# Delphi OplossingsCourant

**Vol 4. No. 1.** Een gratis kwartaalpublicatie van **Bob Swart Training & Consultancy** (**eBob42**) - http://www.eBob42.com

Helmond, 1 januari 2002,

Om te beginnen wens ik iedereen natuurlijk een gelukkig nieuwjaar. Ik hoop van harte dat 2002 een beter jaar wordt dan 2001 - in alle opzichten.

Niet dat 2001 ons op technisch punt weinig heeft nieuws gebracht (ik denk aan Kylix, Delphi 6, Kylix 2, JBuilder 6, maar ook Windows XP en de betas van .NET), maar op andere gebieden is 2001 een jaar dat ik het liefst snel achter me wil laten.

Het jaar 2002 begin ik met de oprichting van de eenmanszaak **Bob Swart Training & Consultancy** dat zich, zoals de naam al zegt, zal richten op Training en Consultancy met en over Borland tools zoals Delphi en Kylix (maar aandacht voor C# en .NET zal daarnaast onvermijdelijk zijn).

Een groot deel van mijn tijd zal nog gevuld zijn met het schrijven van artikelen, boeken en mijn eigen cursusmateriaal. Ik ben zelfs van plan om in 2002 voor het eerst een geheel boek van (alleen) eigen hand te produceren. Als onderwerp zal uiteraard Internet Development centraal staan. Meer informatie volgt in latere nummers van de **D**elphi **O**plossings**C**ourant (DOC), die ik eveneens voortzet vanuit de eenmanszaak.

In dit nummer van de Delphi OplossingsCourant staan twee nieuwe artikelen. In het eerste stuk beschrijf ik Container Classes in Delphi - uiterst handige hulpmiddelen voor het beheren van objecten in alle soorten en maten. Het tweede stuk gaat over WebSnap en de Adapter componenten die daar een belangrijke rol in spelen. Met name dit laatste onderwerp komt terug in de WebSnap Delphi 6 Clinics die vanaf begin 2002 worden verzorgd in Eindhoven.

Wie interesse heeft in een gratis abonnement op de Delphi OplossingsCourant kan een mailtje sturen naar doc@eBob42.com. Je krijgt dan ieder kwartaal een mailtje met de inhoudsopgave zodra het nieuwe nummer beschikbaar is.

**Inhoudsopgave**



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

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

## Delphi 6 Clinics in 2002

Ook in 2002 zal er weer eens in de drie weken een Delphi 6 Clinic plaatsvinden in Eindhoven. Deze keer niet meer op de vrijdag maar op **donderdag**. De data en onderwerpen zijn als volgt:

10 jan - **dbExpress & DataSnap**
31 jan - **WebBroker/InetExpress & WebSnap**
21 feb - **Advanced WebSnap & Adapters**
14 mrt - **BizSnap: XML/SOAP & Web Services**
4 april - **WebSnap & BizSnap** (in praktijk)

Wie zich inschrijft voor twee of meer dagen krijgt bovendien een gesigneerd exemplaar van het boek Delphi 6 Developer's Guide (of Kylix Developer's Guide) kado. Zie http://www.eBob42.com/training

## Conferenties 2002

Na de Delphi 6 Clinics die van januari tot april 2002 lopen, zal het conferentie seizoen in mei 2002 weer losbarsten, met de volgende evenementen:

13-14 mei - **CttM 2002**, Veldhoven
18-22 mei - **BorCon 2002**, Anaheim (USA)
9-11 juni - **DCon 2002**, Reading (UK)

De call-for-papers is inmiddels geweest, maar op dit moment weet ik alleen zeker dat ik zal spreken tijdens DCon 2002 in de UK en de BorCon 2002 in Anaheim. Tijdens BorCon zal ik een tweetal presentaties verzorgen (allebei tweemaal) over *Introduction to dbExpress and ClientDataSets* en *Multitier Application Development using DataSnap.*

# Container Classes in Delphi

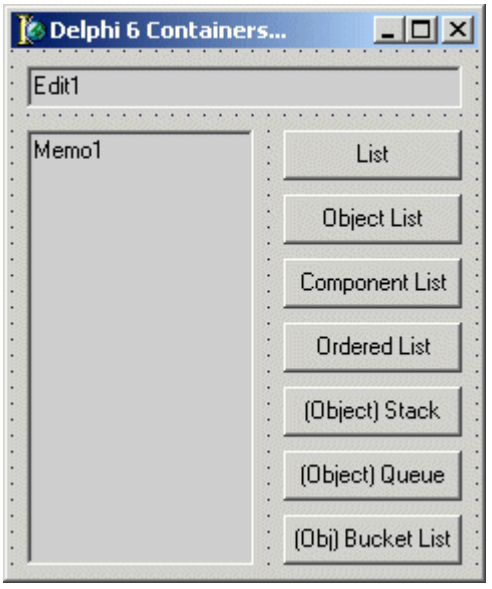
*Een container kan gezien worden als een plaats waar je een aantal elementen op een bruikbare manier in kan opslaan. Niet zomaar een verzameling van elementen dus, maar net iets meer. Voorbeelden die ik wil laten zien zijn de ComponentList, de OrderedList en - het meest specifiek - de Stack of Queue.*

## TList

Maar voor we beginnen met de specifieke container classes in Delphi, wil ik eerst nog eens kijken naar de "moeder van alle container en collection classes", namelijk de TList. Deze oude rot bestaat al sinds de eerste versie van Delphi, en biedt de mogelijkheid om een pointer in een lijst op te slaan. Met name handig als je niet van te voren weet hoeveel pointers je wilt opslaan, zodat een (fixed) array niet voldoet (in die tijd hadden we nog geen dynamic arrays). Omdat de lengte van de TList kan groeien kun je altijd nieuwe elementen aan de lijst toevoegen (niet oneindig uiteraard, maar je snapt wat ik bedoel hoop ik). De TList houdt bovendien alleen de pointers bij, en niet het geheugen van de elementen zelf; als je de TList weggooit worden alleen de pointers vrijgegeven, en niet de elementen zelf.

Het grote nadeel van de TList komt voort uit het feit dat er alleen maar pointers worden opgeslagen, en geen objecten. Er is nu geen enkele manier waarop we weten wat er in de TList is opgeslagen (tenzij je dat van te voren weet door bijvoorbeeld alleen maar TMyObjects erin te stoppen). Dit is niet zo'n groot probleem als je alleen maar elementen van hetzelfde type erin stopt, maar op het moment dat je verschillende soorten elementen wilt opslaan en gebruiken heb je een (potentieel) probleem. Aangezien we alleen maar een pointer terugkrijgen, en geen object, is er ook geen RTTI (RunTime Type Information) beschikbaar om te bekijken hoe we de individuele elementen kunnen gebruiken. Speciaal hiervoor is de TObjectList uitgevonden (alhoewel we voor de eerste versie van Delphi dit wiel nog zelf moesten uitvinden).

Voor ik verder ga met de TObjectList en de andere collection en container classes, wil ik eerst even de demonstratie toepassing toelichten (zie rechter kolom voor een screenshot). Ik heb een TEdit, een TMemo en zeven TButtons op een Form gezet, en achter iedere OnClick van een button hangt de code om een specifieke container te gebruiken.




*Figuur 1. Delphi 6 Containers*

Het stukje code hieronder maakt een instantie van TList aan, en voegt alle componenten van het huidige Form toe aan de lijst (in dit geval zijn dat de Edit, Memo en Buttons). Niet een echt zinvol voorbeeld, want we kunnen de items van de lijst nauwelijks gebruiken (zonder RTTI), maar het laat wel zien hoe het gebruik van de TList zelf is:

```
procedure TForm1.btnListClick(Sender: TObject);
var
  i: Integer;
  List: TList;
begin
  List := TList.Create;
  try
    List.Clear;
    for i:=0 to Pred(ComponentCount) do
      List.Add(Components[i]);
    // Usage of List...
  finally
    List.Free
  end
end;
```

De volgende stukken code zullen een vergelijkbare inhoud hebben. Vanaf de TObjectList kunnen we bovendien echt gebruikmaken van de elementen in de containers.

## TObjectList

De TObjectList - afgeleid van de TList - is eigenlijk pas het eerste type dat ik echt een container zou willen noemen. In plaats van alleen maar pointers kunnen we in de TObjectList elementen stoppen die afgeleid zijn van TObject. Het enorme grote voordeel hiervan is natuurlijk het feit dat TObject en afgeleide objecten onderzocht kunnen worden met behulp van RTTI en de IS en AS keywords.

Dit heeft tot gevolg dat we verschillende soorten objecten in een TObjectList kunnen stoppen, en toch bij ieder element kunnen vragen van welk type het is, en het ook als zodanig kunnen gebruiken. Sinds de TObjectList heb ik de TList dan ook niet meer nodig gehad.
Een ander verschil met de TList is het feit dat de TObjectList een property genaamd OwnsObjects heeft. Hiermee kan de TObjectList aangeven dat het de eigenaar (Owner) van de Objecten is die in de TObjectList zijn opgenomen. Let wel: het is maar een property, maar de default waarde is True, zodat bij default de TObjectList de eigenaar is van de Objecten in zijn lijst. Deze zullen dus worden opgeruimd op het moment dat je de TObjectList zelf opruimt. Even oppassen dus! Gelukkig kun je bij het aanmaken van een TObjectList via de constructor al aangeven dat OwnsObjects eventueel op False gezet moet worden (de constructor heeft één argument, met als default waarde True, wat wordt gebruikt om de OwnsObjects property zijn waarde te geven).
Als we kijken naar het stukje voorbeeld code voor de TList (zie vorige pagina), dan kunnen we dan uitbreiden voor de TObjectList door te kijken naar het item dat een TEdit "is", en indien gevonden de inhoud van dit component te gebruiken als Caption voor de Form zelf. Dit gaat als volgt:

```
procedure TForm1.ObjectLst(Sender: TObject);
var
  i: Integer;
  ObjectList: TObjectList;
begin
  ObjectList := TObjectList.Create(False);
  try
    ObjectList.Clear;
    for i:=0 to Pred(ComponentCount) do
      ObjectList.Add(Components[i]);
    // Usage of ObjectList...
    for i:=0 to Pred(ObjectList.Count) do
      if (ObjectList[i] IS TEdit) then
        Caption :=
         (ObjectList[i] AS TEdit).Text
  finally
    ObjectList.Free
  end
end;
```

Merk op dat ik de TObjectList aanmaak met als argument "False" voor de Create constructor. Als ik dat niet gedaan zou hebben, dan zouden alle componenten automatisch verwijderd worden bij het ObjectList.Free statement (en het Form zou vervolgens leeg zijn achtergebleven). Een leuk effect, maar niet echt wat de bedoeling van de demonstratie zou zijn geweest.

## TComponentList

Nog een stapje lager in de hierarchie komen we de TComponentList container tegen, afgeleid van de TObjectList die we net zagen. Het grote verschil tussen de TComponentList en de TObjectList is het feit dat een TComponentList zijn elementen (de componenten) daadwerkelijk kan beheren. Dat wil zeggen dat de TComponentList automatisch aktie kan ondernemen zodra een componenten (van buitenaf) wordt opgeruimd. Als dat niet zou gebeuren, zou er een pointer naar een niet meer bestaand component in de lijst kunnen blijven bestaan (en da's niet goed natuurlijk). Zodra een element uit de lijst wordt verwijderd doordat het betreffende component zelf wordt opgeruimd, zal de TComponentList automatisch zichzelf updaten door de reference naar het component op te ruimen (en de lijst weer sluitend te maken).
Dit gebeurt allemaal "achter de schermen", maar is makkelijk te demonstreren met het volgende code voorbeeld waarin ik eerst alle componenten van het Form aan de TComponentList toevoeg, om dan vervolgens de button waar we zojuist op klikte te verwijderen. Het effect (uitgevoerd door TCompoentList achter de schermen) zal zijn dat de button ook uit de TComponentList verwijderd zal worden. Dit kunnen we controleren door de namen van alle elementen uit de ComponentList in de Memo neer te zetten. De naam van de button waar we op drukte (de button met caption "Component List") zal niet in de lijst voorkomen, omdat deze button niet meer bestaat.

```
procedure TForm1.ComponentLst(Sender: TObject);
var
  i: Integer;
  ComponentList: TComponentList;
begin
  ComponentList := TComponentList.Create(False);
  try
    ComponentList.Clear;
    for i:=0 to Pred(ComponentCount) do
      ComponentList.Add(Components[i]);
    // Removal of Sender...
    Sender.Free;
    // Usage of ObjectList...
    Memo1.Lines.Clear;
    for i:=0 to Pred(ComponentList.Count) do
      if (ComponentList[i] IS TEdit) then
        Caption := (ComponentList[i] AS TEdit).Text
      else
        Memo1.Lines.Add(
        (ComponentList[i] AS TComponent).Name)
  finally
    ComponentList.Free
  end
end;
```

## TOrderedList

Soms is het handig of noodzakelijk om te werken met een lijst of container waarin de elementen op een bepaalde manier gesorteerd zijn. In dat geval is de TOrderedList de aangewezen keuze. Of liever gezegd: een afgeleide klasse van de TOrderedList, aangezien de TOrderedList zelf een abstract class is (afgeleid van TList), met de abstract methode PushItem.

Even los van de PushItem die geïmplementeerd moet worden, is het gebruik van de TOrderedList (en afgeleide componenten) vrij eenvoudig. Met de Push methode kun je een nieuw element toevoegen aan de lijst. Waar dat element geplaatst wordt is afhankelijk van de implementatie in de PushItem methode (die inderdaad intern door de Push wordt aangeroepen). Los van de plek waar een nieuw element via Push komt, kunnen we altijd naar de "top" van de lijst kijken, via Peek. En ook kunnen we dit top element van de lijst verwijderen met de Pop methode.

Het volgende voorbeeld compileert wel, maar zal niet werken omdat de PushItem methode van de TOrderedList nog abstract is (het dient dan ook meer ter illustratie van het gebruik van de Push en Pop methoden):

```
procedure TForm1.OrderedLst(Sender: TObject);
var
  i: Integer;
  OrderedList: TOrderedList;
begin
  OrderedList := TOrderedList.Create;
  try
    for i:=0 to Pred(ComponentCount) do
      if (Components[i] IS TButton) then
        OrderedList.Push(Components[i]);
    // Usage of OrderedList...
    Memo1.Lines.Clear;
    while OrderedList.Count > 0 do
      Memo1.Lines.Add(
        TButton(OrderedList.Pop).Name)
  finally
    OrderedList.Free
  end
end;
```

De implementatie van de PushItem methode bepaalt dus het gedrag van de TOrderedList. Twee eenvoudige implementaties zijn wel bekend, namelijk het toevoegen van nieuwe elementen aan de "onderkant" (met als resultaat een wachtrij of Queue), en het toevoegen van nieuwe elementen aan de "bovenkant" (met als resultaat een stapel of Stack). Lost van die twee bekende varianten kun je natuurlijk ook zelf de PushItem implementeren door bijvoorbeeld een gesorteerde lijst te bouwen.

## TStack en TObjectStack

De TStack is een container die afgeleid is van de TOrderedList met als implementatie van de PushItem methode het gedrag dat ik zojuist al noemde: nieuwe elementen worden bovenaan de lijst toegevoegd (met als resultaat een stapel). Dit staat ook wel bekend als LIFO, oftewel "Last In First Out".

De TStack werkt met normale pointers, net als de TList. Ik heb eerder al aangegeven dat ik dat persoonlijk niet zo erg handig vind, maar gelukkig bestaat er ook een TObjectStack versie die in plaats van pointers dus wel TObjects (en afgeleide objecten) met volledige RTTI opslaat. Dat is dus mijn eerste keuze als ik een stack van objecten nodig heb!

Het volgende voorbeeld gebruikt Push om alle componenten van het Form in een ObjectStack te stoppen, en zal vervolgens de elementen in deze stapel aflopen (in omgekeerde volgorde - de laatste het eerst) en hun namen in de Memo component afbeelden.

```
procedure TForm1.btnStack(Sender: TObject);
var
  i: Integer;
  ObjectStack: TObjectStack;
begin
  ObjectStack := TObjectStack.Create;
  try
    for i:=0 to Pred(ComponentCount) do
        ObjectStack.Push(Components[i]);
    // Usage of ObjectStack...
    Memo1.Lines.Clear;
    while ObjectStack.Count > 0 do
      Memo1.Lines.Add(
      (ObjectStack.Pop AS TComponent).Name)
  finally
    ObjectStack.Free
  end
end;
```

Merk op dat ik alleen maar de Name property van ieder element in de ObjectStack nodig heb. En omdat die al gedefinieerd is op TObject niveau, hoef ik niet specifiek te weten of een element een TEdit, een TMemo of een TButton component is: het feit dat het een TComponent is, is voldoende om bij de Name property te komen. Scheelt weer een aanzienlijke hoeveelheid code, zeker als er zich meer dan alleen maar TEdits, TMemos en TButtons op het Form bevinden.

Als laatste wil ik nog opmerken dat de TStack en TObjectStack de elementen die we "erin" stoppen niet beheren, en dus ook niet automatisch zullen vrijgeven als we de Stack weggooien.

## TQueue en TObjectQueue

Waar een Stack (en ObjectStack) de elementen in een LIFO volgorde opslaan, daar zal een Queue uiteraard de FIFO volgorde gebruiken: elementen worden onderaan de lijst toegevoegd en van bovenaan verwijderd. Handig voor bijvoorbeeld de implementatie van wachtrijen. En waar de TQueue weer alleen pointers opslaat, zal de TObjectQueue weer objecten gebruiken.

```
procedure TForm1.btnQueue(Sender: TObject);
var
  i: Integer;
  ObjectQueue: TObjectQueue;
begin
  ObjectQueue := TObjectQueue.Create;
  try
    for i:=0 to Pred(ComponentCount) do
        ObjectQueue.Push(Components[i]);
    // Usage of ObjectQueue...
    Memo1.Lines.Clear;
    while ObjectQueue.Count > 0 do
      Memo1.Lines.Add(
      (ObjectQueue.Pop AS TComponent).Name)
  finally
    ObjectQueue.Free
  end
end;
```

## Custom Ordering

De Stack en Queue kunnen dus gebruikt worden om elementen aan de ene of de andere kant van de lijst toe te voegen. Maar wat als we de elementen op basis van bijvoorbeeld hun naam (of een andere eigenschap) willen sorteren? Dan moeten we zelf een nieuwe TSortList (en TObjSortList) van de TOrderedList afleiden en implementeren. De belangrijke methode is hierbij PushItem, die voor ieder nieuw element eerst de juiste plaats in de container moet zoeken alvorens het element daadwerkelijk (op die plek) toe te voegen. Voor de TObjSortList moeten we ook nog de Push, Pop en Peek methoden overriden en zorgen dat die met TObjects in plaats van pointers werken.

```
type
  TObjSortList = class(TOrderedList)
  protected
    procedure PushItem(AItem: Pointer); override;
  public
    function Push(AObject: TObject): TObject;
    function Pop: TObject;
    function Peek: TObject;
  end;

function TObjSortList.Peek: TObject;
begin
  Result := TObject(inherited Peek)
end;

function TObjSortList.Pop: TObject;
begin
  Result := TObject(inherited Pop)
end;

function TObjSortList.Push(AObject: TObject):
  TObject;
begin
  Result := TObject(inherited Push(AObject))
end;

procedure TObjSortList.PushItem(AItem: Pointer);
var
  i: Integer;
  NewName: String;
begin
  i := 0;
  if Count = 0 then List.Add(AItem)
  else
  begin
    NewName := TComponent(AItem).Name;
    while (i < Count) and
      (CompareText(TComponent(List[i]).Name,
       NewName) > 0) do Inc(i);
    if i = Count then
      List.Add(AItem)
    else // new one can be inserted...
      List.Insert(i,AItem)
  end
end;
```

Merk op dat bovenstaande code niet echt efficient is, omdat ik slechts een lineair zoekalgoritme gebruik om de juiste plek te zoeken waar nieuwe elementen toe gevoegd moeten worden (dit kan beter een binair zoekalgoritme worden, maar dat laat ik over als oefening voor de lezer).
Het gebruik van de TObjectSortedList kan op de inmiddels gebruikelijke manier:

```
procedure TForm1.btnSortedLst(Sender: TObject);
var
  i: Integer;
  SortedList: TObjSortList;
begin
  SortedList := TObjectSortedList.Create;
  try
    for i:=0 to Pred(ComponentCount) do
        SortedList.Push(Components[i]);
    // Usage of ObjSortList...
    Memo1.Lines.Clear;
    while SortedList.Count > 0 do
      Memo1.Lines.Add(
      (SortedList.Pop AS TComponent).Name)
  finally
    SortedList.Free
  end
end;
```

Uiteraard is het ook mogelijk om op basis van een andere property (of waarde) de elementen van de TObjSortList te sorteren.

# WebSnap Adapters

*WebSnap is de nieuwe cross-platform architecture voor het bouwen van web server toepassingen in de Enterprise versies van Delphi 6 en Kylix 2. WebSnap bevat een flinke hoeveelheid wizards, componenten en design-time editors, en is daarnaast ook nog eens uitbreidbaar met eigen hulpmiddelen.*

## Adapters

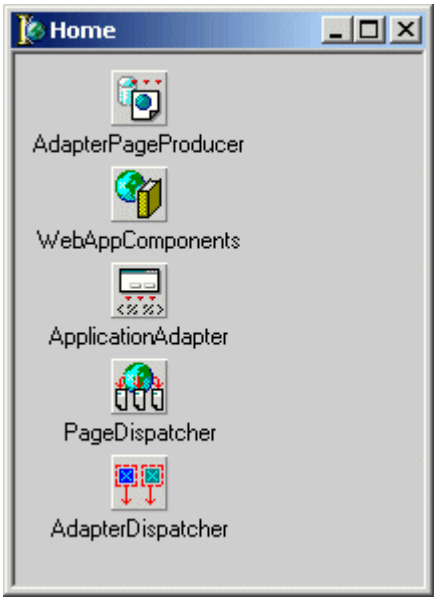
Deze keer wil ik de centrale rol van de TAdapters binnen WebSnap laten zien. Een Adapter kun je zien als de lijm tussen de data en de presentatie laag. De informatie die je wilt presenteren kan hierbij uit een aantal bronnen komen, waaronder databases (met de TDataSetAdapter component). De presentatie, oftewel de opbouw van de resulterende HTML pagina's gaat met behulp van een TAdapterPageProducer - een PageProducer die met Adapters kan omgaan.

Ten einde de koppeling tussen informatie en presentatie mogelijk te maken bestaat een adapter component uit twee onderdelen: velden en akties. Als voorbeeld zal ik met behulp van een standaard TAdapter iets bouwen - een creditcard verificatie pagina - die ik later (in de Advanced WebSnap Clinic) in de TCreditCardAdapter custom adapter component zal inbouwen.

Start Delphi 6 Enterprise (of Kylix 2 Enterprise), en begin een nieuwe WebSnap toepassing (via *File | New - Other*, ga naar de WebSnap tab en kies voor WebSnap Appication). Kies een web server toepassing die je kunt testen op je machine - afhankelijk van de web server die erop staat. Bij twijfel kun je altijd een Web App Debugger toepassing maken, die je kunt testen zonder web server.

Kies voor een Page Module, en verander de Page Name in "Home" (of iets dergelijks). Omdat ik meteen in deze home pagina het gebruik van een Adapter wil laten zien, moeten we even op de "Page Options" knop drukken en in de Application Module Page Options dialoog aangeven dat ik een AdapterPageProducer wil gebruiken in plaats van - default - een normale PageProducer. De laatste is namelijk gewoon een "oude" WebBroker PageProducer, die niet met Adapters om kan gaan.

Figuur 1 laat het resultaat zien. Vijf componenten staan er al, waaronder de AdapterPageProducer, de WebAppComponents, de ApplicationAdapter, de PageDispatcher en de AdapterDispatcher.




*Figuur 1. WebSnap Page Module*

Ga nu naar de WebSnap tab en bekijk de componenten erop. Er zijn al een aantal Adapter componenten aanwezig, namelijk (links) de TAdapter, TPagedAdapter, TDataSetAdapter, TLoginFormAdapter, etc. Daarnaast zie een ApplicationAdapter component in Figuur 1 - een van de zgn. globale adapters, waar je er maar eentje per toepassing van nodig hebt. Twee andere voorbeelden hiervan zijn de EndUserAdapter en EndUserSessionAdapter (nodig om in te loggen).

## TAdapter

We beginnen met een normale TAdapter component. Zet die op de Page Module, en klik er met de rechtermuisknop op. Je kunt nu kiezen voor de Fields Editor (om nieuwe Adapter velden aan te maken) en de Actions Editor (om uiteraard nieuwe Adapter akties aan te maken).

### Adapter Fields

De Adapter velden vormen de doorgeefluiken voor de informatie die je wilt presenteren. Ze kunnen of zelf direkt de gegevens opslaan, of die ergens anders vandaan halen. Voor de standaard TAdapter zijn zes mogelijke soorten Adapter velden aanwezig: de AdapterBooleanField, het generieke AdapterField (een string of Variant), de AdapterFileField, de AdapterImageField, de AdapterMemoField en AdapterMultiValueField. Ieder veld is afgeleid van TCustomAdapterField die geen gepubliceerde properties of events heeft, maar wel al intern de OnGetValue aanroept om de waarde van het Adapter veld op te halen, te valideren (met OnValidateValue), te wijzigen

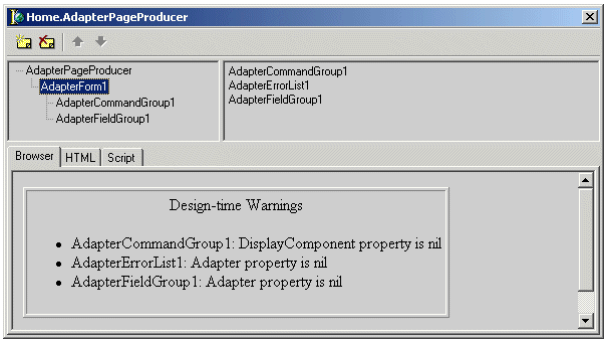
(OnUpdateValue) en te laten zien (met OnGetDisplayText). Zie de on-line help voor meer informatie over de verschillende adapterveld soorten. Voor het voorbeeld in dit artikel wil ik een drietal "normale" Adapter velden gebruiken. Geef ze de namen AdaptCreditCardNumber, AdaptExpirationDate en AdaptUserName.
Selecteer nu elk van de drie Adapter velden en gebruik de Object Inspector om de DisplayLabel properties een leuke tekst te geven (die zien we straks nog terug). Je kan ook de Required property op True zetten voor alle velden, omdat ze allemaal verplicht zijn voor de credit card verificatie. De velden hebben een aantal event handlers, waaronder de OnGetValue die we kunnen gebruiken om de waarde van het veld op te halen (bijvoorbeeld uit een vorige bezoek of sessie). Dat zullen we zo zien; eerst een aktie toevoegen.

### Adapter Actions

Nu we een drietal Adapter velden hebben aangemaakt wordt het tijd om na te denken over de aktie die we eraan toe kunnen voegen. Sluit de Adapter Fields Editor en klik weer met de rechtermuisknop op de Adapter component, maar kies nu de Adapter Actions Editor. Hier kunnen we maar één soort adapter aktie toevoegen: de standaard AdapterAction. Geef hem de naam UseCreditCard, en geef de DisplayLabel property weer een leuke tekst. De UseCreditCard aktie heeft een vijftal event handlers, waaronder OnExecute, maar ook de OnGetEnabled, OnGetParams, OnBeforeGetResponse en OnAfterGetResponse.

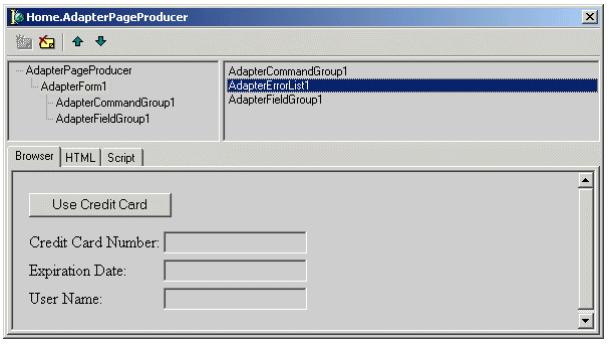
## TAdapterPageProducer

Voordat ik de andere adapters laat zien, lijkt het me eerst zinvol om de samenwerking tussen de adapters en de TAdapterPageProducer te demonstreren. Omdat ze allebei al op dezelfde Page Module staan, kun je direkt dubbelklikken op de AdapterPageProducer. Dit resulteert in de Web Page Editor, die sommige mensen wellicht nog kennen uit de Delphi 5 InternetExpress tijd. Hij werkt inderdaad nog precies hetzelfde, alleen nu met andere componenten (gekoppeld aan WebSnap adapter velden en akties).
Het scherm bestaat uit drie delen. Linksboven is de boom met web componenten. Daar meteen rechts van is het window waarin de kinderen van het huidig geselecteerde component in staan. Dit is nodig, omdat links alleen maar componenten staan die kinderen hebben of kunnen krijgen (dus alle "bladeren" staan alleen maar rechts - is soms even zoeken). Onderaan staat de preview, zowel de browser preview als de HTML en het server-side script dat gegenereerd wordt.
Met de rechtermuisknop of de Insert knop kun je nieuwe componenten toevoegen als kind van het huidig geselecteerde component in de boom. Begin met een AdapterForm, en daaronder een AdapterFieldGroup, AdapterCommandGroup en AdapterErrorList. We krijgen nu meteen een paar "Design-time Warnings" te zien:




*Figuur 2. Web Page Editor en Design-time Warnings*

Merk op dat de AdapterErrorList is wel zichtbaar in het rechter window, maar niet links (omdat er geen subcomponenten meer onder de ErrorList kunnen komen).
Om de drie warnings op te lossen moeten we eerst de DisplayComponent property van de AdapterCommandGroup laten wijzen naar de AdapterFieldGroup, en de Adapter property van zowel de AdapterFieldGroup als de AdapterErrorList laten wijzen naar de Adapter component die op de Page Module staat (dus Adapter1, en niet de ApplicationAdapter). Het resultaat ziet er dan al een stuk beter uit:

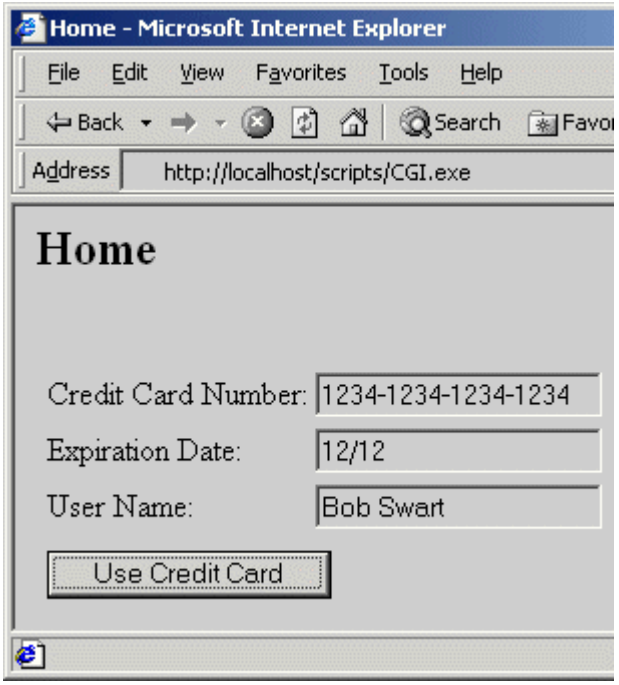



*Figuur 3. Web Page Editor met Preview in de Browser*

Nu nog optioneel de volgorde veranderen (zodat bijvoorbeeld de AdapterError list bovenaan de lijst staat, en de eventuele foutmeldingen dus bovenaan het scherm komen te staan), en dan kunnen we het geheel gaan testen.

## Eerste Test

Bewaar alles (bijvoorbeeld de Page Module in wsWebMod.pas en het project in CGI.dpr), compileer de WebSnap toepassing, en gebruik hem op de geschikte wijze (een CGI executable of ISAPI DLL zal naar de scripts of cgi-bin directory moeten, een Web App Debugger executable kun je direct vanuit de Delphi 6 IDE uitvoeren). Vergeet bij een CGI of ISAPI toepassing niet om ook de wsWebMod.html mee te nemen. Je kunt de toepassing als CGI executable hieronder in de IE5 browser zien als http://localhost/scripts/CGI.exe




*Figuur 4. WebSnap toepassing in Internet Explorer*

Als je de WebSnap toepassing op een machine zet waar Delphi 6 zelf niet ook staat, zul je een tweetal .tlb bestanden uit de Delphi6\bin directory naar die machine moeten meenemen, en aldaar moeten registreren met tregsvr. Dit kan met:

```
tregsvr WebBrokerScript.tlb
tregsvr stdvcl40.dll
```

Je hebt daarnaast ook de Microsoft Script Engine nodig, maar die staat al op elke machine waar Windows 2000 of Internet Explorer versie 5 of hoger staat. Indien nodig kun je de Microsoft Script Engine ook zelf ophalen van de Microsoft website te http://msdn.microsoft.com/scripting/

Als je de drie editvelden invult en dan op "Use Credit Card" drukt gebeurt er verder niks: je krijgt weer hetzelfde scherm met lege velden. Om iets zinvols te doen met de gegevens zullen we de OnExecute event handler van de UseCreditCard Adapter aktie moeten invullen, net als de OnGetValue event handlers van de drie Adapter velden.

### OnExcecute

In de OnExecute kunnen we de credit card gegevens testen, maar ook opslaan om in een later scherm te gebruiken. Dit kan in een speciaal WebSnap sessie object, waarvoor we het TSessionServices component nodig hebben. Maar omdat die niet werkt in CGI toepassingen (het SessionService object wordt na ieder request weer uit het geheugen verwijderd), sla ik voor dit voorbeeld de gegevens maar even op in een test bestand, en haal ze er in de OnGetValue weer uit. Puur om te laten zien dat het werkt, meer niet.
Klik weer met de rechtermuisknop op de Adapter component en selecteer de Action Editor. Kies hierin voor de UseCreditCard action en ga naar de Object Inspector om zijn OnExecute event handler als volgt in te vullen:

```
const
  sCreditCard     = 'CreditCard';
  sCardNumber     = 'CardNumber';
  sExpirationDate = 'ExpirationDate';
  sUserName       = 'UserName';
  IniFileName     = 'WebSnap.ini';

procedure THome.UseCreditCardExecute
  (Sender: TObject; Params: TStrings);
var
  Value: IActionFieldValue;
  Logfile: TIniFile;
begin
  LogFile := TIniFile.Create(IniFileName);
  try
    Value := AdaptCreditCardNumber.ActionValue;
    if Value.ValueCount > 0 then
      Logfile.WriteString(sCreditCard,
        sCardNumber,Value.Values[0]);
    Value := AdaptExpirationDate.ActionValue;
    if Value.ValueCount > 0 then
      Logfile.WriteString(sCreditCard,
        sExpirationDate,Value.Values[0]);
    Value := AdaptUserName.ActionValue;
    if Value.ValueCount > 0 then
      Logfile.WriteString(sCreditCard,
        sUserName,Value.Values[0]);
  finally
    LogFile.UpdateFile;
    LogFile.Free
  end
end;
```

### OnGetValue

In de OnGetValue event handler van AdaptCreditCardNumber, AdaptExpirationDate en AdaptUserName kunnen we de adapter velden van een waarde voorzien (zonder dat zullen ze

leeg zijn). De implementatie van eentje van hen is als volgt (de rest is hetzelfde). Merk op dat de sCreditCard en sCardNumber strings ook hier weer gebruikt worden - al is het maar om tikfouten te voorkomen.

```
procedure THome.AdaptCreditCardNumberGetValue
  (Sender: TObject; var Value: Variant);
var
  Logfile: TIniFile;
begin
  LogFile := TIniFile.Create(IniFileName);
  try
    Value := Logfile.ReadString(sCreditCard,
      sCardNumber,'8888-8888-8888-8888')
  finally
    LogFile.Free
  end
end;
```

Het resultaat hiervan is dat je de eerste keer de default waardes ziet (die je opgeeft bij het inlezen van de adapter velden uit de WebSnap.ini file), en daarna steeds de vorige waarden terugziet.

## Meer Adapters

Naast de generieke TAdapter, die ik zojuist heb gedemonstreerd, bevat Delphi nog een aantal speciale Adapters. De TPagedAdapter heeft als extra de PageSize property, die aangeeft hoeveel elementen er per pagina moet worden weergegeven. Dit is bijvoorbeeld zinvol bij een zoekmachine, waarbij je maar tien resultaten per pagina wilt zien. Of een catalogus, waarbij je ook maar een beperkt aantal items per pagina wilt zien. De PageSize property is ook onderdeel van de TDataSetAdapter, dus is het makkelijker om het effekt te zien met deze laatste component. De TDataSetAdapter wordt gekoppeld aan een DataSet (table of query) en haalt zijn Adapter velden en akties direkt uit deze dataset. Erg fijn om op die manier snel van een tabel naar een interaktief HTML formulier te gaan.
Tot slot is de TLoginFormAdapter een speciale adapter die je kunt gebruiken om gebruikers in te laten loggen. Met UserName, Password en NextPage velden en de ActionLogin action.

## TCreditCardAdapter

Tijdens de Advanced WebSnap Clinic van 21 februari zal ik o.a. laten zien hoe we zelf custom WebSnap adapter componenten (zoals een TCustomCreditCardAdapter component) kunnen bouwen, registreren, installeren en gebruiken in onze WebSnap toepassingen. Voor meer informatie zie http://www.eBob42.com/training.

## Delphi 6.01 Update en Web

De meeste mensen die Delphi 6 gebruiken zullen inmiddels de Delphi 6.01 Update al geïnstalleerd hebben (de goede versie dan, van 27 september in plaats van die van 6 september).
Helaas heeft deze 6.01 update een aantal zaken "opgelost" die ingrijpende gevolgen kunnen hebben, met name voor WebBroker en WebSnap toepassingen. Er wordt namelijk vanaf de versie 6.01 update extra moeite gedaan om te zorgen dat de web modules na gebruik worden opgeruimd. Helaas is het opruimen van web modules gevoelig voor de volgorde waarin dat gebeurt, met name als er ook database zaken gedaan worden. Om een lang verhaal kort te maken: mijn CGI web server toepassingen genereerde na de 6.01 update plots een critical error, en mijn ISAPI DLLs bleven hangen als ik ze uit het geheugen wilde unloaden.

Afhankelijk van het type web server toepassing moet je het bestand CGIApp.pas (voor CGI), ApacheApp.pas (voor Apache), ISAPIApp.pas (voor ISAPI/NSAPI) of ComApp.pas (voor de Web App Debugger) aanpassen. Voor de eerste drie bestanden moet je de (enige) regel code in de finalization sectie in commentaar zetten, en in de initialization sectie een regel met AddExitProc toevoegen, als volgt:

```
initialization
  InitApplication;
  AddExitProc(DoneApplication);
finalization
//DoneApplication
end.
```

Voor de ComApp.pas moeten we eerst zelf een DoneApplication routine schrijven (die bestaat daar nog niet), en vervolgens de initialization en finalization sectie aanpassen:

```
procedure DoneApplication; // nieuw
begin
  WebReq.WebRequestHandlerProc :=  nil;
  FreeAndNil(FComWebRequestHandler);
end;

initialization
  WebReq.WebRequestHandlerProc :=
    ComWebRequestHandler;
  AddExitProc(DoneApplication);
finalization
//FreeAndNil(FComWebRequestHandler);
end.
```

Wie meer wil weten over WebBroker en WebSnap kan een van de Delphi 6 Clinics in 2002 volgen.

# Delphi 6 Clinics in 2002

*De Delphi 6 Clinic onderwerpen in 2002 zijn als volgt:*

10 januari

## Delphi 6 - dbexpress & DataSnap

In deze Delphi 6 Clinic behandelen we de cross-platform dbExpress/DataCLX data access laag en de multi-tier architectuur DataSnap (de nieuwe naam van MIDAS).

31 januari

## WebBroker & WebSnap

In deze Delphi 6 Clinic starten we met WebBroker als architectuur om web server toepassingen te ontwikkelen, en breiden deze ervaring uit met InternetExpress en WebSnap. De WebSnap Architectuur onderkent een centrale rol voor Adapters. We zullen zien hoe de Adapters werken en alle componenten van de WebSnap tab van het component palette aan de orde laten komen, door steeds kleine web server toepassingen te schrijven.

21 februari

## Advanced WebSnap

In deze Delphi 6 Clinic zien we hoe WebSnap achter de schermen in elkaar zit en bouwen WebSnap custom componenten (zoals een TCreditCardAdapter). Daarnaast besteden we veel aandacht aan sessiemanagement, en ontwikkelen een techniek die ook werkt voor CGI toepassingen.
*Voorkennis van WebSnap is vereist (zoals de Clinic van 31 januari).*

14 maart 2002

## BizSnap (XML/SOAP) & Web Services

Deze Delphi 6 Clinic begint met een overzicht van de XML ondersteuning in Delphi 6, met XML Document Programming, de XML Data Mapping en tenslotte de XML Mapper (een tool waarmee XML documenten naar data packets te transformeren zijn - en terug).
Daarna verschuift de aandacht naar SOAP en Web Services. Na een korte introduktie in zowel SOAP en WSDL als de toepassing van Web Services, zullen we zien hoe we in Delphi 6 bestaande Web Services kunnen gebruiken en nieuwe cross-platform Web Services kunnen maken en deployen.

4 april 2001

## Web Solutions met WebSnap & BizSnap

Tijdens deze Delphi 6 Clinic bouwen we een multi-tier e-business toepassing met behulp van de technieken uit DataSnap, WebSnap en BizSnap (XML en SOAP). We zullen met behulp van BizSnap (de B2B XML-transformatie van een XML document naar een datapacket en terug) en WebSnap een e-business/e-commerce toepassing bouwen, met alle toeters en bellen zoals login, database access, zoeken, bestellen, winkelwagentjes, etc.
*Enige voorkennis van WebSnap en BizSnap is vereist. (zoals de Clinics van 31 januari, 21 februari en 14 maart).*

De lokatie van de Delphi 6 Clinics is een cursusruimte bij Centric PSS in Eindhoven, waar de trainingen op donderdag worden gegeven van 9.00 tot 17.00 uur. De clinic is praktisch (dus niet alleen maar een slideshow), doch niet hands-on (voor de deelnemers), alhoewel u natuurlijk wel een eigen laptop kan meenemen en gebruiken gedurende de dag. Tijdens de clinic worden de onderwerpen door de trainer direkt toegepast met Delphi 6 Enterprise. Door het beperkte aantal deelnamers (maximaal 10) en het "live" karakter van de clinic bestaat hierbij altijd de mogelijkheid tot het samen onderzoeken van zaken die wellicht net buiten het cursusmateriaal vallen. Voor cursisten is de trainer (ook na de clinics) altijd per e-mail te bereiken voor vragen of nadere uitleg!

De adviesprijs per afzonderlijke clinic is **€ 395** per persoon (ex.btw). Deze prijs is inclusief koffie, lunch en (engelstalige) syllabus geschreven door Bob Swart. De syllabus bevat meer informatie dan tijdens de clinic behandeld kan worden, en is dan ook een geschikt naslagwerk voor gebruik in de periode na de clinic.
Indien twee of meer personen van hetzelfde bedrijf deelnemen aan dezelfde trainingsdag geldt **een korting van € 45 per persoon per dag**.

*Bovendien geldt dat wie zich inschrijft voor alle vijf trainingsdagen (van 10 januari t/m 4 april) er maar vier hoeft te betalen, dus 20% korting!*

**Speciale Aanbieding:** Wie zich voor twee of meer Clinics inschrijft krijgt een door mij gesigneerd exemplaar van de Delphi 6 Developer's Guide, de Kylix Developer's Guide of de C++Builder 5 Developer's Guide kado.