatDef(tbCelsius.Text,0)))
end;
```

Note that you should probably want to place all code inside a try-catch block, since creating an instance of the web service can fail (if you cannot connect to it for some reason), but even if you have created an instance, the call to Celsius2Fahrenheit or Fahrenheit2Celsius can also fail or throw an exception - for example when trying to convert a temperature which is below the absolute zero. You can experiment and discover this in practice for yourself.
Now, do *File | Save All* to make sure everything is saved, compile and run the project which will show the Celsius Client. As quick demo, I've entered 42 in the tbCelsius TextBox and clicked on the bntCelsius2Fahrenheit button:



When entering a value of -400 in the TextBox for Celsius, and then trying to convert from Celsius to Fahrenheit will result in an exception being thrown by the web service, as shown below:



A similar error dialog (with a longer message) will be shown if the web service isn't available, and hence the call to create an instance of the Celsius.localhost.Celsius web service constructor will fail. Using try-except will make sure that the client application can intercept the exception and optionally present some more useful information to the endusers. This can be done with a code construct as follows:

```
procedure TWinForm.btnFahrenheit2Celsius_Click(sender: System.Object;
  e: System.EventArgs);
var
  C2F: nx02.Celsius.TCelsius;
begin
  try
    C2F := nx02.Celsius.TCelsius.Create;
    tbCelsius.Text := FloatToStr(
      C2F.Fahrenheit2Celsius(StrToFloatDef(tbFahrenheit.Text,0)))
```

```
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end
  end;

  procedure TWinForm.btnCelsius2Fahrenheit_Click(sender: System.Object;
    e: System.EventArgs);
  var
    C2F: TCelsius;
  begin
    try
      C2F := TCelsius.Create;
      tbFahrenheit.Text := FloatToStr(C2F.Celsius2Fahrenheit(StrToFloatDef(tbCelsius.Text,0)))
    except
      on E: Exception do
        MessageBox.Show(E.Message)
    end
  end;
```

Obviously, it would be a good idea to give a more user-friendly error message in the except-clause, but I leave that as exercise for the reader.

Summary

In this article I've used Delphi 8 for .NET - Borland's new IDE for Microsoft's .NET Framework - to build, implement, deploy and consume an ASP.NET web service. We've seen numerous IDE features that support web service developers and consumers while working with ASP.NET web services in Delphi 8 for .NET. For more information about ASP.NET and Web Services with Delphi, check out my special Delphi 8 for .NET Clinic training day at February 19th or June 24th, 2004 about ASP.NET Web Forms, Web Services and .NET Remoting.