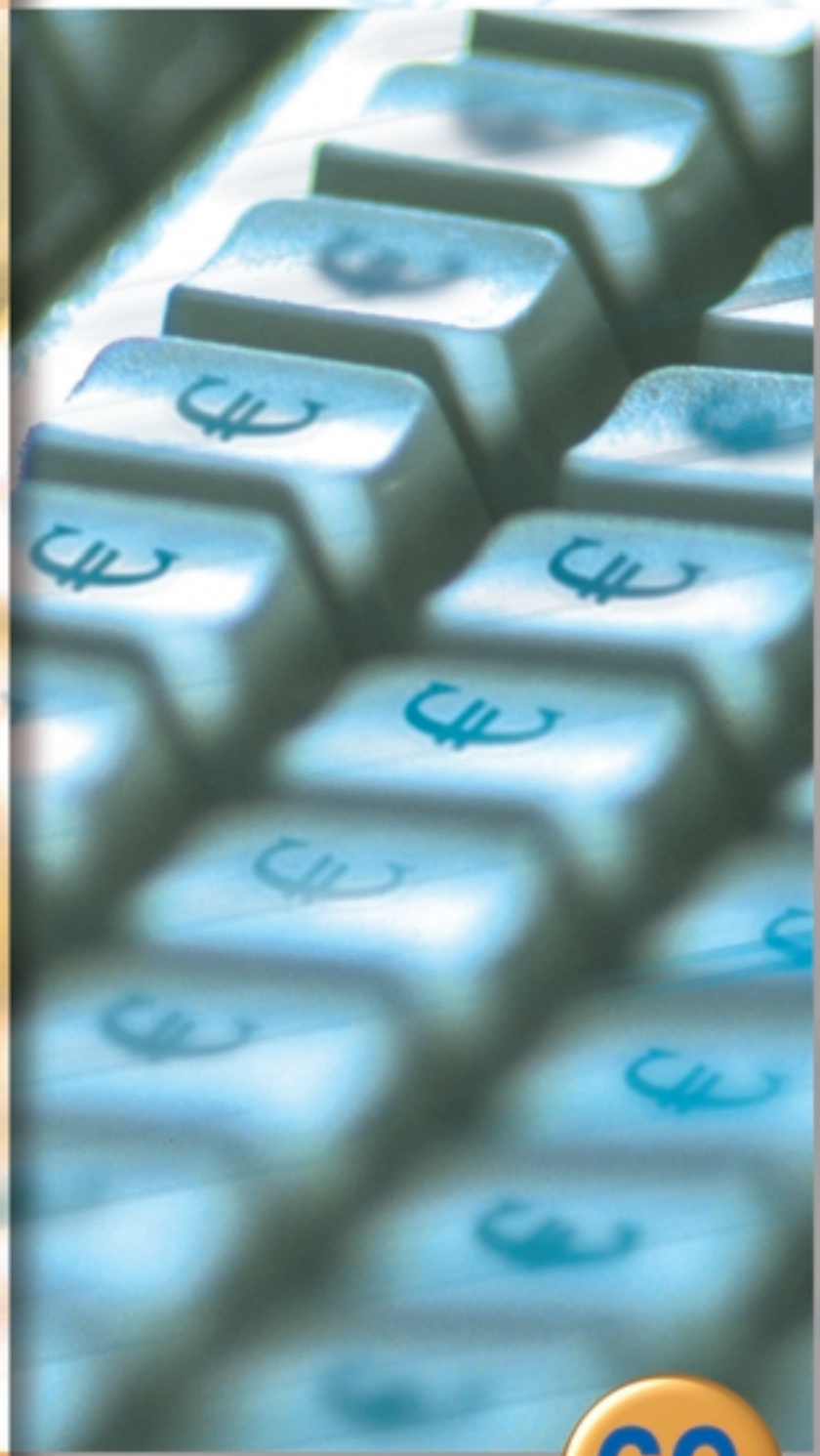
SDGN

Software Developers Group Netherlands

MAGAZINE

in dit nummer



Magazine voor Softwareontwikkelaars • Elfde jaargang • No. 69 •December 2001 • Een uitgave van Software Developers Group Netherlands

Colofon

SDGN Magazine Nr. 69

Uitgave:

Software Developers Group Netherlands

Ontmoetingsplaats en platform voor professionele ontwikkelaars:

Postbus 506,
7100 AM Winterswijk.

Telefoon : (0543) 51 80 58
Telefax : (0543) 51 53 99

Internet : www.sdgn.nl

Email:

Bestuur : bestuur@sdgn.nl
Sectiehoofden : secthfd@sdgn.nl
Redactie : redactie@sdgn.nl
WebSite Team : webteam@sdgn.nl
Secretariaat : info@sdgn.nl

Bestuur van SDGN:

Ad van de Lisdonk, voorzitter
Joop Pecht, secretaris
Rob Suurland, penningmeester
Robert van der Hulst, vice-voorzitter
Pepijn Smits, bestuurslid

Sectiehoofden:

Clipper : Robert van der Hulst
Joop Pecht
VO : Ed Richard
Delphi : Harry Maes
Microsoft-Development : Remi Caron
Internet-Development : Guus Hofstede

Redactie SDGN-Magazine:

Jan van der Graaf (eindredacteur)
Mark Blomsma
Johan Parent
Patrick Vletter
Mark Vroom
Rob Willemsen

SDGN Web Team:

Ronald Steenbergen • Cor Fransen
Erik Visser • Joop Muis

Met medewerking van:

Bram Laumans, Peter van Ooijen, Remi Caron, Ed van Akkeren, Bob Swart, Mart van Ineveld, Les Pinter, Ronald van Aken, Peter van Ooijen, Frits Bosschert, Anthony Smith, Marco Hemmes, Mike French, Paul Maskens & Andy Kramek, Mark Vroom

Vormgeving en opmaak:

Reclame-adviesbureau JE/ES, Winterswijk (www.je-es.nl)

Druk:

Drukkerij Loor b.v., Varsseveld (www.loor.nl)

Inhoudsopgave

Adverteerders

Listings

Zie de WebSite voor eventuele source files uit deze uitgave.

Adverteren?

Informatie over adverteren en de advertentietarieven kunt u vinden op www.sdgn.nl onder de rubriek Magazine. Deze informatie is tevens telefonisch of per email verkrijgbaar via het SDGN secretariaat.

Software Developers Group Netherlands

Het is herfst. En niet zo'n beetje. In het begin zag het er buiten nog wel mooi uit met al die kleurtjes maar inmiddels is het ronduit deprimerend om uit het raam te kijken. Bijna iedereen die ik ken is ziek of onderweg. De enkeling die niet ziek is, is chagrijnig omdat iedereen in zijn omgeving wel ziek is of onderweg. Daar komt bij dat er nog steeds teveel mensen gezond zijn. Anders stond ik niet zo vaak in de file. Afijn, u begrijpt mij wel: Herfst dus.

Het kan buiten nog zo'n guur weer zijn, maar bij SDGN is het lekker warm. Daar knappert het haardvuur zogezegd. Alles loopt eigenlijk best lekker. We weten inmiddels hoe we conferenties moeten organiseren, een magazine moeten produceren en een website moeten onderhouden. Voor je het weet beginnen dingen dan automatisch te gaan en na een tijdje lijkt het net of het minder leuk wordt. Mede om hiervoor te waken en elkaar scherp te houden, hebben we - naast het reguliere, maandelijkse werkoverleg - één keer per jaar een zogenaamde 'heisessie.' Dat is het moment waarop we met name vooruitkijken naar de uitdagingen en kansen die voor ons liggen. Dat is het moment waarop je het hartgrondig met elkaar oneens mag/moet zijn over de te voeren koers. Dan dondert het nog wel eens in de vergaderzaal. Net een herfststorm, maar dan zonder depressie.

Van u horen we dat we ons werk goed doen. U bent tevreden over de producten die uw vereniging u biedt. Het leek u alleen wel fijn indien de sectiescheiding werd opgeheven. We kauwen dan een tijdje op het hoe en waarom, en besluiten dat het wat ons betreft ook een goed idee is. Op de ALV hebben we goedkeuring hiervoor gehad en per 1 januari bent u dus lid van een 'verenigd' SDGN. Elke CttP wordt dan een soort mini-CttM. Zoveel feestdagen per jaar! Het lijkt wel Kerstmis. Alleen bij *onze* feestjes komt uw schoonmoeder niet op bezoek ☺

Toch zijn wij niet helemaal tevreden. We zien ons ledenaantal namelijk heel langzaam teruglopen. En dat is vreemd. De wereld om ons heen wordt steeds complexer, het aantal tools en platforms neemt alleen maar toe en de tijdsdruk op projecten wordt groter en groter. We kunnen met zijn allen de hulp van onze collega's dus heel goed gebruiken. SDGN geeft u dat netwerk via bijeenkomsten en nieuwsgroepen. SDGN houdt u op de hoogte via magazine, website en sessies. SDGN haalt toppers van overal ter wereld naar Nederland om u te helpen uw gereedschap onder de knie te krijgen. *Elke* ontwikkelaar in het Nederlandse taalgebied zou dus eigenlijk bij ons moeten zijn aangesloten. Waarom is dat dan niet het geval?

Voor een deel ligt dat ongetwijfeld aan de marketing en PR van SDGN. Daar zijn we geen sterren in en daar gaan we dus ook wat aan doen. Maar het ligt voor een deel ook aan u. U bent immers SDGN. De kracht van de vereniging is haar vereniging. U verkrijgt geen voorsprong door stiekem lid te zijn van SDGN en er vooral voor te zorgen dat uw collega's niet van het bestaan van onze vereniging afweten. U verkrijgt juist een voorsprong door uw collega's te introduceren. Als u nou komende herfst en winter gebruikt om net als wij pro-actief leden te werven voor uw eigen vereniging gaan we met zijn allen een hele mooie zomer tegemoet!

Ad van de Lisdonk

Het lijkt stil op het Visual Objects-front, maar dat is maar schijn. Er wordt hard gewerkt op verschillende fronten om CA duidelijk te maken dat het nu het moment is om iets te ondernemen. Met name sinds de laatste CARE conferentie (voor User Group Presidents) in Orlando is er bij het CA-management een (gewillig?) oor gevonden en worden er regelmatig telefonische conferenties gehouden waarin de toekomst van applicatieontwikkeling in het algemeen en Visual Objects in het bijzonder onderwerp van gesprek zijn.

De eerste resultaten zijn inmiddels zichtbaar, met ingang van 2002 is er wederom een Visual Objects Technical Track tijdens CA World. Mocht je geïnteresseerd zijn daar als spreker naar toe te gaan aarzel dan niet even contact op te nemen met ondergetekende. Doe het als het even kan niet buiten de SDGN om, want er zijn voor de vereniging voordelen aan verbonden als leden optreden als spreker, en je wordt er zelf niet slechter van. Deze zomer is er ook weer een Australische DevCon geweest, waar de SDGN vertegenwoordigd is door Robert van der Hulst en ondergetekende, de opkomst was niet spectaculair, maar het is duidelijk dat VO behoorlijk leeft Down Under. De Duitse DevCon is ondertussen ook weer achter de rug, en daar is bekend gemaakt door Ansgar Trimborn (huidige Product Owner) dat men alles in het werk stelt zo snel mogelijk een 2.5c-patch uit te brengen. Deze patch bestaat vooral uit bugfixes, die men graag beschikbaar wil stellen aan de VO ontwikkelaars, en support voor nieuwe Windows features zoals XP-Themes. Mocht je dus nog bugs willen melden dan is het nu een goed moment. Dat kan via de VO nieuwsgroep van de SDGN, of direct aan CA via de support medewerkers, te bereiken via support@ca.com waar ook de FAQ te vinden is: http://support.ca.com/techbases/ca-visobj/ca-vo1000.html.

Los van deze patch zijn de User Group Presidents en enkele vertegenwoordigers van grote bedrijven die VO gebruiken op dit moment met CA in dialoog om te praten over de toekomstplannen. Er wordt daar veel besproken, maar helaas is er nog niets publiek mede te delen. Uiteraard zal ik ervoor zorgen dit te doen via de website zodra er iets concreets te melden is.

Ed Richard,
Sectiehoofd Visual Objects

Software Developers Group Netherlands

tools

- ✔ VB
- ✔ IE6
- ✔ MSXML4
- ✔ XMLNotepad

auteur

Bram Laumans

XML validatie – zit het er goed in?

Samenvatting

Dit artikel beschrijft hoe de inhoud van XML-documenten gevalideerd wordt met een Document Type Definition en een XML-Schema. De structuur van beide soorten documenten wordt kort uiteengezet en aan de hand van een praktisch voorbeeld wordt de validatie gedemonstreerd. Een VB-applicatie (broncode beschikbaar) valideert de meegeleverde XML-documenten. Deze zijn te downloaden via de SDGN website.

Systeem: IE 6 en MSXML4

De voorbeelden van dit artikel zijn gebaseerd op Internet Explorer 6 en de MSXML4 parser. Ook wordt voor de validatie met Document Type Definitions gebruik gemaakt van XMLNotepad (alles gratis te downloaden bij www.microsoft.com/downloads.)

Wat is valideren?

Valideren is het controleren of de inhoud van een bestand overeenkomt met dat wat er verwacht wordt. Valideren is nodig om te voorkomen dat er bij het automatisch verwerken van het bestand iets fout gaat. Stel dat u bij drie verschillende Internetbedrijven een offerte aanvraagt voor het leveren van boeken. Zowel de prijs als de levertijd zijn voor u belangrijke criteria om te bepalen bij wie u de bestelling plaatst. U verwacht dus bij de of-ferte een prijs en een levertijd. Als nu één van die offertes geen levertijd bevat, kunt u uw vergelijking niet maken. En als u een programma hebt geschreven die de vergelijking geautomatiseerd laat verlopen, dan loopt uw programma wellicht vast. Essentieel is dan een controlemechanisme dat kijkt of de offerte zowel een prijs als een levertijd bevat. Met andere woorden: of de inhoud van het bestand overeenkomt met wat er verwacht wordt.

DTD en Schema

Dit artikel gaat over het valideren van XML-documenten. De inhoud van een XML-document kan op twee manieren gevalideerd worden. De eerste manier is met behulp van een Document Type Definition (DTD). XML (eXtensible Markup Language) is een subset van SGML (Standardized General Markup Language). Dit betekent onder andere dat SGML wel XML kan lezen, maar niet andersom. Bij de ontwikkeling van XML was er direct behoefte aan een wijze van validatie. In SGML is het mogelijk om de structuur van SGML-documenten te beschrijven met een DTD. Omdat met SGML een XML-document gelezen kan worden, kan een DTD ook probleemloos de structuur van een XML-document valideren. En zo is het gekomen dat er voor het werken met valide XML-documenten zowel de XML-syntax voor het XML-document als de SGML-syntax voor de DTD gebruikt wordt. Dat voelt net zo aan als het leren van het Frans met een grammatica voor het Latijn. Het Frans is ontstaan uit het Latijn, maar het heeft inmiddels een hele eigen ontwikkeling doorgemaakt, en is niet meer goed met het Latijn te beschrijven.

De tweede manier voor het valideren van een XML-document is het XML-Schema. Het XML-Schema is vrij nieuw: de definitieve aanbeveling ('Recommendation') van het World Wide Web Consortium is op 2 mei 2001 gepubliceerd. Met een XML-Schema kan de vali-datie van een XML-document op een XML-manier gebeuren. Een XML-Schema gebruikt namelijk wèl de XML-syntax: het is dus zelf ook een XML-document, met alle eisen en kenmerken die daarbij horen. Om de vergelijking met het Frans nogmaals te gebruiken: nu gaat het om het leren van het Frans met een franse grammatica. Een tweede voordeel van XML-schema's is dat er veel meer verschillende datatypes te definiëren zijn. Alle types die in databases gebruikelijk zijn, kunnen hier probleemloos terugkomen. Een XML-Schema kan veel beter dan een DTD vastleggen dat het element "hoogte ten opzichte van zeeniveau" van het datatype integer zijn met een waarde tussen -11515 (Cook-diep) en +8848 (Mount Everest). Ook zijn er veel meer mogelijkheden om zelf datatypes te maken. Dit wordt verderop in dit artikel behandeld.

Intern versus Extern Valideren

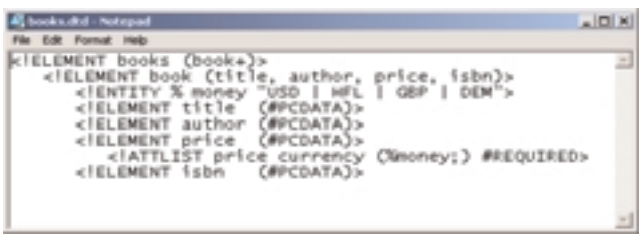
De validatie van een XML-document kan in het document zelf of in een apart document zijn opgenomen. In het eerste geval is er sprake van een interne validatie, in het tweede van een externe validatie. In dit artikel wordt alleen ingegaan op de meest gebruikte manier: het extern valideren.

Het XML-bestand

In dit artikel wordt gebruik gemaakt van het XML-document books.xml. Hierin worden van drie boeken steeds vier gegevens getoond: de titel, de auteur, de prijs en het ISB-Nummer. Van de prijs wordt tevens de valuta meegegeven. Het bestand is afgebeeld in figuur 1.

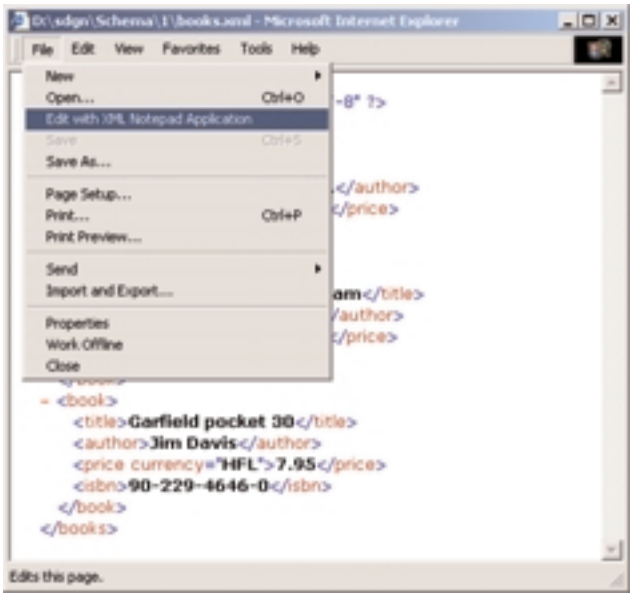
Figuur 1: books.xml

De eerste validatie voor dit document wordt gemaakt met een DTD. In de DTD wordt vastgelegd dat <books> het root-element is. Door een plusje (+) toe te voegen achter "book" wordt aangegeven dat <books> minimaal één child-element <book> bevat. De andere twee tekens zijn de asterisk (*) om nul tot oneindig children aan te geven, en het vraagteken (?) om nul of één (oftewel een optioneel) child-element aan te geven. Geen teken betekent dat het child-element precies één keer voorkomt.
Van elk <book> moeten de <title>, de <author>, de <price> en het <isbn> gegeven worden (alle vier slechts eenmaal). Deze vier elementen zijn van het type #PCDATA, wat Parsed Character Data betekent. Dit is vergelijkbaar met een string of een tekstveld. Het wil zoveel zeggen als dat de parser ook de inhoud van deze velden bekijkt. Het is daarom verboden om in de titel bijvoorbeeld de kleiner dan (<) en groter dan (>) tekens te gebruiken, omdat die gereserveerd zijn voor het begin en het einde van een tag. Bij de prijs is het verplicht gesteld om te vermelden welke munteenheid gebruikt wordt. Hiervoor is een attribuut gedefinieerd onder <price> met de naam currency. Currency is van het type money, dat eerder in de DTD gedefinieerd is als entiteit (lees: een eigen data-type). De entiteit money is hier gedefinieerd als de waarde USD, HFL, GBP of DEM. Er mag dus enkel gekozen worden uit de Amerikaanse dollar, de Nederlandse gulden, de Engelse pond en de Duitse mark. Andere opties zijn niet toegestaan. Achteraan bij currency wordt met #REQUIRED aangegeven dat de currency verplicht is. De DTD is opgeslagen als books.dtd en is afgebeeld in figuur 2.

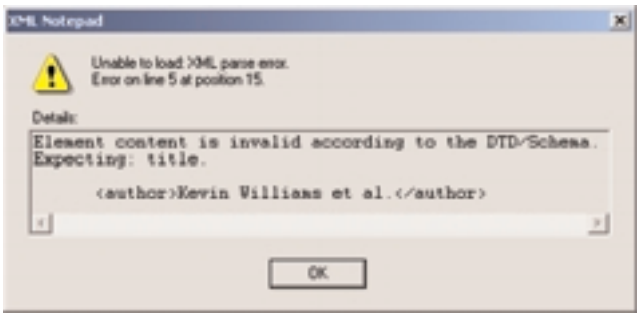


Figuur 2: books.dtd

In het XML-document wordt de regel <!DOCTYPE books SYSTEM "books.dtd"> opgenomen om de DTD te koppelen aan het XML-document. Het is noodzakelijk dat het woord books direct na DOCTYPE exact hetzelfde geschreven is als het root-element van het XML-document. Let op: Internet Explorer valideert dit niet en zal een XML-document met een foutieve verwijzing gewoon openen. De verwijzing naar de DTD gebeurt met behulp van het woord SYSTEM. Achter SYSTEM komt tussen dubbele aanhalingstekens de naam van de DTD. Dit is uitgebreid met pad-aanduidingen als de DTD in een andere directory staat dan het XML-document. Het testen van een DTD kan eenvoudig met XML Notepad gebeuren. Een XML-document dat met Internet Explorer is geopend, kan via File | Edit with XML Notepad Application (zie figuur 3) direct naar XML Notepad gestuurd worden. Elk bestand dat met een DTD gekoppeld is, wordt alleen door XML Notepad geaccepteerd als het in orde is. Als er iets mis is, komt er een foutmelding. In figuur 4 is de fout weergegeven als de titel van het eerste boek ontbreekt.



Figuur 3: open het XML-document met XML Notepad vanuit Internet Explorer



Figuur 4: foutmelding bij incorrect XML-document.

Namespaces (1)

U heeft nu in een kort overzicht gezien wat een DTD is. Eén van de voornaamste bezwaren tegen een DTD zal u hopelijk opgevallen zijn: een DTD is zelf geen XML-docu-

ment. Waar een XML-document bijvoorbeeld duidelijk eindigt met de afsluitende tag van het root-element, doet een DTD dat niet. Daarom is er hard gewerkt aan een XML-verwante techniek om XML-documenten te valideren. Dit heeft op 2 mei 2001 geresulteerd in de "W3C Recommendation" (W3C = World Wide Web Consortium) voor XML-schema.

Om met een XML-Schema aan de slag te gaan, kunt u niet om het begrip "namespace" heen. Een namespace kunt u het beste vergelijken met een woordenboek. De betekenis van alle gebruikte termen staat erin. Een klein voorbeeld maakt dit duidelijk. In dit artikel staan veel engelse woorden. Het feit dat u dit begrijpt komt door uw onderscheidingsvermogen om woorden uit andere talen als zodanig te herkennen. Een XML-parser kan dat niet. Als ik plotseling Shakespeare aanhaal met 'to be or not to be' dan weet u wat ik bedoel. Of niet. Maar in ieder geval weet u dat er een stuk engelse tekst aangehaald wordt. Als een XML-parser dat ook moet begrijpen, moet die zin met "markup" verrijkt worden. Dan wordt het:
<nl:zin>Als ik plotseling Shakespeare aanhaal met '<eng:quote>to be or not to be</eng:quote>' dan weet u wat ik bedoel.</nl:zin>
Op deze manier weet de parser dat in de nederlandstalige zin (alles tussen <nl:zin> en </nl:zin>) een engels citaat (tussen <eng:quote> en </eng:quote>) is opgenomen. Maar ook 'zin' en 'quote' moeten ergens gedefinieerd zijn. Dat gebeurt bovenin het XML-document met een verwijzing (referentie) naar de namespaces en de afkortingen die hiervoor gebruikt worden. Voor het Nederlands wordt de namespace "Van Dale" gebruikt. Deze wordt aangeduid met de afkorting "nl". Voor het Engels wordt een exemplaar van "the Concise Oxford Dictionary" gebruikt die met "eng" aangeduid wordt. Dit wil zoveel zeggen als: de betekenis van alle woorden die voorafgegaan worden door "nl:" (zoals in dit voorbeeld het woord "zin") is te vinden in Van Dale, en de betekenis van alle woorden met het voorvoegsel "eng:" is te vinden in the Concise Oxford Dictionary. Het resultaat wordt getoond in figuur 5.

Figuur 5: Namespaces in Shakespeare.xml

In het root-element <nl:zin> ziet u twee attributen opgenomen die beide beginnen met "xmlns". Dit staat voor XML-namespace. Direct daarachter, gescheiden door een dubbele punt, staat de afkorting die met de namespace geassocieerd wordt. In dit voorbeeld wordt "nl" geassocieerd met de eerste en "eng" met de tweede namespace. Die namespace wordt gedefinieerd door wat er als waarde aan gegeven wordt. Dat wordt zoals bij alle XML-attributen gedaan door het "=" teken, met daarachter tussen dubbele aanhalingstekens de waarde. De XML-namespace die met "nl" wordt aangeduidt, heeft betrekking op "uri:Van Dale", en de XML-namespace die met "eng" wordt aangeduidt, heeft betrekking op "uri:The Concise Oxford Dictionary". De kreet "uri" betekent Unique Resource Identifier. Vaak wordt voor een URI iets als "http://www.sdgn.nl/artikel" of een andere URL (Unique Resource Location) gebruikt. Dit geeft soms verwarring. Het roept vragen op als: gaat het XML-document dan over het internet naar dat adres? Moet ik een verbinding met het Internet hebben? Nee en nee. De naam die als URI gebruikt wordt, moet enkel en alleen een unieke naam te zijn. Maar omdat een URL ook een unieke naam is, kan die hiervoor dus heel prettig gebruikt worden. En het wordt nog prettiger als de definitie van die namespace ook daadwerkelijk op die plek aanwezig is, zodat anderen die ook kunnen gebruiken. Dat is een "good practice", maar het is niet verplicht.

Nee en nee. De naam die als URI gebruikt wordt, moet enkel en alleen een unieke naam te zijn

Wat is nu de relatie tussen een XML-document, een namespace en een XML-Schema? In een XML-document kunt u zelf allerlei tags verzinnen. Dat is zo eXtensible aan XML. Als u gegevens gaat uitwisselen met anderen is het beter dat die extensibility aan bepaalde regels is gebonden. In het XML-document van dit artikel wilt u niet alleen vastleggen dat van een boek de titel, auteur, prijs en isbn gegeven worden, maar ook nog in welke volg-orde en van welk datatype deze vier waarden zijn.
In een DTD kan dat maar in beperkte mate, in een XML-Schema kan dat veel preciezer. Dus u gaat met XML-Schema uw eigen woordenboek schrijven. Om dit in uw XML-document vast te leggen, heeft u verschillende namespaces nodig. Maar eerst bouwen we het XML-Schema zelf.

XML-Schema

De structuur van het XML-document kunt u inmiddels wel dromen: het element <books> bevat een of meer elementen van het type <book>, en het element <book> bevat elementen van het type <title>, <author>, <price> en <isbn>. Bij de prijs wordt de valutasoort als attribuut meegegeven. Om dit in een XML-Schema te definiëren gaat als volgt: Allereerst onderscheiden we het root-element <books>. Dit element is van complexe aard. Daarmee wordt bedoeld dat het een element is die andere elementen bevat. De definitie van het element <books> ziet er zó uit:

```
<xsd:element name="books">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="book" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Lees dit als: <books> is een complex element. Het bestaat uit een serie (sequence) van elementen die gedefinieerd wordt door het element met de naam <book>. Dat element wordt elders in dit document gedefinieerd, en hier wordt alleen een referentie ernaar ge-legd. Er is minimaal één <book> aanwezig, en er is geen limiet aan het maximum aantal ervan. Het voorvoegsel xsd komt van de XMLSchema-namespace die straks uitgelegd wordt.

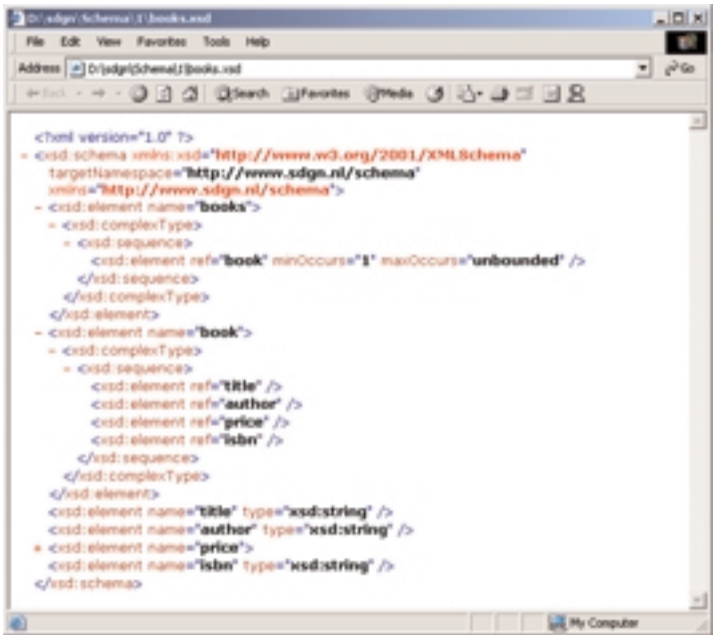
Het element <book> wordt net zo gedefinieerd: Het is een complex element dat bestaat uit een serie van de titel, de auteur, de prijs en het isbn. Bij de elementen (die refereren naar de elementen) <title>, <author>, <price> en <isbn> wordt niets gezegd over het aantal keer dat ze binnen een <book> voorkomen. De default is namelijk één keer, niet meer en niet minder. Dat hoeft niet apart vermeld te worden.

```
<xsd:element name="book">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="title"/>
      <xsd:element ref="author"/>
      <xsd:element ref="price"/>
      <xsd:element ref="isbn"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Als laatste worden binnen het XML-document de vier elementen <title>, <author>, <price> en <isbn> gedefinieerd. Zie figuur 5. Drie ervan zijn snel klaar: ze hebben een naam een een type-aanduiding, meer niet. In tegenstelling tot een DTD kunnen we hier met bekende datatypes als de string, de boolean, de dateTime en diverse numerieke datatypes werken. Let op: sommige numerieke getallen zijn anders dan velen gewend zijn:

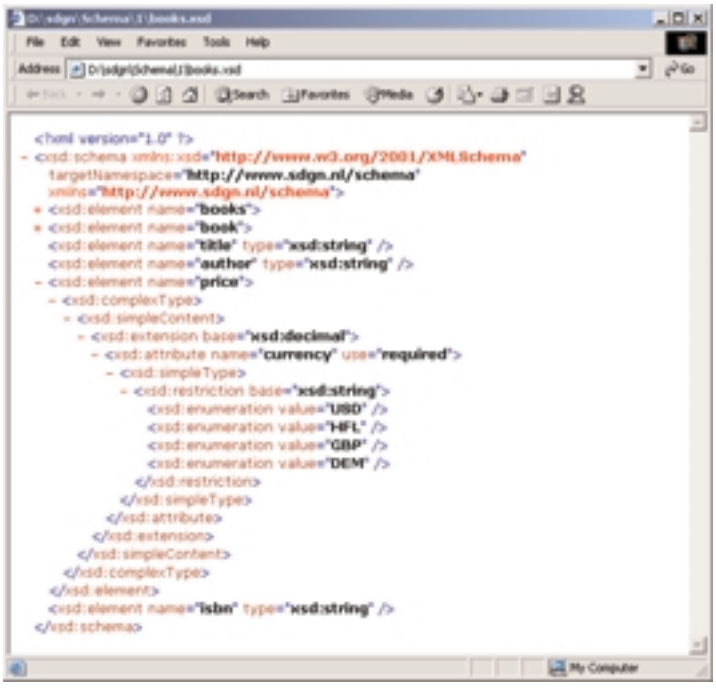
- de short = -32768 tot +32767 (in VBA de Integer)
- de int = -2147483648 tot +2147483647 (in VBA de Long Integer)
- de long = -9223372036854775808 tot +9223372036854775807 (onbekend in VBA, in SQL2000 de BigInt).

```
<xsd:element name="title" type="xsd:string"/>
<xsd:element name="author" type="xsd:string"/>
<xsd:element name="isbn" type="xsd:string"/>
```



Figuur 6: de grote lijn van het XML-Schema.

Deze elementen zijn elk in één tag gedefinieerd: let op de sluitende forward slash! Het vierde element, de prijs, heeft nog een extra toevoeging: het attribuut dat de munteenheid bepaalt. Dat wordt als volgt uiteengezet. Het element <price> is een complexType met een eenvoudige inhoud (simpleContent). De waarde van het element zelf is van het type decimal (base= "xsd:decimal"). Vervolgens heeft het element ook een attribuut met de naam "currency". Dat attribuut is verplicht (use="required"). Het attribuut heeft eveneens een eenvoudige inhoud en is beperkt (restriction) tot het type String (base="xsd:string") met keuze uit de volgende opsomming (enumeration): USD, HFL, GBP of DEM. Andere waardes worden niet toegestaan. In het totaal ziet dat eruit zoals in figuur 7 is getoond.



Figuur 7: het element <price> met attribuut.

Contract

Zoals hierboven getoond wordt met een XML-Schema wordt de structuur van een XML-document vastgelegd. Het is het data model. Maar het is meer dan dat. Het is ook een contract tussen twee partijen. Het legt namelijk vast wat de structuur is van gegevens die uitgewisseld worden. Beide partijen beschikken over een kopie van het XML-Schema. Daarenboven bevat een XML-Schema veel meta-data (informatie over de data), zoals de relatie tussen verschillende elementen of het bereik van geldige waardes voor de data. Op die manier is een XML-Schema een rijke bron van informatie en afspraken.

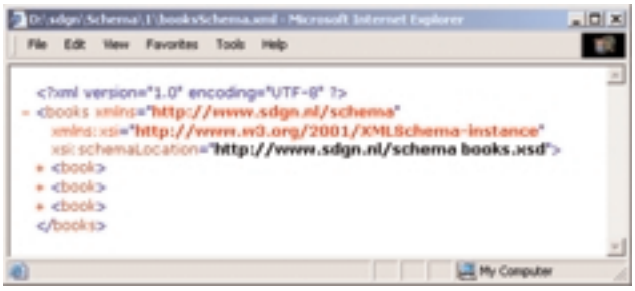
Het legt namelijk vast wat de structuur is van gegevens die uitgewisseld worden

De namespaces (2)

Het XML-document is bekend en het XML-Schema is af. De elementen zijn allemaal gedefinieerd en van beperkingen voorzien. Hoe nu verder? Welnu, hierbij komen verschillende namespaces aan te pas om het XML-document aan het XML-Schema te koppelen. Als eerste het XML-Schema. In het XML-Schema books.xsd (afgebeeld in figuur 6 en 7) zijn in het root-element <xsd:schema> twee namespaces (xmlns) en een attribuut opgenomen:

- xmlns:xsd="http://www.w3.org/2001/XMLSchema". Dit is de enige echte officiële manier om de namespace van het XML-Schema zelf te refereren. Hiermee wordt vermeld dat alle elementen die het voorvoegsel "xsd" hebben, officiële Schema-elementen zijn. Het is een goed gebruik om het voorvoegsel xsd te reserveren voor deze namespace. Om het woordenboek er nog even bij te halen: dit is de grammatica die ten grondslag ligt aan het woordenboek.
- targetNamespace="http://www.sdgn.nl/schema". Dit is de namespace waartoe alle elementen behoren die in dit XML-Schema gedefinieerd worden. Het woordenboek: alle woorden (elementen) die hier gedefinieerd worden, vormen het woordenboek met de naam "http://www.sdgn.nl/schema".
- xmlns="http://www.sdgn.nl/schema". Dit is de default namespace (te herkennen aan het ontbreken van een afkorting). Dit betekent dat alle elementen zonder voorvoegsel (zoals bij ref="book") verwijzen naar deze namespace. Omdat dit dezelfde namespace is als de targetNamespace, betekent het dat die elementen dus in dit document voorkomen! Het woordenboek: bij een verwijzing naar "book" moet gezocht worden naar het woord "book" dat ergens anders in dit woordenboek gedefinieerd is. Op die plek is te vinden wat het betekent.

In het XML-document zijn in het root-element producten twee namespace-verwijzingen opgenomen en een verwijzing naar het XML-Schema opgenomen. De openingstag van het root-element is afgebeeld in figuur 8. In dit voorbeeld is de eerste namespace de default namespace. Tenzij anders vermeld zijn dus alle elementen die in dit XML-document voorkomen gedefinieerd in de namespace xmlns="http://www.sdgn.nl/schema". Deze namespace is hierboven gedefinieerd in het bestand books.xsd. Deze is te vinden op een bepaalde plaats, namelijk in dezelfde directory als het XML-document. Om dat bestand aan te kunnen wijzen is de verwijzing "schemaLocation" van de XMLSchema-instance namespace nodig. Deze namespace wordt in de tweede regel gedefinieerd, met de afkorting "xsi". Op de derde regel kan dan de verwijzing naar het bestand books.xsd staan. In die regel wordt gezegd: "de definitie van het Schema dat http://www.sdgn.nl/schema genoemd wordt, is te vinden in het bestand books.xsd". Het bestand "books.xsd" wordt dus aangewezen als de definitie (of als het woordenboek) van de namespace http://www.sdgn.nl/schema. Indien books.xsd in een andere directory staat, wordt het pad erbij gegeven.



Figuur 8: het root-element van het XML-document booksSchema.xml.

MS Development Tip

Waarom is VB.NET niet 100% compatibel met VB 6.0?

De belangrijkste reden is dat VB.NET probleemloos moest kunnen samenwerken met de CLR en er in twee richtingen moet kunnen worden overerft tussen VB.NET en de andere .NET talen. Daarom moest VB gebruik maken van dezelfde data-typen, classes en interfaces als elk ander .NET taal en dat was niet te verenigen met de volledige oude syntax en semantiek van VB 6.0. Een bijkomend voordeel is dat VB nu meegroeit met het nieuwe .NET platform en dus geen taal op zichzelf, maar onderdeel van een belangrijke, richtinggevende strategie is geworden.

Conclusies

Zowel DTD's als XML-schema's hebben voordelen. De voornaamste reden om DTD's te leren lezen (en te schrijven) is, dat er momenteel veel in omloop zijn. Het is een bekende techniek die breed ondersteund wordt. Het hoort bij SGML en dat blijft nog lang bestaan.
XML-schema's werden tot voor kort nog onvoldoende ondersteund. Microsoft is eerst met een eigen versie (xdr =XML Data Reduced) aan de slag gegaan om verder te gaan. Sla maar eens een ADO-recordset op als XML-document. Daar wordt een intern XDR-Schema gemaakt, dat incompatible is met de officiële XML-Schema recommendation. Sedert de officiële recommendatie van het XML-Schema in mei dit jaar is het echter veel beter geworden. Diverse tools ondersteunen het XML-Schema en met het XML-Schema kunnen de bekende datatypes uit de programmeertalen direct gebruikt worden. Dit alles maakt het werken met XML-Schema en het integreren met applicaties een stuk makkelijker. Voor iedereen die nu begint met het valideren van XML-documenten raad ik aan om dat zoveel mogelijk met XML-Schema te doen.

Testen of het werkt

Om te testen of alles goed werkt is een download beschikbaar. Hierin zijn de voorbeelden van dit artikel opgenomen, samen met een VB-applicatie die XML-documenten valideert. Als een XML-document een DTD of XML-Schema heeft en de validatie is in orde, dan wordt het XML-document in een messagebox getoond. Ook zijn er een paar testbestanden bijgevoegd die opzettelijk een fout bevatten: één mist de titel bij een boek, de ander heeft de prijs in Euro's vermeld. De aanwezigheid van de msxml4 parser is wel een voorwaarde voor het goed werken van de applicatie.

Installeren MSXML4

Als u tussen april en juli dit jaar de MSXML4 parser van Microsoft heeft geïnstalleerd en u wilt de nieuwe versie van juli installeren, dan moet u even op het volgende letten. Met het de-registreren van de april-versie van MSXML4 en het her-registreren van de MSXML3-parser bent u er nog niet, ook al beweert Microsoft op haar website van wel. Bij het runnen van de setup krijgt u de keuze om MSXML4 te de-installeren. Als u dat doet en pas daarna kiest voor de installatie van de nieuwste MSXML4, loopt alles als een zonnetje.

Bram Laumans is trainer bij XCESS expertise center b.v. in Leusden Hij heeft ruim drie jaar erva-ring in het programmeren met Access, VB(A), SQL Server, Visual InterDev en XML. Zijn voornaamste werkzaamheid is het verzorgen van trainingen op deze onderwerpen. Hij is te bereiken via opleidingen@xcess.nl.

tools

- ✔ C#
- ✔ .net
- ✔ Visual Studio
- ✔ Delphi

auteur

Peter van Ooijen

C#, ook mooi

Een Delphi ontwikkelaar zoekt events

Inleiding

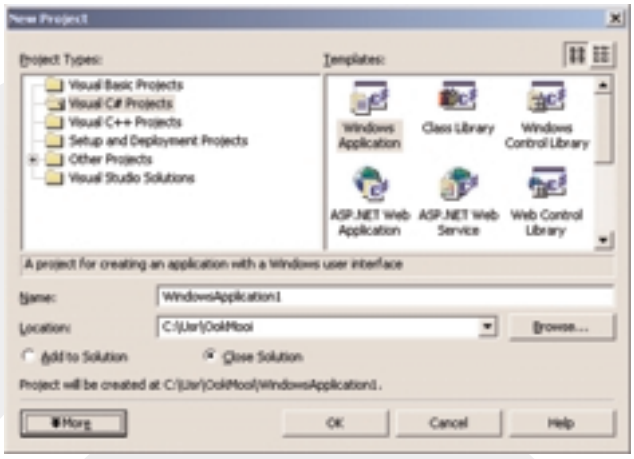
U kent mij waarschijnlijk als iemand die alleen maar verhalen over Delphi en COM schrijft, de titel van dit artikel is misschien dan ook even schrikken. Laat ik U geruststellen, er is wat mij betreft niets mis met Delphi. Maar in de eerste plaats ben ik een erg nieuwsgierig mens. Je wordt overspoeld met aandacht voor het Microsoft dot.net platform. Op een CttP kreeg ik zomaar een mooi boek over C#, de nieuwe programeertaal in dot.net, en de beta versies van Visual Studio dot.net kreeg ik ook al in veelvoud te pakken. Ik kon het dus niet laten er eens echt mee aan de gang te gaan.

Maar ik had meer redenen om er naar te willen kijken. In de eerste plaats kan je, als je applicaties voor Windows maakt, niet om dot.net heen, al zou je willen. Het dot.net framework wordt de nieuwe windows API, je hoeft alleen nog maar te wachten op het moment dat de klassieke API achter de schermen van het framework verwijnt. Niet dat ik dat nou zo erg zou vinden.
Mijn tweede grote reden om naar dot.net, en met name C#, te willen kijken zit hem in de makers. De grote man achter C# is Anders Hejlsberg. In een grijs verleden heeft hij Turbo Pascal uit de grond gestampt. En later Delphi. Anders ging bij Microsoft werken, maakte als vingeroefening nog even J++ en nou heeft hij met C# iets gemaakt dat de moeite van het bekijken zeker waard is. Dr. Dobb's journal kende hem niet voor niets het "Excellence in programming award" toe.

Al lezend in "a Programmers intro to C#" voelde ik me al snel thuis. Het boek richt zich op programmeurs met een C++ achtergrond maar ik herken er eerder Delphi dan C++ in. In #64 van SDGN-magazine schreef ik een artikel "the Beauty of language", over een aantal erg mooie mogenlijkheden van Object Pascal. In dit artikel wil ik dat verhaal nog een keertje overdoen, maar nu bekeken vanuit C#. Al snel zullen we een aantal zaken daarin tegenkomen die zeker niet voor Object Pascal onderdoen.

De praktijk

Het installeren van de Visual Studio beta is een hele klus. De poging beta 1 aan de gang te krijgen leidde indertijd tot een complete herinstallatie van Windows. Deze keer ging het me beter af. Het is zaak dat je van Windows2000 (welke andere ?) het laatste service pack hebt geïnstalleerd en dat je lokale IIS (Internet Information Server) goed draait. Dan hoef je alleen maar te wachten, toe te kijken, een paar keer opnieuw op te starten en uiteindelijk draait het en je kan VS starten. Ik begin met een C# windows application.



Figuur 1

Er verschijnt een lege form. Vanuit de toolbox zet ik er een button op, dubbelklik de button en ben in de code editor. Nou wordt het even zoeken voor ik precies weet hoe ik moet gaan programmeren. Ik wil hier niet ingaan op de syntax van C#, die staat in het boek en ik ga er van uit dat die goed te lezen is. C# is volledig object-georienteerd, zo is er geen globale procedure messagebox, je gebruikt het object MessageBox, diens methode Show toont de melding. Ik moest er nog het meest aan wennen dat C# case-sensitive is. Gelukkig is de helpfile erg uitgebreid.

C# kijkend door een Delphi bril

```
private void button1_Click(object sender, System.EventArgs e)
                {
                        MessageBox.Show("Hello C# world");
                }
```

Ik draai het programma, klik de button en het werkt ! Ik ben de wereld van C# binnen.

De structuur van de applicatie

Laten we de source eens gaan bekijken. Dit doe ik kijkend door een Delphi bril, wie niet met Delphi bekend is moet mijn vergelijkingen maar gebruiken om eens naar Delphi te kijken door een C# bril. Er zijn twee .cs bestanden, één-tje met assemblyinfo en Form1.cs. Deze laatste bevat de code voor het form. De module begint met een lijst classes van het dot.net framework die gebruikt worden

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

Dit lijkt veel op de uses clause in Delphi, wat de VCL is voor Delphi is het dot.net framework voor C#. In het dot.net framework zit wel meer structuur, het is onderverdeeld in zogeheten namespaces, je hebt de namespace System, daarbinnen de namespace Windows en daarbinnen weer de namespace Forms. Ook een applicatie is een namespace, na de using statements zie je

```
namespace WindowsApplication1
```

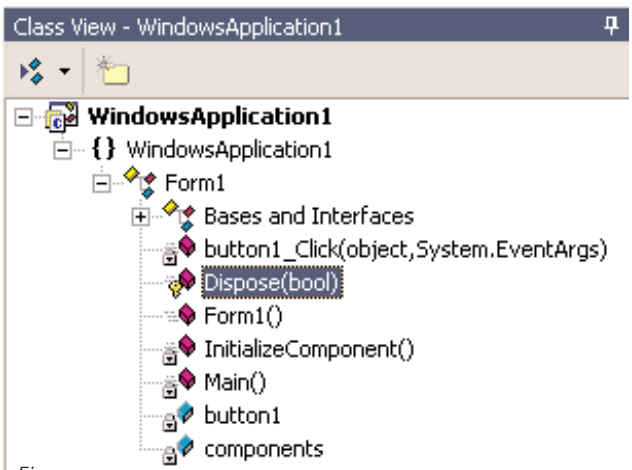
Binnen deze namespace is de form als klasse gedeclareerd

```
public class Form1 : System.Windows.Forms.Form
        {
                static void Main()
                {
                        Application.Run(new Form1());
                }

        }
```

Zoals een Delphi form afstamt van tForm stamt een C# form af van de dot.net klasse System.Windows.Forms.Form. De main methode van het form is het entry punt van de applicatie. De inhoud lijkt veel op de code die Delphi genereert in een dpr, ook hier is er een application object met een run methode.
Goed, we weten hoe een C# programma begint en zijn openings form aanmaakt, in de volgende stap moeten we eens gaan kijken wat er in die form gebeurt.

De opbouw van een form

De classviewer in VS geeft een duidelijk overzicht van de properties en methodes van de Form1 class (figuur 2). Dat zijn er helemaal niet zo veel. Als properties zien we button1 en de components collectie. Form1, de methode met dezelfde naam als de klasse, is de constructor. Dot.net objecten hebben geen destructor, zij worden beheerd door de garbage collector. De methode dispose wordt uitgevoerd voordat het object door de garbage collector opgeruimd wordt. De basisklasse in Delphi, tObject, kent een vergelijkbare BeforeDestruction methode.



Figuur 2

De methode main zijn we net al tegengekomen, het is het entry point van de applicatie. De private methode InitializeComponent wordt aangeroepen in de constructor, hierop kom ik straks terug.
Blijft de private methode button1.Click. Dit is de code die ik net zelf had geschreven. De code wordt uitgevoerd als ik op de button klik, maar het is een methode van de form klasse en niet van de button klasse. Precies dit zelfde verschijnsel zie je in een Delphi form. Om te zien hoe deze code gekoppeld wordt aan de button gaan we een stukje van de InitializeComponent mehode eens bekijken.

```
private void InitializeComponent()
{
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(24, 16);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(88, 24);
            this.button1.TabIndex = 0;
            this.button1.Text = "button1";
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            ...
            //
}
```

De code is gegenereerd door de forms designer. Wijzigingen in de forms designer vind je hier meteen terug en als je hier iets wijzigt zie je dat weer in de forms designer terug. Een nieuwe button wordt aangemaakt en de properties worden ingesteld. In de laatste regel van dit stukje wordt het button.click event gekoppeld aan de button1_click methode.

Je hebt niet per sé Pascal nodig om een Beauty van een language te maken

Je ziet hier dat ook in C#, net als in Delphi, de code die uitgevoerd wordt bij het afgaan van het event een runtime instelbare property van een component is. In C# zijn hier een stel hele mooie mogelijkheden, in de rest van dit verhaal zal ik hier verder op ingaan.

De eventhandler

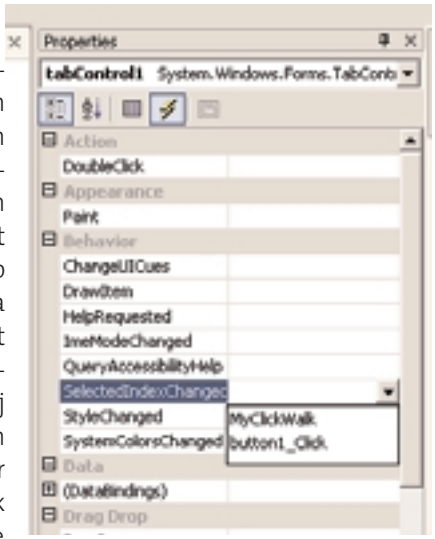
De Eventhandler klasse zit heel diep in het dot.net framework ingebakken, hij is lid van de System namespace en stamt rechtsreeks af van de delegate klasse. Delegates worden gebruikt om een event (zoals bijvoorbeeld het klikken op een button) aan een methode te koppelen. De constructor van een delegate krijgt de uit te voeren methode mee. Het aantal en het type van de parameters van een methode wordt de signature van een methode genoemd. Voor elke andere signature zal er een andere constructor zijn.
Het type van button1.click is een event met als signature (`object sender, System.EventArgs e`). De sender is degene die het event veroorzaakt, in dit geval is het de button

waarop geklikt wordt. De tweede parameter bevat eventuele argumenten die aan de eventhandler kunnen doorgegeven. In de System namespace is de EventArgs klasse gedeclareerd, deze bevat zelf geen data maar kan als basisklasse worden gebruikt als je wel data door wilt geven. Gegeven deze kennis wordt het leuk om zelf een algemene eventhandler te schrijven

```
private void MyClickWalk(object sender, System.EventArgs e)
{
        Control sent;
        sent = sender as Control;
        if (sent != null)
                sent.Left+= 10;
}
```

In de handler declareer ik een lokale variabele van het type control, de basisklasse voor alle windows controls. Ook C# kent, net als Object Pascal, de as operator. Die typecast de binnengekomen sender naar een Control. Als de binnengekomen sender geen Control is dan wordt sent null, maar als die wel een Control is dan kan hem nu als dusdanig behandelen. Dat doe ik en ik laat het control naar rechts wandelen.

Deze eventhandler ken ik toe aan het click-event van de button. Vervolgens zet ik een tabcontrol met twee pagina's op mijn form en ga daar een event uitkiezen dat uitgevoerd wordt bij het wisselen van tab-pagina. Hier zie ik MyClickWalk in de eventkeuze lijst staan en ik kies hem. Als je nu in de InitializeComponenet methode kijkt kom je MyClickWalk twee keer tegen.



Figuur 3

```
this.button1.Click += new
System.EventHandler(this.MyClickWalk);
//
// tabControl1
..
this.tabControl1.SelectedIndexChanged += new
System.EventHandler(this.MyClickWalk);
```

Als je het programma nu draait dan zal het klikken van de button en het wisselen van pagina in de tabcontrol resulteren in het weglopen van de controls. Mijn MyClickWalk methode wordt dus door twee controls als eventhandler gebruikt.

Figuur 4

Eventhandlers aan en uit zetten

We hebben nu gezien hoe een eventhandler in code, bij het opbouwen van het form, gezet wordt. Dat zou ook op een ander moment in runtime moeten kunnen. Waar ik tot nu toe ook overheen been gelopen is dat het zetten gebeurt met de += operator en niet gewoon met de = operator. Dat suggereert dat de eventhandeler ergens aan toe wordt gevoegd.

Om dit te onderzoeken geef ik de knop weer een eigen eigen eventhandler. Deze laat ik een "." toevoegen aan de caption van de knop, dan zie je snel dat er iets gebeurt. Aan de tabcontrol zie je zonder verdere moeite dat de pagina veranderd is. De MyClickWalk method haal ik designtime bij beide controls weg. Op de form zet ik naast elk control een checkbox, ik wil dat MyClickWalk alleen wordt uitgevoerd als de checkbox aan staat. In de CheckedChange event van de checkbox stop ik nu de volgende code.

```
private void checkBox1_CheckedChanged(object
sender, System.EventArgs e)
{
  if (checkBox1.Checked)
    button1.Click+= new
    System.EventHandler(MyClickWalk);
  else
    button1.Click-= new
    System.EventHandler(MyClickWalk);
}
```

Als de checkbox aan is gezet dan voeg ik met += operator mijn eventhandler toe aan de Button1.Click, op precies dezelfde wijze als net in InitializeComponent. Als de checkbox uit staat dan probeer ik hem met de -= operator weer te ontkoppelen. In de CheckedChanged event van de andere checkbox gebeurt precies hetzelfde met de tabcontrol.

Figuur 5

En dit werkt. Het wandelen van de controls kan worden aan en uitgezet door de checkbox. En de tekst van de button wordt ook nog steeds aangepast.

Een C# Eventhandler is een zogeheten multicast delegate, dat wil zeggen dat hij de notificatie van het event naar een hele lijst van geïntereseerden kan sturen. Er zijn een heleboel procedures en properties om met deze lijst te werken, het mooie van C# is dat je deze lijst ook heel eenvoudig kan manipuleren met de += en de -= operator.

EventArgs

Een volgende stap wordt het event ook wat data door te laten geven. In C# is alles een object, de data komen dus in een nieuwe klasse. Een klasse MyEventArgs die afstamt van EventArgs.

```
public class MyEventArgs : System.EventArgs
{
        public MyEventArgs(string aMessage)
        {
                aString = aMessage;
        }
        public string aString;
}
```

De data is een publieke string aString, in de constructor geef ik de inhoud mee.
System.Eventhandler is een delegate die werkt met de signature die door control events gebruikt wordt (`object sender, System.EventArgs e`). Om met mijn eventargs te gaan werken moet ik een bijbehorende delegate declareren

```
public delegate void MyDelegate(object sender, MyEventArgs e);
```

Hier zie je dat een delegate zoveel is als de declaratie van een signature. Het is vergelijkbaar met de manier dat in Delphi een tNotifyEvent is gedeclareerd als een "procedure of object". Delegate is een keyword van C# zelf, het is diep in de taal ingebouwd.
De delegate is gedeclareerd en heb ik een methode nodig die deze signature heeft en iets met de aangeboden eventargs gaat doen. Op de form plaats ik een listbox en ik schrijf de methode

```
private void LogString(object sender, MyEventArgs e)
{
        listBox1.Items.Add(e.aString);
}
```

De methode haalt de string uit het MyEventsArgs object en voegt die toe aan de listbox.
Nu heb ik nog een klasse nodig die MyEventArgs en de delegate bij elkaar brengt. Ik noem hem MyEvent. Hij heeft een punt waar de notificeerder hem aan moet kunnen roepen. De notificeerder is de button of het tabcontrol in wiens notification lijst hij terecht gaat komen. Dit betekent dat hij een methode moet hebben met de signature van System.EventHandler. Hij heeft daarnaast een MyDelegate delegate nodig die het echte werk moet gaan doen. En hij moet een object van MyEventArgs aan kunnen maken.

```
public class MyEvent
{
private string MyMessage;
private MyDelegate Worker;
public MyEvent(string aMessage, MyDelegate YouDoIt)
{
        MyMessage = aMessage;
        Worker = YouDoIt;
}
public void DoIt(object sender, System.EventArgs e)
{
        Worker(sender, new MyEventArgs(MyMessage));
}
}
```

De data voor MyEventArgs wordt opgeslagen in MyMessage. Worker is de delegate en DoIt is het punt van binnenkomst. Je ziet dat de DoIt methode een nieuw MyEventArgs object aanmaakt en dit doorstuurt naar de delegate Worker.
Tot slot moet ik nu de mogenlijkheid hebben om MyEvent toe te voegen aan de eventlist van een control. Naast de checkbox zet ik een button, een klik op de button voegt mijn event toe aan de eventlist

```
private void button2_Click(object sender, System.EventArgs e)
{
        MyEvent ne;
        ne = new MyEvent(button1.Text, new MyDelegate(LogString));
        button1.Click+= new System.EventHandler(ne.DoIt);
}
```

Lokaal maak ik een nieuw MyEvent object aan. De constructor krijgt de huidige tekst op de button mee, bij elke klik werd die tekst een "." langer. Die string moet in de listbox terecht moet komen. Als tweede parameter geef ik een nieuw delegate object mee van het type Mydelegate. De constructor daarvan krijgt als parameter de LogString methode mee. LogString heeft de signature van MyDelegate. In de implementatie van LogString werd de string uit MyEventArgs aan de listbox toegevoegd.

Voor de tabcontrol maak ik een button die de naam van de huidige pagina als string neemt. Als je nu de list button klikt dan wordt de huidige tekst van de knop bij iedere klik aan de listbox toegevoegd. Klik je nog een keer op de list button dan worden twee teksten toegevoegd. Er wordt dan namelijk weer een nieuwe eventhandler aan de button toegevoegd. En die lijst kan je eindeloos laten groeten.

Figuur 6

In dit geval voeg ik alleen maar nieuwe events toe aan de lijst. De nieuwe events maak ik lokaal aan in de methode, voeg ze toe aan de eventhandler en verder vertrouw ik ze toe aan de zorgen van de garbage collector. Om een event weer te verwijderen moet ik wel weten wel event object daarbij hoort. Dat zou ik zelf kennen gaan bijhouden of ik moet verder duiken in de vele properties en methodes van een delegate. En dat wordt weer een verhaal apart.

C# is vergeven van de events, er hoeft maar iets te gebeuren en er wordt een event afgevuurd

Tot slot

We hebben gezien dat het mechanisme van eventhandling in C# erg lijkt op dat in Delphi. Het grote verschil is dat in C# events multicasts zijn, het afgaan van het event kan aan een hele lijst van eventhandlers worden doorgegeven. In Delphi zijn events singlecast.
C# is vergeven van de events, er hoeft maar iets te gebeuren en er wordt een event afgevuurd. In dit verhaal heb ik me beperkt met de klassieke events in windows

controls en daar zul je al hele fraaie tegenkomen. Zoals een events die afgaan als properties van de control van waarde veranderen. Je zult ook events vinden rond databases, webpagina's, printen, het afhandelen van exceptions etcetera, etcetera.

In dit verhaal ben ik niet veel verder gekomen dan het "krabben aan de buitenkant" van de events. Maar ik vind het nu al erg mooi. Anders Hejlsberg heeft goed werk gedaan en laten zien dat je niet per sé Pascal nodig hebt om een Beauty van een language te maken.

Peter van Ooijen is een nieuwsgierige ontwikkelaar. In 1986 starte hij Gekko Software (www.Gekko-Software.nl). In de loop der jaren heeft Peter vele talen en tools voorbij zien komen en overal iets van mee willen nemen. Zijn grootste interesse ligt in de communicatie tussen software componenten en in mooie talen om die te maken. Na een lange tijd bezig geweest te zijn met Delphi en COM is hij nu bezig zijn blikveld verder te verruimen en daarover in SDGN magazine te berichten. Behalve in SDGN Magazine zijn er ook op zijn website meerdere artikelen te vinden.

FoxPro Tip

Problemen met sys(2335,0)

Het SYS(2335) commando wordt gebruikt om in te stellen hoe een COM object van Visual FoxPro zich moet gedragen wanneer er een melding van VFP verschijnt terwijl COM objecten geen user-interface ondersteunen. Dergelijke commando's hangen je applicatie op. Met SYS(2335) kan worden ingesteld dat VFP 'Unattended' werkt, eventuele meldingen triggeren op dat moment onmiddellijk de 'On Error' procedure.

Wanneer SYS(2335,0) gebruikt wordt in een 'normale' VFP omgeving ontstaat door dit commando een C000005 error zodra het commando READ EVENTS gegeven wordt. Je moet er maar achter komen...

tools

- ✔ SOAP
- ✔ Visual FoxPro 7
- ✔ ASP

auteur

Remi Caron

Web Services: where to begin?

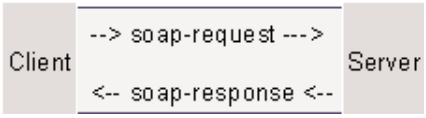
Web-services are the talk of the town since a year or so. But what is it? Can I do this today or do I need to develop in C# or any other .NET language? What do I need? Where do I start? Next to these practical questions are the commercial questions like how to market and what is a useful service to build?

In a series of articles on this new phenomenon hope to address all of these topics. In this first of these articles I start at the very bottom of it all. Samples used in the articles contain VBScript, Jscript, ASP, COM+ and Visual FoxPro code. As in all my articles the technologies used here can be 'copied' to all other programming languages.

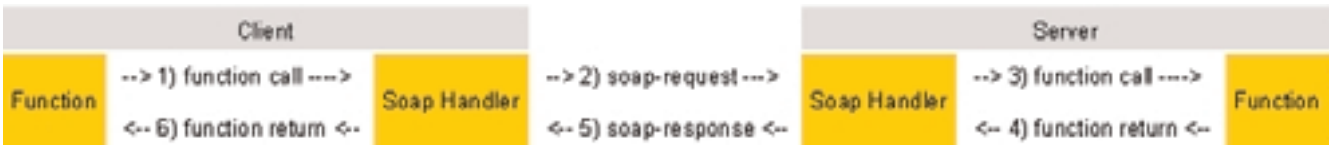
The first thing you'll notice when you start to explore this new buzzword, is that it is not an out of the box product. It is a definition for fulfilling a service by means of a URL. Having said this the practical questions asked earlier don't become any easier to answer.

What is a Web Service?

A Web Service is a website that exposes services without a user interface. Data exchange between the server and the client is XML-based. You can invoke a web service by sending a SOAP-request to the server, and the answer you'll receive will be a SOAP-response. This is shown in the next figure.



Both on the client and the server-side the SOAP-messages will be transformed by a SOAP-handler into processable function calls. The SOAP-handler can be an ASP or JSP page, a CGI script or an ISAPI filter. These are just a few of the options available.



In (very) brief the figure above describes what a web-service is, how it works and how to invoke it from your own development projects. (Notice that these projects don't need to be web based projects). I mentioned SOAP and since there has been a lot of talk about SOAP in our magazine and many others, I will address the matter briefly in the context of this article.

What is SOAP?

SOAP stands for Simple **O**bject **A**ccess **P**rotocol. SOAP is a standard defined by the W3C to invoke a service that is hosted on a website. SOAP is based on XML. A service request is issued by means of a SOAP-request. The answers to this request will be received as a SOAP-response. A SOAP-message can be sent via HTTP and HTTPS and several other protocols. A SOAP message contains 3 sections:

1. **The envelope**
   The envelope is a required part of the message.
2. **The header**
   The header is optional.
   When a header is used, it will always be the first element in the envelope. The header is used to provide extra information in the message. This information is not required for processing the message.
3. **The body**
   The body contains all the information needed to process the request.

If shown in a figure, the skeleton of a SOAP-message looks like this: ▼

```
<SOAP:Envelope                                    ENVELOPE
 xmlns:SOAP=
"http://schemas.xmlsoap.org/soap/envelope"        ENVELOPE
 SOAP:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/">      ENVELOPE
   <SOAP:Header>                                  HEADER
      [info in Xml format]
   </SOAP:Header>                                 HEADER
   <SOAP:Body>                                    BODY
      [info in Xml format]
   </SOAP:Body>                                   BODY
</SOAP:Envelope>                                  ENVELOPE
```

The sample below shows SOAP communication between a server and a client, where the server exposes a calculator as a service. The sample here invokes the add method of the service.

First the client sends the following SOAP-request:

SOAP-Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAPSDK1:Add
      xmlns:SOAPSDK1="http://tempuri.org/message/">
      <A>10</A>
      <B>90</B>
    </SOAPSDK1:Add>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Then it could receive the following SOAP-response:

SOAP Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV=
   "http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
     <SOAPSDK1:AddResponse
       xmlns:SOAPSDK1="http://tempuri.org/message/">
       <Result>100</Result>
     </SOAPSDK1:AddResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

What is next

Now that we've seen (a little bit of) what a web-service is and how to invoke it, let's take a closer look at what is running on the server to answer an incoming request. To keep it simple I've created a COM+ component in VFP 7, that doesn't do very much, but it is the first step to get your mind around the whole idea of a web-service.

```
DEFINE CLASS ContentMgr AS Session OLEPUBLIC

  FUNCTION EchoString (tcString AS String) AS String

    *-------------------------------------------------
    * Abstract....: Echoes string (Soap Test Method)
    * Parameters..: tcString
    * Returns.....: tcString
    *-------------------------------------------------
      RETURN UPPER(tcString)
  ENDPROC

ENDDEFINE
```

Basically, this routine does nothing very special: it retrieves a string and passes it back (pretty useful <bg>). But the goal is to see it work now, and making it more difficult and useful is the next step in this process. Compile this into a DLL and install it on the server.

WSDL and SOAP-listeners

Next we need a WSDL file and a SOAP-listener, which can be either an ISAPI filter or an ASP page (or one of its equivalents like JSP, PHP, ...). For this article I've installed the MS-Soap toolkit (downloadable from the Microsoft site) on the server, and this automatically installs an ISAPI filter for you on the web server. (Next time we'll take a look at the ASP page option for handling requests.) The SOAP-listener is the SOAP-handler on the server side, as explained above.

WSDL stands for Web Service Description Language. It is used to describe what methods a web service has to offer and what kind of parameters it expects. If you want to provide an web service, you must also provide an WSDL file. You can type it in any editor, but fortunately the WSDL file can be also generated with a tool that ships with the MS-SOAP toolkit called the WSDL Generator (on the Programs menu).

The WSDL snippet below is used to make the sample in this article work.

```
<definitions name ='ContentMgr'
  targetNamespace='http://bizzview.com/wsdl/'
  xmlns:wsdlns='http://bizzview.com/wsdl/'
  xmlns:typens='http://bizzview.com/type/'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:stk='http://schemas.microsoft.com/soap-toolkit/
    wsdl-extension'
  xmlns='http://schemas.xmlsoap.org/wsdl/'>
  <types>
    <schema targetNamespace='http://bizzview.com/type/'
      xmlns='http://www.w3.org/2001/XMLSchema'
      xmlns:SOAP-ENC=
        'http://schemas.xmlsoap.org/soap/encoding/'
      xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'>
    </schema>
  </types>
  <message name='ContentMgr.EchoString'>
    <part name='cString' type='xsd:string'/>
  </message>
  <message name='ContentMgr.EchoStringResponse'>
    <part name='Result' type='xsd:string'/>
  </message>
  <portType name='ContentMgrSoapPort'>
    <operation name='EchoString'
      parameterOrder='cString'>
      <input message='wsdlns:ContentMgr.EchoString'/>
      <output message=
        'wsdlns:ContentMgr.EchoStringResponse'/>
    </operation>
  </portType>
  <binding name='ContentMgrSoapBinding'
    type='wsdlns:ContentMgrSoapPort' >
    <stk:binding preferredEncoding='UTF-8'/>
    <soap:binding style='rpc'
      transport='http://schemas.xmlsoap.org/soap/http'/>
    <operation name='EchoString' >
      <soap:operation soapAction=
        'http://bizzview.com/action/
          ContentMgr.EchoString'/>
      <input>
        <soap:body use='encoded'
          namespace='http://bizzview.com/message/'
        encodingStyle=
          'http://schemas.xmlsoap.org/soap/encoding/'/>
      </input>
      <output>
        <soap:body use='encoded'
          namespace='http://bizzview.com/message/'
        encodingStyle=
          'http://schemas.xmlsoap.org/soap/encoding/'/>
      </output>
    </operation>
  </binding>
  <service name='ContentMgr' >
    <port name='ContentMgrSoapPort'
      binding='wsdlns:ContentMgrSoapBinding' >
      <soap:address
        location='http://test.bizzview.com/_contentmgr/
          ContentMgr.wsdl'/>
    </port>
  </service>
</definitions>
```

After seeing this you might get discouraged but don't be: just use the WSDL generator. It'll do all the work for you. But if you analyze this closely, you'll see there is no real rocket science in there. It 'defines' a web service that is running somewhere inside the www.bizzview.com website, that is called ContentMgr and provides the same EchoString functionality as described above.

If your using the MS-SOAP toolkit, there is an MS specific file that needs to be in place as well. It is called WSML, which stands for Web Services Meta Language (WSML). A WSML file provides information that maps the operations of a service (as described in the WSDL file) to specific methods in the COM object. The WSML file determines which COM object to load to service the request for each operation. The use of a WSML file is specific to SOAP Toolkit 2.0. It will also be generated for you by the MS-SOAP toolkit, and for our sample, it will look like this:

```
<servicemapping name='ContentMgr'>
  <service name='ContentMgr'>
    <using PROGID='ContentMgrV2.Viewer'
      cachable='0' ID='ContentMgrObject' />
 <port name='ContentMgrSoapPort'>
    <operation name='EchoString'>
      <execute uses='ContentMgrObject'
        method='EchoString'>
        <parameter callIndex='1'
          name='cString' elementName='cString' />
        <parameter callIndex='-1'
          name='retval' elementName='Result' />
      </execute>
    </operation>
</port>
  </service>
</servicemapping>
```

Using a web service

After having done all this it's time to start calling our first web service. It is the web service described in the before mentioned WSDL and WSML file.
To use this web service, I've worked out 4 samples:

1. Using the SOAP-toolkit
   (from your development environment)
2. Using the MSXML parser
   (from your development environment)
3. Using the SOAP-toolkit (from a webpage)
4. Using the MSXML parser (from a webpage)

Sample 1: Using the SOAP toolkit from your development environment

Notice that you need to have the MS-SOAP toolkit 2.0 installed on your machine to get this working. If you do have this installed, the (VFP) sample here will work right away: I've left the web service used in this sample on the bizzview website for you to test it yourself.

```
cServer = ;
  "http://www.test.bizzview.com/"   +;
  "_contentmgr/contentmgr.wsdl"
cService= "ContentMgr"

oSoap = CREATEOBJECT("mssoap.soapclient")
oSoap.MsSoapInit(cServer, cService)

MessageBox(oSoap.echostring("Hello World"))
```

Sample 2: Using the MSXML Parser from your development environment

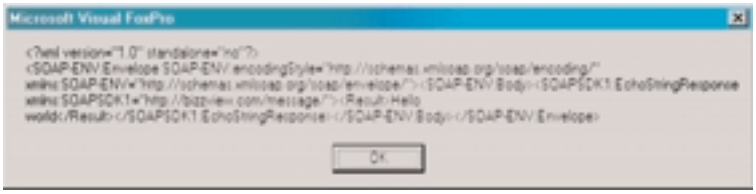
The following sample is based on the MSXML parser version 3. Running this sample will show two message boxes, one displaying what is sent to the server and the other one telling you what is received as an answer from the server. This sample will help you understand the way the client and the server communicate with each other.

```
LOCAL loHTTP, lcEnvelope, loReturn, lcUrl

** URL for the web service
lcUrl = ;
  "http://www.test.bizzview.com/" +;
  "_contentmgr/contentmgr.wsdl"

** create a placeholder for the return message
loReturn = CREATEOBJECT("Line")
loReturn.AddProperty("FullMessage", null)
loReturn.AddProperty("Message", null)
loReturn.AddProperty("lError", .F.)
loReturn.AddProperty("ErrorMessage", null)

*** Build the SOAP:Envelope
lcEnvelope = ""
lcEnvelope = lcEnvelope +;
  [<?xml version="1.0" encoding="UTF-8" ] +;
  [standalone="no" ?>]
lcEnvelope = lcEnvelope +;
  [<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=] +;
  ["http://schemas.xmlsoap.org/soap/encoding/" ] +;
  [xmlns:SOAP-ENV=] +;
  ["http://schemas.xmlsoap.org/soap/envelope/">]
lcEnvelope = lcEnvelope + [<SOAP-ENV:Body>]

lcEnvelope = lcEnvelope +
  [<SOAPSDK1:EchoString ] +;
  [xmlns:SOAPSDK1="http://bizzview.com/message/">]
lcEnvelope = lcEnvelope +;
  [<cString>Hello world</cString>]
lcEnvelope = lcEnvelope + [</SOAPSDK1:EchoString>]

lcEnvelope = lcEnvelope + [</SOAP-ENV:Body>]
lcEnvelope = lcEnvelope + [</SOAP-ENV:Envelope>]

** show the message sent to the server
MESSAGEBOX(lcEnvelope)

*** Create the XMLHttp request object
loHTTP = CREATE("Microsoft.XMLHTTP")

* Add request headers
loHTTP.Open("POST", lcURL, .F.)
loHTTP.setRequestHeader("Content-Type:", "text/xml")
loHTTP.setRequestHeader("SOAPAction:", ;
  "http://bizzview.com/action/ContentMgr.EchoString")
loHTTP.Send(lcEnvelope)

IF loHTTP.Status <> 200
  loReturn.lError = .T.
  loReturn.ErrorMessage  = ;
    TRANSFORM(loHTTP.Status) + " : " + loHTTP.StatusText
  loReturn.FullMessage  = ;
    "<RETURN>" + TRANSFORM(loHttp.status) + "</RETURN>"
  loReturn.Message   = ""
 ELSE
  loReturn.FullMessage  = loHTTP.responseXML.XML
  loReturn.Message   = loReturn.FullMessage
ENDIF

loHTTP = .NULL.

** show the message returned from the server
MESSAGEBOX(loReturn.Fullmessage)

RETURN
```

The two messageboxes displayed look like this:

Message sent:

Message received:



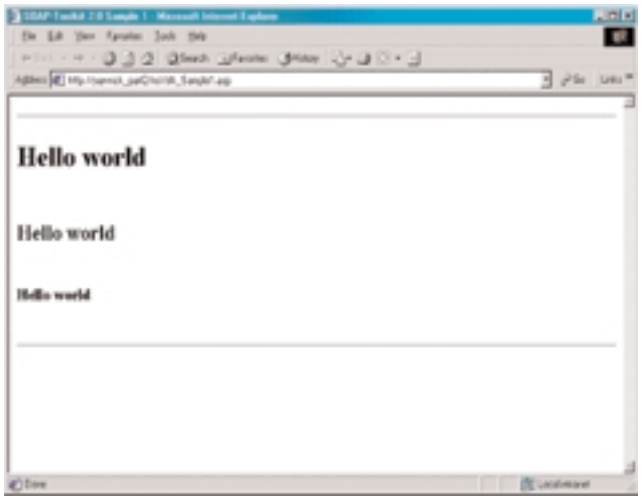
Sample 3: Using the SOAP toolkit from a webpage
Doing the same as in sample 1 but then from within an ASP page looks like this:

```
<%@ Language=VBScript %>
<HTML>
<HEAD>
  <TITLE>SOAP-Toolkit 2.0 Sample 1</TITLE>
</HEAD>
<BODY>
  <P>
    <H2>Test if the connection can be made to the server
      and a valid response is returned</H2>
  </P>
  <%
  ' Set up some vars to use later on
  const cServer = " http://www.test.bizzview.com/" &_
    "_contentmgr/contentmgr.wsdl "
  const cService= "ContentMgr"

  ' Create MSSoapToolKit Soap Client
  Set oSoapClient = _
    Server.CreateObject("MSSOAP.SoapClient")

  ' Initialiaze the soapclient by passing the name
  ' of the server and the name of the service
  oSoapClient.ClientProperty("ServerHTTPRequest") = true

  Call oSoapClient.MSSoapInit(cServer, cService)

  ' Call the test function echostring to write Hello
  Response.Write("<HR>")

  FOR lnLoop = 1 TO 3
    Response.Write("<H" & cStr(lnLoop) & ">" &_
      oSoapClient.echoString("Hello world") & "<H" &_
      cStr(lnLoop) &"></BR>")
  NEXT
  Response.Write("<HR>")
  ' Release the object
  Set oSoapclient = Nothing
  %>
</BODY>
</HTML>
```

The result in your webbrowser will look like this. The "Hello world" string is written 3 times, because of the FOR NEXT loop in the code above.

Nieuwe redacteur

Rob Willemsen, nieuwe redacteur met aandacht voor Microsoft-Development.



Sample 4: Using the MSXML Parser from a webpage
The same sample as the above, but now using the XML parser instead of the SOAP toolkit to get the result.

```
<%@ Language=VBScript %>
<HTML>
<TITLE>SOAP-Toolkit 2.0 Sample 1</TITLE>
<BODY>
  <P>
    <H2>Test if the connection can be made to the server
      and a valid response is returned</H2>
  </P>
  <%

  ' Set up some vars to use later on
  const cServer     = " http://www.test.bizzview.com/" &_
    "_contentmgr/contentmgr.wsdl "
  const cService    = "ContentMgr"

  DIM lcSOAPmessage
  ' build soap request string
  lcSOAPmessage = ""
  lcSOAPmessage = lcSOAPmessage "<?xml version='1.0' " &_
    "encoding='UTF-8' standalone='no' ?>"
  lcSOAPmessage = lcSOAPmessage + "<SOAP-ENV:Envelope" &_
    "SOAP-ENV:encodingStyle=" &_
    "'http://schemas.xmlsoap.org/soap/encoding/'" &_
    "xmlns:SOAP-ENV=" &
    "'http://schemas.xmlsoap.org/soap/envelope/'>"
  lcSOAPmessage = lcSOAPmessage & "<SOAP-ENV:Body>"
  lcSOAPmessage = lcSOAPmessage & "<SOAPSDK1:EchoString
    xmlns:SOAPSDK1='http://bizzview.com/message/'>"
  lcSOAPmessage = lcSOAPmessage &_
    "<cString>Hello World</cString>"
  lcSOAPmessage = lcSOAPmessage &_
    "</SOAPSDK1:EchoString>"
  lcSOAPmessage = lcSOAPmessage + "</SOAP-ENV:Body>"
  lcSOAPmessage = lcSOAPmessage + "</SOAP-ENV:Envelope>"

  ' send soap request string by ServerXmlHttp
  set oXmlHttp =
    server.CreateObject("MSXML2.ServerXMLHTTP")
  call oXmlHttp.open("POST", cServer, false)
  call oXmlHttp.setRequestHeader(_
    "Content-Type:","text/xml")
  call oXmlHttp.setRequestHeader("SOAPAction:",_
    "http://bizzview.com/action/ContentMgr.EchoString")
  call oXmlHttp.send(lcSOAPmessage)

  ' Check status of the http request (200 = succes)
  if oXmlHttp.status = 200 then
    lcSOAPresponse = oXmlHttp.responseText
   else
    Response.Write("<HR>Failed to get SOAP response: " &_
      oXmlHttp.status )
    Response.End
  end if

  ' filter useful response out of SOAP response
  set oXmlParser =
    Server.CreateObject("MSXML2.DomDocument")
  oXmlParser.loadXML(lcSOAPresponse)
  set loReturn = oXmlParser.documentElement.selectNodes(_
    "//SOAPSDK1:EchoStringResponse/Result")
  if loReturn.length = 1 then
```

```
      lcReturn = loReturn.item(0).text
     else
      ' return error string
      lcReturn="<ERROR><CODE>0</CODE><DESCRIPTION>" &_
        "Geen soap result</DESCRIPTION></ERROR>"
    end if

    FOR lnLoop = 1 TO 3
      Response.Write("<H" & cStr(lnLoop) & ">" &_
        lcReturn & "<H" & cStr(lnLoop) &"></BR>")
    NEXT

    Response.Write("<HR>")
    ' Release the object
    Set oXmlHttp   = Nothing
    Set oXmlParser = Nothing
    %>
</BODY>
</HTML>
```

The result in your web browser will look like the same as in the previous sample.

If you're lazy, you'll probably fancy the SOAP toolkit approach: less code and very high-level functions to work with

Notice however that it takes more code than the previous sample to reach this same result. What is the best solution then? As always it depends. If you're lazy, you'll probably fancy the SOAP toolkit approach: less code and very high-level functions to work with. To be honest I started out that way also. But when I used the XML parser approach and benchmarked the two ways of working, the performance penalty for using the SOAP toolkit approach turned out to be high. In some cases the difference between the two was 10 seconds or more, so it pays off to do some extra work here. The SOAP toolkit though offers great debugging capabilities, which isn't possible using the XML parser. The debugging capabilities and why it takes longer using the SOAP-Toolkit are issues I'll address in the next article about web-services.

Remi Caron is technical director and co-founder of BizzView B.V. a company that specializes in e-business solutions. Remi has a long history as a Foxpro developer. These days his company uses any Visual Studio tool that suits the job best.

He is also the section manager for the MS-Development group within the Software Developers Group Netherlands. He can be reached at Remi.Caron@BizzView.com

auteur
Ed van Akkeren

Ontwikkeling en Beheer: een Gedwongen Huwelijk

In de ICT werken vogels van velerlei pluimage en dan heb ik het nu eens niet over de diverse achtergronden, want die zijn inmiddels legendarisch. Ooit iemand ontmoet die zo vanaf de school/college de ICT is ingedoken? Bekijk hem.haar maar eens goed, want het is een heel erg zeldzaam wezen. Nee, ik heb het hier over de verschillende subdisciplines binnen de branche. In de eerste plaats hebben we natuurlijk de scherpe scheiding tussen de hardware en de software, maar binnen die software kunnen we ook een aantal bloedgroepen onderscheiden die, terwijl ze zouden moeten samenwerken, maar al te vaak op voet van (bijna) oorlog met elkaar leven. Een schrijnend voorbeeld is het al sinds het begin van ICT durende gekissebis tussen de ontwikkelaars enerzijds en de beheerders anderzijds. Hierbij is mijn definitie van een ontwikkelaar iemand die een informatiesysteem initieel ontwikkelt, terwijl een beheerder in mijn optiek diegene is die het systeem daarna in de lucht houdt en de nodige aanpassingen verricht, inspelend op veranderende gebruikerswensen, of eisen van markt en/of overheid. Beide groepen houden zich bezig met hetzelfde product, maar dat voorkomt niet dat de kloof tussen de twee beroepsgroepen nog altijd niet is overbrugd.

Geen systeemontwikkelingsmethode kan compleet zijn zonder een fase beheer, doorgaans volgend op de fase acceptatie

Geen systeemontwikkelingsmethode kan compleet zijn zonder een fase beheer, doorgaans volgend op de fase acceptatie. Als je de boeken dus mag geloven is het gewoon een processtap binnen het ontwikkelproces, dus zou het een kwestie moeten zijn van gewoon doorgaan met die invuloefening die (bijvoorbeeld) SDM heet en alles loopt goed. De praktijk echter, blijkt keer op keer toch anders te zijn. Beheer van een eenmaal opgeleverd informatiesysteem blijkt in de praktijk maar al te vaak een lange lijdensweg te zijn, waarbij irritatie over en weer, ingrijpen van het management en uiteindelijk veel hogere kosten dan geraamd eerder regel dan uitzondering zijn. Het zijn niet alleen technische of procedurele problemen die hierbij een rol spelen, maar ook zaken als het type mens dat zich met de twee vakgebieden bezig houdt en de invalshoek van waaruit het management de zaken bekijkt spelen hier een grote rol. Dit essay pretendeert niet een antwoord te geven op de hierbij opkomende vragen –daar zou een compleet boek voor nodig zijn-, maar probeert de probleemgebieden op een beknopte wijze in kaart te brengen met als achterliggende gedachte dat het herkennen (en erkennen) van een probleem vaak al de helft van de oplossing is. We kunnen nu wel blijven schelden op 'die anderen', maar we moeten uiteindelijk wel samenwerken.

De Bron van alle Ellende

De eerste problemen dienen zich doorgaans al aan bij de Gebruikers Acceptatie Test. Een ontwikkelorganisatie richt zich vrijwel volledig op het leveren van een programma dat aan de wensen van de gebruikers tegemoet komt. Echter, een gebruiker is in principe niet geïnteresseerd in de manier waarop zulks is gebeurd. Een lasser wil dat een lasapparaat werkt en als dat zo is wordt het ding geaccepteerd. Hoe het apparaat er van binnen uitziet is voor de gebruiker van geen enkel belang. In de film *Kelly's Heroes* zit tankcommandant Donald Sutherland temidden van de krijgshandelingen rustig voor zijn kapotte tank'te wachten op de monteur. "I just drive a tank, man; I don't know anything about them", hoor je hem daarbij blijmoedig zeggen. Dit geldt natuurlijk precies zo voor een eindgebruiker en een informatie verwerkend systeem. De eindgebruiker kijkt of al zijn eisen zijn gerealiseerd: krijgt men de lijsten die men wil hebben, kunnen de gegevens makkelijk (gebruikersvriendelijk) worden ingebracht, liggen de responsetijden een beetje gunstig, zien de schermen er rustig en overzichtelijk uit. Als dat allemaal naar wens is, wordt het systeem geaccepteerd, waarna het overgedragen wordt naar de afdeling Beheer. En dan begint doorgaans het gedonder.

Documentatie en Documentatie

De afdeling Beheer zal in eerste instantie gaan kijken naar de documentatie. Nu is dit vaak al een sluitpost op de begroting van de ontwikkelaar, maar zelfs als dit goed

verzorgd is zal het vaak zijn toegespitst op de eisen die de programmeur stelt en niet op degene die het stokje daarna moet overnemen. Documentatie t.b.v. de ontwikkeling van informatiesystemen richt zich op de realisatie van het grote geheel. Bij het bouwen van een huis heeft het geen zin over het dak te gaan praten als de fundering er nog niet ligt en het hele plan van aanpak zal een voor een bouwers logische volgorde aanhouden: eerste de fundering, dan de eerste verdieping, dan de volgende verdiepingen, het dak, enz, gevolgd door de afbouw, welke weer gevolgd wordt door de installatie van verwarmingsketels, de elektriciteit, enz. Een plan van aanpak voor de realisatie van een applicatie zal zich eerst bezig houden met de database, gevolgd door de onderhoudsschermpjes met moduleboom, weer gevolgd door de binnen deze schermen te realiseren functionaliteit, foutafvang, enz. Zo'n indeling is voor een ontwikkelaar volkomen logisch, maar voor een beheerder betekent het vaak waden door de overbodige informatie voordat men arriveert op het stuk dat men wil hebben. Zelfs als alles wat een beheerder nodig zou hebben er keurig in zou staan, dan nog leidt dit speurwerk tot veel irritatie en gescheld op "die koekebakkers van de ontwikkeling" die weer eens niet goed hun documentatie verzorgd hebben. Vaak is dit ook zo, maar vaak is het gewoon een kwestie van de andere invalshoek die het voor de beheerder onnodig veel moeilijker maakt.

Een ander probleem is dat het vaak erg onduidelijk is wat een beheerder nu graag aan documentatie ziet en hoe hij dat het liefste ingericht zou willen zien

Een oplossing hiervoor zou het uitbreiden van de documentatie met op de beheerder toegesneden informatie moeten zijn, maar documentatie is, ik meldde het al eerder, doorgaans een sluitpost op de begroting en als er ergens bezuinigd moet worden dan is doorgaans het papier als eerste aan de beurt. Zolang de eindgebruikers tevreden gesteld kunnen worden is men bij het management al in zijn nopjes, maar het management is doorgaans ook eindgebruiker en dus niet de meest neutrale partij. Ik kom hier verderop nog op terug. Een ander probleem is dat het vaak erg onduidelijk is wat een beheerder nu graag aan documentatie ziet en hoe hij dat het liefste ingericht zou willen zien. De ene beheerder is de andere niet en verschillende organisaties stellen vaak verschillend eisen, doorgaans voorkomend uit bijkomende zaken als bedrijfscultuur, de gebruikte ontwikkelomgeving en de vaardigheid van het zittende beheersteam. Pas als deze zaken goed in kaart zijn gebracht is het mogelijk iets van richtlijnen op te stellen, maar dan is het doorgaans vanuit bedrijfseconomisch standpunt niet meer haalbaar.

Boutjes en Moertjes

Moedeloos van het speurwerk in de documentatie duikt de beheerder dan maar de code in. Als je de bouwtekeningen van de motor niet snapt, rommel dan maar wat met boutjes en moertjes, dan wordt het allemaal wel duidelijk. Hopelijk. Helaas luidt dit meteen de tweede frustratieronde in. Met het risico een hoop beheerders tegen de schenen te schoppen (waarvoor bijgaand dan direct mijn excuus, maar mijn praktijkervaring is niet anders), moet ik stellen dat de technische vaardigheden van een beheerder zich doorgaans niet op het niveau van de ontwikkelaar bevinden. Maakt dit de beheerder dan tot een gemankeerd programmeur, net als een basgitarist ook doorgaans een weinig getalenteerd (en dus gefrustreerd) leadgitarist is? Nee, zeker niet. Van nature is de ontwikkelaar meer gericht op de techniek en hij zal zich dan ook eerder bedienen van de trucs van de echte techneut. Vaak speelt hier ook een beetje pedanterie een rol, zo van 'kijk mij eens flitsend programmeren'. In de goede oude procedurele Clippertijd hadden we dat al. Bij het zien van een commando als CLEAR SCREEN kon zelfs een niet-programmeur nog wel bedenken wat er ging gebeuren. Vandaar dat een echte goeroe de (in deze context ongedocumenteerde) functie Scroll() gebruikte, waarmee hij fijntjes aangaf de /p compilatieparameter bestudeerd te hebben. Met de advent van object oriëntatie is het alleen maar erger geworden. Zo kon je op seminars in de begintijd de OO-predikers enthousiast horen vertellen over hoe je met encapsulatie en inheritance een hoop 'nasty code' voor mindere begaafde vakbroeders kon verbergen en zo mooie schone code kon opleveren. Duidelijk aan de ontwikkelaar gerichte taal, dus, want een beheerder wil die 'nasty code' duidelijk en op een logische plek kunnen zien en niet ergens ver weggestopt in een vaag benoemd (MyOtherObject.SecondMethod) user-defined method van een great-great-great-great-grandparent. Kritiek van deze aard wordt door een ontwikkelaar doorgaans weggewimpeld met een opmerking als "een programmeur die dit niet begrijpt heeft hier niets te zoeken", een aardige variant op het aloude "if it was hard to write it should be hard to read". Waar men dan vaak aan voorbij gaat is dat de ervaren rot altijd wordt vervangen door een novice. Oftewel: kende jij het Scroll() commando toen je begon?

Een oplossing is dus meer aandacht aan de inherente begrijpelijkheid en interne logica van de code zelf en dan heb ik het niet over nog meer commentaar. Code moet een logische flow hebben, functies moeten zinnige namen hebben. Een methode om een form te openen

(als je bijvoorbeeld de standaard methode hebt uitgebreid met een foutafhandeling) moet iets als *OpenForm* heten en niet *PietjesMethode*. Als er een keuze is tussen compactheid en begrijpelijkheid moet altijd voor het laatste gekozen worden, dus geen twintig functies binnen elkaar, maar hulpvariabelen doorgeven. Helaas wringt hem daar doorgaans de schoen. In tegenstelling tot wat je zou verwachten heeft de doorsnee programmeur (en zeker hij die is opgegroeid in het cut-and-paste paradijs van de GUI workbench) een broertje dood een extra klopwerk. Als je het in een enkele regel kan ga je het natuurlijk niet in vijf doen. En om het nog erger te maken is de projectleider het doorgaans wel met zo'n instelling eens. Vijf regels kosten meer tijd dan een en budgets staan altijd en overal onder druk, economische hausse of baisse. Het resultaat is vaak dat een beheerder uiteindelijk een cryptomaan codewoud van elkaar aanroepende functies over zich heen krijgt, waarin ook nog eens niet of schaars gedocumenteerde features worden gebruikt. De verzuchting dat 'dit zooitje niet langer meer te onderhouden is', is dan al snel geslaakt.

De Aard van het Beestje

Veel van de verschillen tussen ontwikkelaars en beheerders zijn ook terug te voeren op de aard van het beestje. Men wordt al in den beginne op andere competenties geselecteerd en zoiets leidt in elke organisatie natuurlijk tot de opkomst van subculturen. Een typisch ontwikkelaar is wat ze in de moderne psychologie ook wel een thrillseeker noemen, iemand die altijd te porren is voor iets nieuws, een beetje risico erbij vindt horen. De ontwikkelaar ziet zichzelf graag als van nature creatief, vernieuwend en ondernemend. Niet voor niets is het percentage bungeejumpers onder de ontwikkelaars onevenredig hoog. Beheerders zijn in hun ogen vaak saaie, behoudzuchtige figuren die op elke slak zout leggen en op elke weg hele cohorten beren menen te zien. De beheerder is inderdaad sterk gericht op continuïteit en betrouwbaarheid en dat tracht hij te bereiken door juist alle risico's uit te bannen. In tegenstelling tot de ontwikkelaar gruwt hij van nieuwe technologieën. "Tried and tested" is zijn devies, alleen dan kan je de gebruiker garanderen dat het systeem ook morgen nog werkt, als de ontwikkelaars naar huis zijn en de routine zijn intrede moet doen. Voor hem is de ontwikkelaar een onverantwoordelijke charlatan die continu bezig is de beheersbare infrastructuur en service te torpederen met weer een nieuwe luchtfiets. Een ontwikkelaar richt zich volledig op die ene keer. Zijn product moet een keer schitteren en daarna is hij weg. Na hem de zondvloed. Het is de taak van de beheerder die zondvloed te voorkomen. Overigens is dat niet meer dan menselijk: niemand staat er op te wachten om 's nachts drie uur zijn bed uit te moeten om een probleem op te lossen dat iemand anders (die vermaledijde ontwikkelaars, natuurlijk) veroorzaakt heeft. Helaas moeten deze twee tegengestelden diersoorten vervolgens wel tegen hun natuur in met elkaar samenwerken.

De beheerder is inderdaad sterk gericht op continuïteit en betrouwbaarheid en dat tracht hij te bereiken door juist alle risico's uit te bannen

Instinctief zou je zeggen dat de enige oplossing voor deze tegenstelling het kweken van onderling begrip is. Breng ontwikkelaars en beheerders al vroeg in het ontwikkeltraject bij elkaar en men krijgt inzicht in elkanders problematiek. Eenvoudig, en zo nuchter bekeken zou dat alleen maar tot wederzijds voordeel moeten leiden: de beheerder heeft kennis van valkuilen waar de ontwikkelaar fluitend aan voorbij gaat. En als hij ruim van tevoren weet wat voor eisen er aan zijn documentatie gesteld gaan worden, dan wordt het voor de ontwerper of documentalist een stuk eenvoudiger om daar rekening mee te houden. Helaas levert dit dan weer verdere problemen op: het is voor beheerders vaak erg moeilijk om eisen vanuit beheer en acceptatiecriteria te formuleren: de materie is vaak nog te onbekend of nog maar zelden vanuit de invalshoek van de informatietechnologie bekeken. En als het wel lukt om zinnige ijkpunten op te stellen duurt dit vaak zo lang dat het management het allang om bedrijfseconomische redenen heeft afgeschoten. Hiernaast blijft het 'gemiereneuk van die beheerders' natuurlijk een bron van irritatie, welke de voortgang van het ontwikkelproject dermate kan frustreren dat er uiteindelijk helemaal niets wordt opgeleverd.

Internet Tip

Clientside JavaScript: History.Go(-2)

In een internetapplicatie wil je soms na het invullen van een formulier terug naar een bepaalde pagina. In plaats van het redirecten kun je ook gebruik maken van de browsercache. Dit levert meestal een betere performance.
Doe dit met het history-object. Onderstaand voorbeeld laat de browser twee pagina's terugspringen.

```
history.go(-2)
```

Manage IT

Helaas voor de beheerders, het management staat maar al te vaak aan de kant van de ontwikkelaars. De realisatie van een systeem is een keurig afgebakend stukje werk, waar je als manager makkelijk kan ingrijpen. Iedere manager weet dat hij scoort bij zijn superieuren als hij iets van een budget af kan schaven en een ontwikkeltraject leent zich hier natuurlijk prachtig voor. Voor het op tijd (en binnen het budget) afronden van een systeem is het natuurlijk niet nodig om 'netjes' te programmeren of de documentatie volledig op orde te hebben. Zij worden daarin gesteund door de eindgebruikers, die vaak niet eens de moeite nemen om de handleiding te lezen, omdat ze toch wel een training krijgen en daarbij de helpdesk twee deuren verder zetelt. Als een soort van zoenoffer aan het beheersteam wordt er natuurlijk wel het nodige papier opgeleverd, maar de bruikbaarheid daarvan blijkt toch pas als na een jaar het eerste onderhoud gepleegd moet worden en dan zijn de ontwikkelaars toch al weer buiten de driemijlszone.

Als een soort van zoenoffer aan het beheersteam wordt er natuurlijk wel het nodige papier opgeleverd

Binnen de hardware speelt het concept Total Cost of Ownership al een tijdje een belangrijke rol. Men zag in dat de kosten van een PC uit meer componenten bestonden dan de aanschafprijs van de hardware, maar dat ook zaken als het salaris van de systeembeheerder en de prijs van de aansluiting op het internet erin verwerkt moesten worden. Zoiets zou voor een maatwerkapplicatie ook geen slechte oplossing zijn, alhoewel het aantal parameters natuurlijk zeer veel groter is. Bovendien is het vaak erg moeilijk in te schatten wat de kosten zullen zijn van onderhoud als je nog niet eens weet of, wanneer en voor hoeveel er onderhoud gepleegd gaat worden. Toch is het mogelijk om vanuit de historie iets zinnigs te zeggen over het totale financiële plaatje, wat het voor een manager dan weer mogelijk maakt om inzicht te krijgen in de gevolgen van zijn ongebreidelde cost cutting tijdens de ontwikkeling, en de beheerder een instrument geeft om zijn focus op continuïteit en beheersbaarheid economisch te onderbouwen.

De Architectuur

De laatste tijd wordt er veel geschreven over de systeemarchitectuur als middel tegen dit soort kwalen. Als we alle standaards en richtlijnen maar vervatten in een enkel dwingend document, dan zal alles wel goed komen. Natuurlijk, als de kaders zijn gezet, dan is het voor de ontwikkelaar vanaf het prille begin duidelijk wat er van hem wordt verwacht en wordt het voor de beheerder ook een stuk inzichtelijker wat hij kan verwachten. Helaas moeten de kaders wel worden afgedwongen en daar zit hem nu net de kneep. Als de kaders te strak zijn opgesteld zal de hele architectuur al snel als te beklemmend worden gezien en is de kans groot dat de acceptatiegraad nul is. Op papier klopt het dan allemaal wel, maar in de praktijk wordt het een zooitje, wat het voor het management dan natuurlijk helemaal oncontroleerbaar maakt. Andersom hebben te losse kaders ook weinig tot geen nut: als de grenzen zo ruim worden gesteld dat zo'n beetje elk gekkenwerk nog binnen de sjabloon past zal de stammenstrijd nog onverdroten doorgaan, alleen nu onder de nominale paraplu van de bedrijfsarchitectuur. Directie blij, duurbetaalde consultant blij, maar de oorlog tussen ontwikkelaars en beheerders duurt gewoon voort.
Het grootste probleem is dus het fine tunen van de richtlijnen en het parallel daaraan enthousiasmeren van de betrokkenen, want, laten we wel zijn: op papier kan het er nog zo fraai uitzien, het zijn uiteindelijk de mensen die het hem moeten doen. De theorie is geduldig, maar de praktijk verlangt actie. Toch lijkt het de moeite waard om hier aandacht aan te besteden. De mogelijke opbrengsten zijn verleidelijk hoog: beheerders zijn doorgaans net zo duur als ontwikkelaars, maar wel veel langer bezig, dus hoe eenvoudiger we het onderhoud kunnen maken, hoe meer we hier kunnen besparen.

Conclusie

De documentatie, de code, de personen, het management: met deze vrijwel onuitputtelijke bronnen van onderlinge misverstanden en wantrouwen zal het nog wel even duren voordat de ontwikkelaar en de beheerder de samenwerking kunnen opbrengen die de kwaliteit van het softwareproduct op een hoger plan kan brengen. Toch behoeven we niet te wanhopen: zolang we de klant en het resultaat centraal kunnen blijven stellen is er sprake van een gemeenschappelijk doel en zoiets bindt. Op langere termijn kan dan een verdere uitgewerkte bedrijfsarchitectuur soelaas gaan bieden. Immers, de enige echte oplossing voor dit soort problemen blijft nog altijd de meta-aanpak, het vanuit een hoger niveau bekijken van het probleemgebied. Want vanaf de maan gezien, zo schreef Multatuli ooit eens, zijn we allemaal even groot...

Ed is van hetzelfde bouwjaar als Madonna en Michael Jackson, maar daar houdt verder iedere gelijkenis op, want zijn creatieve aanvallen viert hij alleen bot op de PC. Hij kwam via Clipper de IT (toen nog zonder C) binnenrollen, stapte daarna over op Delphi, maar houdt zich thans voornamelijk bezig met informatie analyse en het ontwerp van systemen voor de diverse klanten van zijn werkgever, Getronics Business Solutions. In zijn vrije tijd wil hij als hobby nog wel eens trachten zijn jaargenoten middels gitaarspel en zang te emuleren, maar de meningen daarover zijn nogal verdeeld...!

tools

- ✔ Delphi 6
- ✔ XML

auteur

Bob Swart

Delphi 6 XML mapper

Een van de gebieden waarop Delphi 6 is uitgebreid met nieuwe features en extra ondersteuning is XML en het werken met XML documenten of XML datasets. Er zit zelfs een geheel nieuwe tool bij Delphi 6 (ook als losse tool bruikbaar) genaamd de XML Mapper, waarvan ik deze keer wil laten zien wat je er mee kan.

XML Documenten

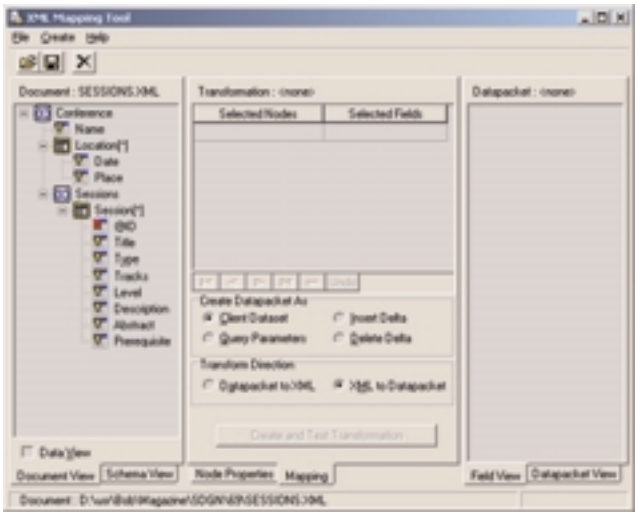
De XML Mapper is als externe tool beschikbaar (maar ook onder het Delphi 6 Tools menu), en kan gebruikt worden voor het omzetten van een XML document naar een (DataSnap compatible) dataset, een XML document naar een (ander) XML document, of een dataset terug naar een XML document. De XML Mapper weet al hoe een dataset eruit moet zien, zodat we alleen nog moeten aangeven hoe ons XML document eruit ziet. Hiervoor kunnen we twee dingen gebruiken: het XML document zelf, of een XML schema bestand. Een XML schema is waar we vroeger een DTD bestand voor gebruikte, en bevat de beschrijving van het XML bestand. Omdat niet iedereen een XML Schema zal hebben, zal ik als voorbeeld beginnen met een normaal XML document, en wel eentje die gebruikt kan worden om conferentie sessies in te vullen (bijvoorbeeld als resultaat van een "Call for Papers"). Het SESSIONS.xml document dat ik gebruik beschrijft een drietal sessies die ik voor de Europese Borland Conferentie heb gehouden, die zelf weer op twee lokaties is geweest. Een op twee plaatsen genest XML document dus (waarbij ik de abstract in de listing even leeg heb gemaakt om niet teveel ruimte in beslag te nemen):

```
<?xml version="1.0" standalone="yes"?>
<Conference>
<Name>BorCon Europe 2001</Name>
<Location>
  <Date>2001/09/17-2001/09/18</Date>
  <Place>London, UK</Place>
</Location>
<Location>
  <Date>2001/09/20-2001/09/21</Date>
  <Place>Noordwijkerhout, NL</Place>
</Location>

<Sessions>
 <Session ID="1001">
   <Title>Building WAP Apps with Delphi</Title>
   <Type>regular session</Type>
   <Tracks>Delphi/Kylix, Internet</Tracks>
   <Level>Intermediate</Level>
   <Description>Building WAP applications in Delphi using
WebBroker</Description>
   <Abstract>...</Abstract>
   <Prerequisite>None</Prerequisite>
 </Session>
 <Session ID="1002">
   <Title>Crossplatform Application Development with
Delphi and Kylix</Title>
   <Type>regular session</Type>
   <Tracks>Delphi</Tracks>
   <Level>Beginning, Intermediate</Level>
   <Description>Crossplatform application development
using Kylix Delphi 5 and 6</Description>
   <Abstract>...</Abstract>
   <Prerequisite>None</Prerequisite>
 </Session>
 <Session ID="1003">
   <Title>VisiBroker for Delphi (CORBA)</Title>
   <Type>regular session</Type>
   <Tracks>Delphi</Tracks>
   <Level>Intermediate</Level>
   <Description>CORBA Programming Techniques using
VisiBroker for Delphi</Description>
   <Abstract>...</Abstract>
   <Prerequisite>None</Prerequisite>
 </Session>
</Sessions>
</Conference>
```

Het volledige XML document kan bijvoorbeeld gebruikt worden om in één keer mijn sessie abstracts op te sturen als reactie op de call-for-papers, terwijl op dit moment nog vaak via een ouderwets HTML web interface een formulier moet worden ingevuld, of een word template moet worden bewerkt. Wellicht kunnen we over een jaar of wat een XML schema krijgen als input, en dan een ingevuld XML document opleveren als output. De bewerking daar weer van kan dan ook wat makkelijker gebeuren, zoals ik in dit artikel zal proberen te laten zien.

XML Mapping Tool

Start nu de XML Mapper, en doe *File | Open* om het voorbeeld XML Document te laten analyseren. Het scherm van de XML Mapper bestaat uit drie delen (zie screenshot #1). Links staat het XML Document, rechts het Datapacket (komen we zo op), en in het midden de Transformation informatie.



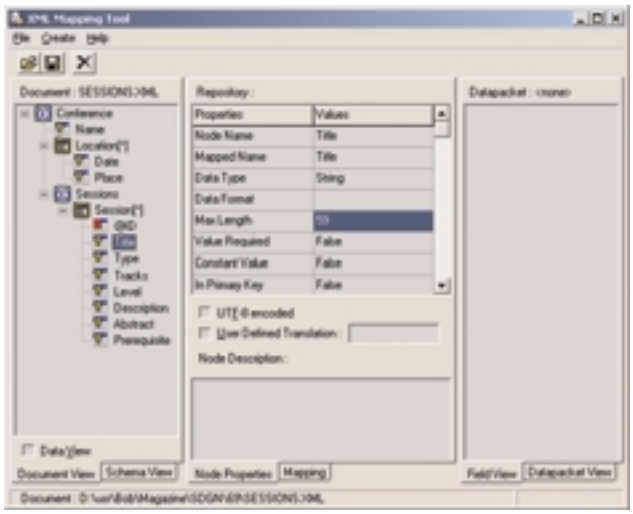
Screenshot #1

Om met links te beginnen: het XML document kan zowel in de oorspronkelijke Document View bekeken worden als in de Schema View. Dat laatste is leuk om naar te

kijken, ook als je geen XML Schema had, want de XML Mapper is in staat om zelf een soort XML Schema te genereren voor je XML Document. Het ziet er niet geweldig uit, maar je kan het wel gebruiken om zelf mooier te maken (tip: met de rechtermuisknop in de Schema View kun je het XML Schema opslaan, en daarna gewoon als XSD document verder bewerken).

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Conference" type="ConferenceType"/>
  <xs:complexType name="ConferenceType">
    <xs:sequence>
      <xs:element name="Name" type="NameType"/>
      <xs:element name="Location" type="LocationType"
minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Sessions" type="SessionsType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Name" type="NameType"/>
  <xs:simpleType name="NameType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Location" type="LocationType"/>
  <xs:complexType name="LocationType">
    <xs:sequence>
      <xs:element name="Date" type="DateType"/>
      <xs:element name="Place" type="PlaceType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Date" type="DateType"/>
  <xs:simpleType name="DateType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Place" type="PlaceType"/>
  <xs:simpleType name="PlaceType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Sessions" type="SessionsType"/>
  <xs:complexType name="SessionsType">
    <xs:sequence>
      <xs:element name="Session" type="SessionType"
minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="Session" type="SessionType"/>
  <xs:complexType name="SessionType">
    <xs:sequence>
      <xs:element name="Title" type="TitleType"/>
      <xs:element name="Type" type="TypeType"/>
      <xs:element name="Tracks" type="TracksType"/>
      <xs:element name="Level" type="LevelType"/>
      <xs:element name="Description"
type="DescriptionType"/>
      <xs:element name="Abstract" type="AbstractType"/>
      <xs:element name="Prerequisite"
type="PrerequisiteType"/>
    </xs:sequence>
    <xs:attribute name="ID" type="xs:string"/>
  </xs:complexType>
  <xs:element name="Title" type="TitleType"/>
  <xs:simpleType name="TitleType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Type" type="TypeType"/>
  <xs:simpleType name="TypeType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Tracks" type="TracksType"/>
  <xs:simpleType name="TracksType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Level" type="LevelType"/>
  <xs:simpleType name="LevelType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Description" type="DescriptionType"/>
  <xs:simpleType name="DescriptionType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Abstract" type="AbstractType"/>
  <xs:simpleType name="AbstractType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:element name="Prerequisite"
type="PrerequisiteType"/>
  <xs:simpleType name="PrerequisiteType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
</xs:schema>
```

Het middelste deel van de XML Mapper kan overigens ook gebruikt worden om de properties van de verschillende knopen in de XML boom (links in de Document View) aan te passen. Bij het inlezen van het document wordt namelijk voor iedere knoop het type bepaald en de maximale lengte. Dit laatste gebeurt gewoon door de lengte van de waarden te nemen, en er vanuit te gaan dat dat meteen de maximale lengte is. Bij Title is de maximale lengte dus kennelijk 59, omdat we een Session met de Title "Crossplatform Application Development with Delphi and Kylix" hebben. Door op de Node Properties tab te klikken (in het midden van de XML Mapper) kun je per Node de properties aanpassen, zoals de Max Length voor Title (van 59 naar 64 ofzo - zie screenshot #2).



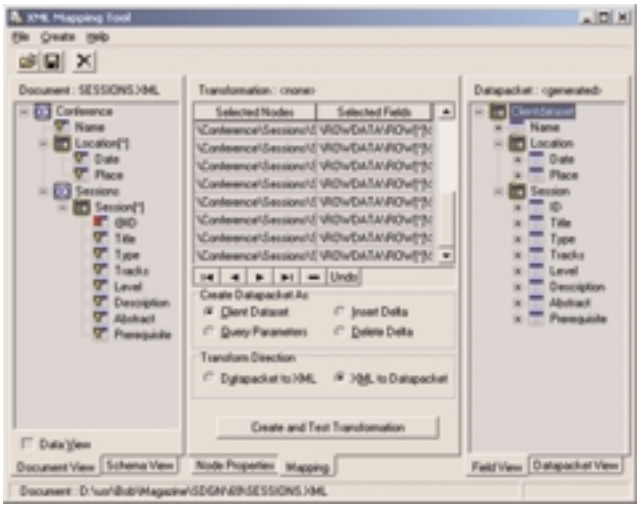
Screenshot #2

Overigens krijg je de data van het XML document te zien als je op de "Data View" checkbox klikt in het linker deel van het scherm. Dat levert echter niet een duidelijker overzicht op, vandaar dat ik zelf deze optie vrijwel nooit gebruik - hooguit een keer om te kijken hoe de data achter de knopen hangt.

Transformeren

Als je klaar bent met het aanpassen van de node properties, kun je in het linker deel van de XML Mapper aangeven welke onderdelen van de boom je eigenlijk allemaal wilt transformeren. Het zou bijvoorbeeld kunnen zijn dat je alleen interesse hebt in het geneste Session deel. Klik dan met de rechtermuisknop op de Session knoop, en kies "Select All Children" (alleen Select heeft weinig zin, want de Session knoop zelf heeft geen inhoud). Hierna zul je de geselecteerde knopen in het midden van het scherm terugzien, onder de Mapping tab. In mijn geval ben ik echter geïnteresseerd in de volledige inhoud van het XML document, inclusief de twee geneste delen Location en Session, dus ik klik met de rechter-muisknop en kies "Select All" (Ctrl+A). Aals je dat doet nadat je "Select all Children" hebt gedaan dan is de volgorde van de knopen in de mapping verkeerd - klik dan met de rechter-muisknop eerst in het midden van het scherm

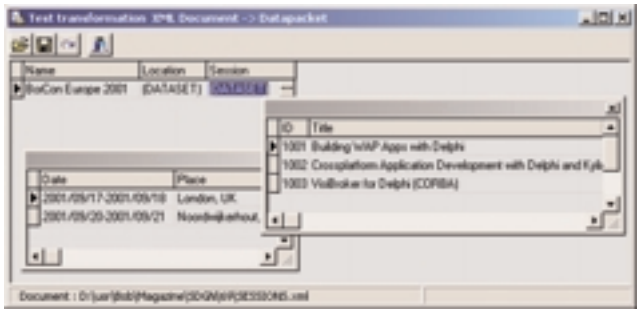
en kies "Clear" (Ctrl+C) om het middenstuk weer leeg te maken. Klik vervolgens weer links, en kies "Select all".
Om ook de rechterkant van de XML Mapper te vullen moet je nog eenmaal met de rechtermuisknop klikken en kiezen voor "Create Datapacket from XML" (Ctrl+D). In het middelste deel van de XML Mapper zie je nu zowel de geselecteerde knopen als de getransformeerde velden die erbij horen. Daarnaast is in het rechterdeel nu de gegenereerde dataset te zien (zie screenshot #3). Merk op dat Location en Session allebei als geneste dataset zijn opgenomen. Die zullen we straks nog in detail terugzien.
We zijn er bijna, maar nog niet helemaal.



Screenshot #3

Create and Test

Om nu de transformatie daadwerkelijk te starten moeten we nog even expliciet op de grote button met "Create and Test Transformation" drukken. We krijgen dan een form te zien met daarin de "Test transformation XML Document -> Datapacket" in de vorm van een DBGrid dat aan een ClientDataSet verbonden is. Naast het veld Naam bevat dit twee geneste datasets: Location en Session. Beide geneste datasets kunnen vertoond worden door de velden in het grid te selecteren en daarbij op de elipsis (...) te klikken. Dit levert per geneste dataset een pop-up scherm op met daarin weer een DBGrid en de records van de betreffende geneste dataset (zie screenshot #4).



Screenshot #4

Bewaren

Via *File* | *Save* kunnen we nu zowel een ClientDataSet definitie bestand bewaren in XML formaat (dus de vertaling van het XML Schema naar een XML definitie), als ►

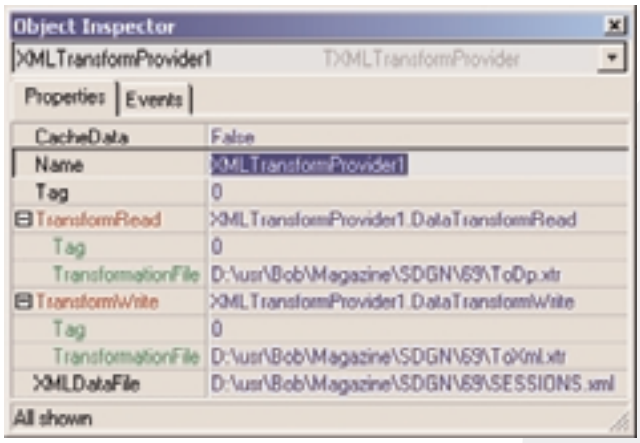
de transformatie informatie. Om met het eerste te beginnen: de ClientDataSet XML bestaat uit één lange regel van 915 bytes, maar kan iets leesbaarder gemaakt worden tot:

```
<?xml version="1.0" standalone="yes"?>
<DATAPACKET Version="2.0">
<METADATA>
 <FIELDS>
  <FIELD attrname="Name" fieldtype="string" WIDTH="18"/>
  <FIELD attrname="Location" fieldtype="nested">
   <FIELDS>
    <FIELD attrname="Date" fieldtype="string" WIDTH="21"/>
    <FIELD attrname="Place" fieldtype="string" WIDTH="19"/>
   </FIELDS>
   <PARAMS/>
  </FIELD>
  <FIELD attrname="Session" fieldtype="nested">
   <FIELDS>
    <FIELD attrname="ID" fieldtype="string" WIDTH="4"/>
    <FIELD attrname="Title" fieldtype="string" WIDTH="59"/>
    <FIELD attrname="Type" fieldtype="string" WIDTH="15"/>
    <FIELD attrname="Tracks" fieldtype="string" WIDTH="22"/>
    <FIELD attrname="Level" fieldtype="string" WIDTH="23"/>
    <FIELD attrname="Description" fieldtype="string" WIDTH="173"/>
    <FIELD attrname="Abstract" fieldtype="bin.hex" SUBTYPE="Text"/>
    <FIELD attrname="Prerequisite" fieldtype="string" WIDTH="4"/>
   </FIELDS>
   <PARAMS/>
  </FIELD>
 </FIELDS>
 <PARAMS/>
</METADATA>
<ROWDATA/>
</DATAPACKET>
```

Zoals kenner meteen zullen zien bevat dit XML dataset bestand alleen maar de metadata, en geen inhoud voor de rowdata. Oftewel: dit XML dataset bestand bevat de definitie, maar is verder een lege dataset. Dat kan toch handig zijn, want hiermee kun je data-entry schermen bouwen. Wat echter vaak handiger is, is om de transformatie informatie op te slaan, zodat XML documenten met inhoud omgezet kunnen worden naar een XML dataset met de hierboven beschreven definitie. Hiervoor moeten we de transformatie informatie opslaan in een XTR bestand, waarvan de XML Mapper als naam ToDp.xtr voorstelt (To DataPacket). Het aardige is dat ook de transformatie informatie nu is opgeslagen in XML formaat. Hierdoor kun je later de transformatie nog met de hand aanpassen, alhoewel het altijd makkelijker zal blijven om daarvoor de XML Mapper zelf weer te gebruiken. In ieder geval is het ToDp.xtr bestand de informatie die we nodig hebben om het oorspronkelijke XML document om te zetten naar een datapacket dat we in een ClientDataSet kunnen gebruiken. Als we ook de weg terug nodig hebben (om het ClientDataSet datapacket weer terug om te zetten naar een XML document), dan zullen we ook de transformatie in de andere richting moeten maken. Dat gaat makkelijk, want het middenste deel van de XML Mapper bevat de "Transform Direction" opties, die default op "XML to Datapacket" staat, maar we ook op "Datapacket to XML" kunnen zetten. Ook daarna moeten we weer expliciet op de knop "Create and Test Transformation" drukken om een nieuwe transformatie te kunnen maken en testen. Het resultaat is een XML document (hetzelfde als in het linker deel van het scherm, maar dan met de data ingevuld). Hierna kunnen we weer verschillende zaken bewaren, zoals de transformatie informatie die deze keer in ToXml.xtr bewaard wil worden. Beide XTR bestanden hebben we nodig om de conversie van een XML document naar een datapacket en terug te kunnen realiseren. Sluit dan nu de XML Mapper af, en start Delphi 6 om de transformatie informatie bestanden toe te passen.

XMLTransformProvider

Het makkelijkst is om te beginnen met een nieuwe data module, aangezien we het XML document in feite omzetten tot een dataset. Allereerst hebben we de XMLTransformProvider nodig van de Data Access tab van het Component Palette. De TransformRead property heeft een subproperty TransformationFile, en deze moet naar de ToDp.xtr wijzen (om van het XML document een data packet te maken). En als we toch bezig zijn: de TransformWrite heeft ook een subproperty TransformationFile, en die moet uiteraard naar ToXml.xtr wijzen. Blijft er nog één property over die een waarde moet krijgen, en dat is de XMLDataFile property die naar het SESSIONS.xml input bestand zelf moet wijzen.



Screenshot #5

Met deze drie properties op hun plaats, is de XMLTransformProvider in staat om het XML input document te "voeren" aan een ClientDataSet component, en de eventuele updates daarvan weer terug te schrijven naar het XML document (dat in beide gevallen gevonden wordt via de XMLDataFile property).
We hebben maar liefst drie ClientDataSets nodig: eentje voor het resultaat van de XMLTransformProvider, en twee om naar de Location en Session nested datasets te kunnen verwijzen. De eerste noem ik ClientDataSetXML (omdat die het XML document representeert), en de andere twee resp. ClientDataSetLocation en ClientDataSetSession (zie ook screenshot #6).

Screenshot #6

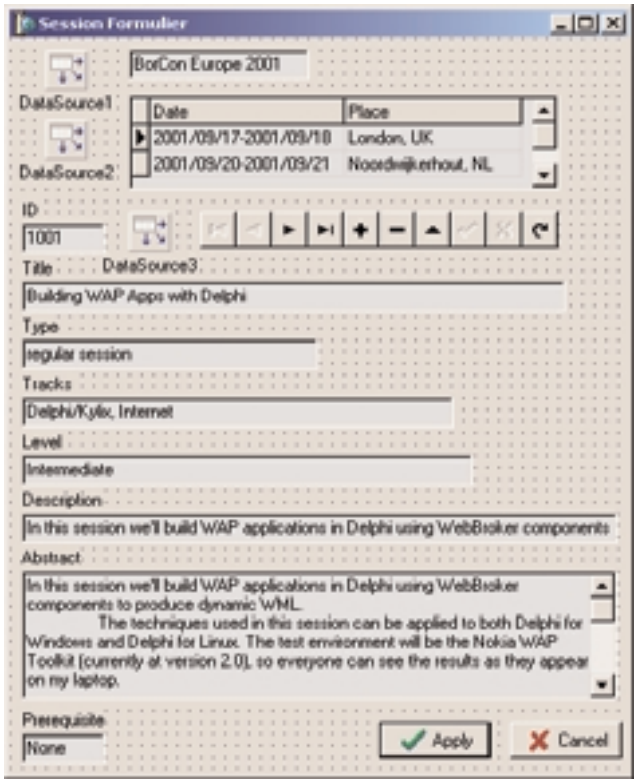
De ProviderName property van ClientDataSetXML moet wijzen naar XMLTransformProvider1. Hierna moet je met de rechter-muisknop klikken op ClientDataSetXML en de Fields Editor opstarten. We moeten expliciet alle velden toevoegen (met Ctrl+F) en daarmee persistent maken zodat de andere twee ClientDataSets ze kunnen vinden om mee te koppelen. We kunnen daarna voor de ClientDataSetLocation de DataSetField property de waarde ClientDataSetXMLLocation geven, en voor ClientDataSetSession de DataSetField de waarde ClientDataSetXMLSession geven. Van deze laatste twee ClientDataSets is het niet echt noodzakelijk om persistent fields te maken (ik doe het meestal wel, want het is wel makkelijk op direct met de velden zelf te werken).
Als we nu de ClientDataSetXML openen (of de Active property op True zetten) dan zal deze aan zijn provider om data vragen. De XMLTransformProvider zal deze data kunnen leveren door het XML document (verwezen in de XMLDataFile property) te transformeren van een XML document naar een data packet, volgens de transformatie informatie die in de TransformRead.TransformationFile aangetroffen wordt.

XML Formulier

Nu we via de XMLTransformProvider op het data module een XML document in feite kunnen splitsen in drie ClientDataSet componenten wordt het tijd om die te gebruiken op een beetje zinvolle manier. Ga terug naar het main form van je project, en zorg dat de data module in de uses clause is opgenomen.

Wellicht kunnen we over een jaar of wat een XML schema krijgen als input, en dan een ingevuld XML document opleveren als output

We hebben nu om te beginnen drie DataSources nodig op ons main form; een voor iedere ClientDataSet in de data module. Van de DataSource die naar ClientDataSetXML wijst gebruik ik alleen het "Name" veld (de andere twee, Location en Session zijn immers geneste datasets). Voor de DataSource die naar ClientDataSetLocation wijst gebruik ik een DBGrid om de velden Date en Place te laten zien. En tenslotte wil ik alle velden van de ClientDataSetSession laten zien in DBEdit controls, met uitzondering van het Abstract veld dat in een DBMemo vertoond moet worden. Een DBNavigator verbonden met de DataSource van de ClientDataSetSession zorgt er bovendien voor dat we makkelijk door de verschillende abstracts heen kunnen lopen.
Als laatste wil ik nog twee buttons toevoegen. Eentje om eventuele wijzigingen die we in het formulier hebben aangebracht terug te sturen naar het oorspronkelijke XML document, en eentje om eventuele foutieve wijzigingen ongedaan te maken en de oorspronkelijke data uit het XML document weer terug te krijgen.



Screenshot #7

De twee buttons zullen de enige plaats zijn waar we enige code moeten schrijven - en zelfs die is niet lang. Voor het doorsturen van de gewijzigde gegevens moeten we ApplyUpdates van de ClientDataSet aanroepen. En wel bij de ClientDataSetXML - de "moeder" van de twee geneste ClientDataSets. De ClientDataSetXML stuurt het delta data packet vervolgens door naar zijn provider, wat in dit geval de XMLTransforProvider is. Deze zal de transformatie van data packet terug naar XML document maken volgens de transformatie informatie die in de TransformationFile subproperty van TransformWrite wordt aangetroffen. De code voor de ApplyButton is dan ook als volgt:

```
procedure TForm1.BtnApplyClick(Sender: TObject);
begin
  (DataSourceXML.DataSet AS TClientDataSet).ApplyUpdates(-1)
end;
```

Overigens moeten we de unit DBClient ook aan de uses clause van ons main form toevoegen (anders kent de compiler TClientDataSet niet). Om te testen kun je bijvoorbeeld de waarvan de het Prerequisite veld van de eerste session abstract wijzigen van "None" in "n/a", en daarna op de Apply button drukken. Op het eerste gezicht is er niks veranderd, maar als je de toepassing afsluit en opnieuw opstart zie je dat je nu plotseling het tweede session abstract (met ID 1002) als eerste te zien krijgt. De abstract die we zojuist hebben gewijzigd is achteraan de lijst terecht gekomen! Mocht dit een probleem zijn (stel dat

je de data in de ClientDataSet altijd op ID gesorteerd wilt hebben), dan kun je dit oplossen door ID op te geven als waarde voor de IndexFieldName property van de ClientDataSetSession component. De laatste regel code moeten we schrijven voor de Cancel button, die alle wijzigingen gedaan in het XML formulier weer ongedaan maakt - en dus in feite de data uit het oorspronkelijke XML document weer ophaalt. Dat laatste is niet nodig, want de ClientDataSet zelf heeft een methods "CancelUpdates" genaamd die we hiervoor kunnen gebruiken. De code voor de CancelButton is dan ook als volgt:

```
procedure TForm1.BtnCancelClick(Sender: TObject);
begin
  (DataSourceXML.DataSet AS TClientDataSet).CancelUpdates
end;
```

Tot slot nog een paar regels code om te controleren of er nog wijzigingen zijn gemaakt die nog niet opgeslagen zijn. Ook hier kunnen we de ClientDataSet zelf voor gebruiken, en controleren of ChangeCount een waarde groter dan 0 heeft. Zo ja, dan moeten we even vragen of de wijzingen bewaard moeten worden. In de OnClose van de main form schreef ik hiervoor de volgende code:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if DataSourceXML.State in [dsEdit,dsInsert] then
    DataSourceXML.DataSet.Post;
  if (DataSourceXML.DataSet AS TClientDataSet).ChangeCount > 0 then

    if MessageDlg('Veranderingen bewaren?',
      mtConfirmation, [mbYes,mbNo], 0) = mrOK then
      (DataSourceXML.DataSet AS TClientDataSet).ApplyUpdates(-1)
end;
```

Merk op dat ik eerst nog even de "State" van de DataSourceXML bekijk om te zien of de laatste wijzigingen (in het huidige record) al opgeslagen zijn.
Mocht er gekozen worden voor het opslaan van de wijzigingen (of direct op de Apply knop gedrukt worden), dan vinden we de nieuwe inhoud in het XML document. In feite doet het XML document dienst als database tabel. Maar het voordeel is natuurlijk dat we dit XML document (met of zonder het XML schema bestand) uit een andere bron hadden kunnen ontvangen, of naar een andere bestemming kunnen sturen. De XML Mapper stelt ons in ieder geval in staat om een XML document te gebruiken alsof het een dataset is, en dat werkt in Delphi nu eenmaal erg makkelijk.

Bob Swart - Delphi Trainer en Consultant

Bob Swart is een freelance schrijver, trainer en webmaster van Dr.Bob's Delphi Clinic te http://www.drbob42.nl

tools

- ✔ Internet
- ✔ Java
- ✔ .NET

auteur

Mart van Ineveld

Webservices, Java en interoperabiliteit!

Inleiding

*Webservices op basis van XML is een nieuwe architectuur voor onder meer Business to Business (B2B) samenwerking. Webservices kunnen op verschillende platforms worden geïmplementeerd en met behulp van verschillende producten. De nieuwe .NET architectuur van Microsoft maakt bijvoorbeeld intensief gebruik van webservices en webservices vormen tevens een integraal onderdeel van de bijbehorende Visual Studio .NET ontwikkelomgeving. Ook in Java kunnen webservices kunnen worden geimplementeerd. Hierbij wordt naadloos aangesloten op de bewezen Java 2 Enterprise Edition (J2EE) architectuur. Java en XML passen perfect bij elkaar: **portable code** en **portable data**. Dit artikel concentreert zich op de toepassing van Webservices in een Java omgeving, waarbij ook is gekeken naar interoperabiliteit tussen de verschillende platforms.*

Wat zijn webservices

Een webservice kent drie rollen. In goed Nederlands: de *service provider*, de *service requester*, en de *service broker*. Daarbij zijn hoofdzakelijk drie protocols betrokken:

- Simple Object Access Protocol (SOAP) voor de interactie tussen de provider en requester. Het daadwerkelijk aanroepen van de service waarbij de naam van de service, de method en de parameters in een XML document zijn beschreven. Als transportlaag wordt HTTP(S) of SMTP gebruikt.

- Universal Description, Discovery, and Integration (UDDI): Een standaard voor het publiceren van en zoeken naar webservices. Het telefoonboek.

- Webservices Description Language (WSDL): de gestandaardiseerde beschrijving van de syntax en semantiek van de webservice in de vorm van een XML document.

SOAP

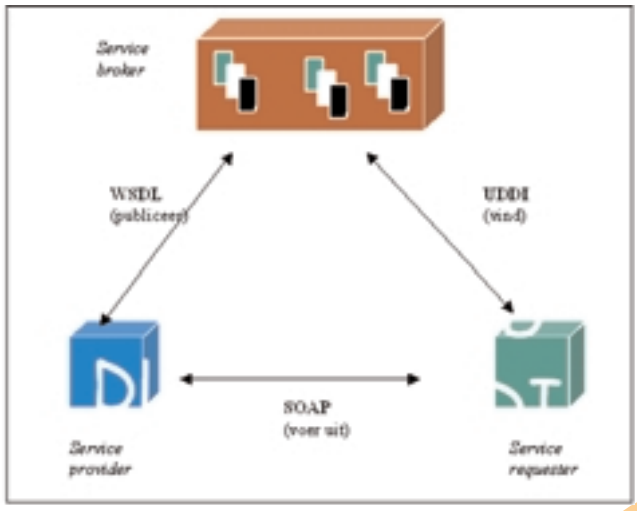
SOAP is de kern van webservices: het verpakken van een service call en de response in XML over HTTP. Dit protocol is voldoende om een service te executeren, ervan uitgaande dat de requester op de hoogte is van de exacte syntax en semantiek van de service die de provider biedt. SOAP standaardiseert service requests en responses als XML-document, waarbij verschillende transportprotocols kunnen worden gebruikt, zoals HTTP(S).
De kracht ontleent SOAP aan drie factoren:

1. Het gebruik van XML: platform- en taal-onafhankelijk, uitbreidbaar (de X), en massaal ondersteund door de ICT industrie.
2. Het gebruik van HTTP: een eenvoudig Internet-protocol dat op alle platforms standaard beschikbaar is en geen last heeft van firewalls.
3. De eenvoud van SOAP: een ongecompliceerd protocol dat gemakkelijk te implementeren is.

Zie als concreet voorbeeld een SOAP request over HTTP voor de method "echoInteger()":

```
POST /asmx/simple.asmx HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 469
SOAPAction: "http://soapinterop.org/echoInteger"

<?xml version='1.0' encoding='UTF-8'?>

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Body>

<ns1:echoInteger
xmlns:ns1="http://soapinterop.org/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<inputInteger xsi:type="xsd:int">5</inputInteger>
</ns1:echoInteger>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

De HTTP header bevat informatie over de aangeroepen service. De inhoud van het HTTP POST request bestaat uit een XML document conform de SOAP standaard. Hierin zijn de method naam (echoInteger), de parameter (inputInteger) en de waarde van de parameter (5) gemarkeerd. Een soortgelijke boodschap wordt door de provider teruggestuurd met de response. Dat kan het resultaat van de method zijn of een foutmelding.



UDDI

UDDI is het telefoonboek voor webservices. Providers kunnen hun service publiceren binnen het UDDI project. Services zijn daarin onderverdeeld -op Amerikaanse leest geschoeid- in:

- White pages (business)
- Yellow pages (per onderwerp gegroepeerd)
- Green pages (type service)

De wijze waarop services gepubliceerd kunnen worden is in een XML-schema vastgelegd. Deze zoekservice geeft echter geen richtlijnen voor de wijze waarop de syntax en semantiek van de service beschreven moeten worden.

WSDL

Daartoe dient het eveneens bij het W3C-consortium neergelegde voorstel voor beschrijving van de exacte syntax van een webservice: Webservice Description Language. WSDL bevat in XML-vorm de volgende elementen:

- Een beschrijving van het formaat van de typen en messages die een request/response kan bevatten.
- De semantiek van de boodschap: request-only, request-response, of response-only.
- De syntax van de aanroep van de service: over HTTP/HTTPS etcetera.
- De URL van de service.

Evenmin als UDDI is WSDL nodig voor het aanroepen van een webservice. Deze twee standaarden (of voorstellen daartoe) zijn alleen nodig voor het vinden van een webservice en voor een beschrijving van de juiste aanroep in gestandaardiseerde vorm. Deze beschrijving kan natuurlijk ook op andere wijze worden gegeven, desnoods in een email. Voor het daadwerkelijk aanroepen van een webservice is een API nodig.

Standaard API's in Java

SOAP en J2EE

SOAP maakt een integraal onderdeel uit van de .NET ontwikkelomgeving. Het is daarmee bijna kleuterspel om een webservice te implementeren. Momenteel is dat in Java want minder gemakkelijk, met name omdat de ontwikkelomgevingen, zoals bijvoorbeeld Borland's Jbuilder, nog niet vooruitlopen op de in ontwikkeling zijnde standaarden. Een bedrijf als Microsoft heeft veel meer controle over het uitontwikkelen van intern gedefinieerde standaarden en het tegelijkertijd uitbrengen van een *state of the art* ontwikkelomgeving. Dit neemt niet weg dat een webservice minstens zo goed in Java geïmplementeerd kan worden, alleen kost dat wat meer kennis en moeite, wat tot bijkomend voordeel heeft dat de ontwikkelaar heel goed weet wat hij doet. Bovendien sluit SOAP naadloos aan op de in vergelijking met .NET veel verder uitontwikkelde J2EE server-standaard, waarin entity-objects via session-objects en servlets aan SOAP webservices gekoppeld kunnen worden. Stuk voor stuk reeds in de praktijk bewezen technologieen.

Een implementatie van een webservice in J2EE kan er als volgt uitzien:

1. Een SOAP request wordt door de HTTP server geleid naar de J2EE servlet die op de URL respondeert.
2. De servlet vertaalt het (SOAP-)XML request via een unmarshalling functie naar de gewenste method met parameters van een Enterprise Java Bean (EJB), meestal een session bean.
3. Deze session bean executeert het request in de door een application server beheerde omgeving. Hierbij kunnen ook externe systemen worden aangeroepen, of andere webservices.
4. Het resultaat van de bewerking wordt teruggegeven aan de servlet, die een marshalling functie aanroept voor het vertalen van het resultaat uit de EJB naar (SOAP-)XML.
5. De servlet retourneert dit XML resultaat via HTTP naar de aanroepende client.

JAX Pack

Dit scenario kan uiteraard geheel worden geïmplementeerd in zelf ontwikkelde Java classes. Maar binnen het Java Community Process (JCP) zijn al diverse standaarden uitgekristalliseerd, of in een vergevorderd stadium. Deze standaarden rond Java en XML (SOAP in ruime zin, inclusief WSDL en UDDI) worden samengevat het JAX Pack genoemd: Java API for XML. Dit JAX-Pack bevat de volgende onderdelen:

- **JAXP:** Java API for XML Processing. Deze API kan inmiddels worden gedownload van de Sun Java site. Evenals JDOM (een initiatief van verschillende Java ontwikkelaars, zoals SAX) bevat JAXP wrapper-classes voor het verwerken en transformeren van XML-documenten op basis van SAX, DOM en XSLT.
- **JAX-RPC:** Java API for XML Remote Procedure Calls.
- **JAXM:** Java API for XML Messaging. Deze standaard kan samen met de vorige worden gebruikt voor het in Java aanroepen of verwerken van SOAP-messages of requests (RPC).
- **JAXR:** Java API for XML Registry. Deze API standaardiseert het uitvragen van webservice registries op basis van UDDI of ebXML (eveneens een in ontwikkeling zijnde registratiestandaard voor webservices).
- **JWSDL:** begint niet met JAX: een standaard voor het uitgeven en importeren van WSDL-documenten in Java. Deze API is eveneens nog in de standaardisatiefase. Het JAX-pack bevat meer dan de hier genoemde API's (zoals JAXB), maar die zijn in het kader van SOAP van minder belang.

Dit JAX-Pack bevat daarmee een complete API voor het verwerken van XML, en meer specifiek SOAP/WSDL en registries in Java.

Huidige implementaties

Het meest dringende probleem bij deze laag API's is dat ze nog niet zijn uitontwikkeld, en dus ook nog niet allemaal beschikbaar zijn. De huidige Java implementaties van SOAP zijn daarom niet compleet (alleen SOAP, eventueel uitgebreid met WSDL), of wijken als *early adopters* af van het uiteindelijke resultaat van het standaardisatieproces.

Momenteel bestaan onder meer de volgende Java implementaties voor webservices:

- **Apache SOAP:** GNU public license open source software. De huidige versie ondersteund de SOAP standaard zonder UDDI of WSDL.
- **IDOOX:** deze firma geeft WASP uit (Web Applications and Services Platform). Een zogenaamde lite version van WASP is vrij te gebruiken. Deze versie kan worden geintegreerd met Jbuilder of Sun Forte for Java, en automatiseert zaken als het genereren van een WSDL document voor de beschrijving van de webservice in Java. Ervaring met WASP Lite versie 2.0 leert echter dat de gegenereerde WSDL file niet compatibel is met Microsoft .NET beta 2.
- **GLUE** is een 100% Java implementatie van SOAP, WSDL en UDDI van The Mind Electric (een firma in Texas). Een enigszins beperkte versie is vrij te downloaden.

Interoperability: proef op de som

Het belangrijkste kenmerk van webservices is dat ze interoperabel zijn: platform- en taal-onafhankelijk. Daarom is de proef op de som genomen op basis van de volgende configuraties:

- Tomcat versie 4.0 en Apache SOAP 2.2 op een Red Hat 7.1 Linux server (Intel)
- Microsoft .NET beta 2 op Windows 2000 professional.

Als testservice is een bestaande interoperabilitytest gekozen. Zie http://www.mssoapinterop.org/ voor een volledige specificatie van deze test.

Tomcat en Apache SOAP

Deze combinatie van open source Java componenten kan eenvoudig worden gedownload en volgens de read me's worden geinstalleerd. Bij Apache Soap worden ook verschillende voorbeelden meegeleverd, die na installatie als test kunnen fungeren.

Apache SOAP fungeert in grote lijnen als volgt:

- Apache levert een soap.war (web application archive) aan met daarin onder meer een servlet die al eerder beschreven SOAP-requests vertaalt naar een aanroep van een Java class. Deze war-file kan in de omgeving van Tomcat worden gekopieerd, waardoor deze automatisch wordt gedeployed door Tomcat. De soap.jar file in dit pakket bevat alle classes voor executie van de voorbeeld webservices. Deze classes kunnen uiteraard worden uitgebreid of vervangen met eigen implementaties.
- Deployment van een webservice vindt plaats door middel van een deployment descriptor (via een management web servlet of door middel van XML documents). Deze descriptor beschrijft de service in een Apache-only formaat. Dus niet in WSDL vorm. Het aardige van deze methode is echter dat in deze descriptor verschillende deployments kunnen worden beschreven. Zo kan ook een beschrijving worden gemaakt van een service die direct in een EJB is geimplementeerd, of geheel buiten Java in een BSF script. De Apache SOAP servlet roept dan direct deze EJB of dat script aan. Deze Java class of EJB moet uiteraard beschikbaar zijn in het classpath van de Apache SOAP servlet.

Het belangrijkste kenmerk van webservices is dat ze interoperabel zijn

- De deployment descriptor bevat ook "type mappings": koppelingen van SOAP types (de parameters of return types van request en response) naar classes die de waarden vertalen naar Java en terug. Voor veel types, zoals int, double en dergelijke, levert Apache standaard mappings en classes (serializers en deserializers genoemd). Voor echte Java Beans bestaat ook een (de)serializer (Een Java Bean is een Java class die aan specifieke voorwaarden voldoet, zoals een get- en een set-funtie voor alle instance-variabelen, en een constructor zonder argumenten). Maar voor andere typen moet zelf een (de)serializer in Java met bijbehorende mapping in XML worden geschreven.

De deployment descriptor zorgt er dus voor dat de SOAP servlet weet welke class aangeroepen moet worden voor de SOAP service, en welke classes moeten worden gebruikt voor de marshalling van SOAP XML naar Java en terug. De aangeroepen class kan voorts de business layer aanroepen voor het uitvoeren van de bedoelde functionaliteit.

De webservice in Apache SOAP

De interoperabilityservice is simpel: de meegegeven parameter wordt als return type teruggegeven. Zie bijvoorbeeld de buitengewoon eenvoudige serverfunctie:

```
public int echoInteger(int i)
 {
   return i;
 }
```

De SOAP aanroep van deze functie is in het begin van dit artikel reeds getoond.
Op soortgelijke wijze worden zo verschillende typen geechood. Onder meer een float, een string, een eigen struct (met integer, float en string), en arrays van al deze typen. Kern van de interoperabiliteitstest is immers het correct (un)marshallen van alle typen. De rest van de honneurs wordt waargenomen door XML over HTTP, en die protocols hebben zich inmiddels wel bewezen.

Deployment van de webservice kan vanaf de shell met behulp van de deploy-functie van de ServiceManagerClient van Apache:

```
>java org.apache.soap.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter deploy
DeploymentDescriptor.xml
```

Het bestand DeploymentDescriptor.xml beschrijft de service-functies en de type-mappings. Het opvragen van de lijst met reeds "gedeployde" services toont aan dat alles in orde is. De service heeft als urn "http://soapinterop.org":

```
>java org.apache.soap.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter list
```

```
Deployed Services:
                http://soapinterop.org/
                urn:AddressFetcher2
                urn:xmltoday-delayed-quotes
```

Een lokale test, met een specifiek voor deze service geschreven Java client, bewijst dat ook het uitvoeren van de functie goed verloopt:

```
>java samples.interop.EchoTestClient
http://localhost:8080/soap/servlet/rpcrouter
echoInteger       OK
echoFloat         OK
echoString        OK
echoStruct        OK
echoIntegerArray  OK
echoFloatArray    OK
echoStringArray   OK
echoStructArray   OK
```

Client in C#

Voor de test is een al even eenvoudige C#-client gebruikt in .NET:

```
Console.WriteLine("Connect to bunker.xs4all.nl...");
SimpleTest service = new SimpleTest();
Console.WriteLine("echoInteger");
int iTest = service.echoInteger(123);
Console.WriteLine("Result: " + iTest);
```

De C#-class SimpleTest is verkregen met de .NET utility WSDL.EXE. Dit utility genereert een C#-proxy op basis van een WSDL-file met de beschrijving van de functie. Indien deze WSDL-file voldoet aan de syntax die Microsoft ondersteund, dan wordt een C#-file gegenereerd met een class voor aanroep van de webservice. Bijvoorbeeld het volgende fragment is gebruikt voor aanroep van de juist getoonde Java functie als webservice:

```
[System.Web.Services.WebServiceBindingAttribute(
Name="SimpleTestSoap", Namespace="http://soapinterop.org/")]
[System.Xml.Serialization.SoapIncludeAttribute(
typeof(SOAPStruct))]
public class SimpleTest :
System.Web.Services.Protocols.SoapHttpClientProtocol {
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    public SimpleTest() {
        this.Url =
"http://bunker.xs4all.nl:8080/soap/servlet/rpcrouter";
    }

    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.Web.Services.Protocols.SoapRpcMethodAttribute(
"http://soapinterop.org/",
RequestNamespace="http://soapinterop.org/",
ResponseNamespace="http://soapinterop.org/")]
    [return:
System.Xml.Serialization.SoapElementAttribute("return")]
    public int echoInteger(int inputInteger) {
        object[] results = this.Invoke("echoInteger", new object[] {
                    inputInteger});
        return ((int)(results[0]));
    }
(….)
```

Wat opvalt aan deze gegenereerde C#-functie is dat de URL van de webservice hard gecodeerd is in de constructor. Normaliter wordt een URL geparametriseerd, en uitgebreid met fallback URL voor het geval deze service niet beschikbaar is.

Het aardige aan de Microsoft implementatie van een webservice client is dat impliciet asynchrone functies beschikbaar zijn voor aanroep van de service. De functie echoInteger() kan asynchroon worden aangeroepen met een zogenaamde C#-delegate als callback handler. De executie van de client wordt dan niet gestopt door aanroep van een webservice.

Resultaten

De resultaten van de interoperabilitytest zijn veelbelovend: hoewel alle standaarden nog convergeren werkten de beschreven testfuncties (echoInteger(), echoFloat(), echoStruct()...) vanuit een C# client naar een Java webservice zonder noemenswaardige problemen.

Vanuit Java naar de C#-service gaf problemen bij de array-functies (echoIntegerArray() enzovoorts). Gelukkig bevat het Apache SOAP pakket ook een utility om de HTTP-POST en GET-messages te bekijken, en dankzij dit utility werd snel duidelijk dat het probleem gemakkelijk kon worden ondervangen door het zelf schrijven van een (de)serializer class voor arrays.

Deze utility tunnelt een HTTP POST/GET van een bepaalde *port* van de client machine naar de opgegeven port van de webservice en vice versa. In twee panelen wordt het request en het response weergegeven, zodat snel zichtbaar wordt wat er mankeert:

```
>java org.apache.soap.util.net.TcpTunnelGui listenport
tunnelhost tunnelport
```

of concreet:

```
>java org.apache.soap.util.net.TcpTunnelGui 8081
www.mssoapinterop.org 80
```

De marshalling van arrays die terugkeren van de .NET service blijkt fout te gaan: de Apache SOAP implementatie herkent de type-aanduiding binnen een array niet. Met een aangepaste type-mapping in de Java-client, eventueel aangevuld met eigen deserializer classes, kan dit probleem verholpen worden.

Conclusie

Webservices zijn een integraal onderdeel van de .NET visie van Microsoft. Sun's Java ONE initiatief ijlt wat na, maar de standaardisatie in Java van alle API's rondom SOAP functies is zeer ver gevorderd, en ten dele al beschikbaar. De huidige implementaties in Java zijn werkbaar, maar wellicht is het beter even te wachten tot het JAX-Pack is uitgekristalliseerd over enkele maanden.
Zowel .NET als Java bieden met webservices een platform-onafhankelijke omgeving die interoperabel is. Nu nog een ontwikkelomgeving voor Java die de Microsoft Studio's evenaart.

Internet resources

Overzicht van de belangrijkste links:

- http://www.w3.org/TR/SOAP/
- http://www.idoox.com
- http://www.themindelectric.com/products/glue/glue.html
- http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnsoap/html/soapinteropbkgnd.asp
- http://www.whitemesa.com/interop.htm
- http://soap.weblogs.com/validator1
- http://www.perfectxml.com/articles/xml/soapguide.asp
- http://www.xmethods.net/ilab

Mart van Ineveld
Software Consultant Omnext.NET B.V.
(www.omnext.net)
Omnext specialiseert zich in het ontwikkelen van webapplicaties en het integreren van nieuwe applicaties met bestaande systemen.
Voor vragen, mail naar: mart.van.ineveld@omnext.net

Delphi Tip

Je kunt je veel code besparen door actions te gebruiken

Open de action-editor door dubbel-klik op het action component, typ <Ctrl><Insert> voor het toevoegen van standaard acties, zoals dataset, edit, help, window, etc.
Vaak kun je beter een actie gebruiken in plaats van een UDF (User Defined Function).
Acties kun je via de action property aan diverse componenten koppelen.
Verschillende properties (hint, caption, image) worden automatisch overgenomen. Met de execute methode kun je de actie gemakkelijk starten.

auteur
Les Pinter

Object Oriented Programming - a Primer

Object-oriented programming looks a lot different from the procedural code that we used to write. But it's not harder. In fact, it's easier. Let's start with a simple example:

```
* MAIN.PRG
X = CREATEOBJECT ( "MyClass" )
X.Show
X.Info = "Goodbye"
X.Show
RELEASE X

DEFINE CLASS MyClass AS Relation
Name = "MyRel"
Info = "This is the initial value to display"
PROCEDURE Show
? THIS.Info
? REPL("-",20)
ENDPROC
ENDDEFINE
```

Now, DO MAIN to run the example:
This is the initial value to display

Goodbye

The class has a property, info, and a method, show. To instantiate an object x based on the class, you use x = CREATEOBJECT ("classname"). To assign the property info a value, you use objectname.propertyname = value. To call a method, you use objectname.methodname.

What's in a name?

Methods are functions and procedures. There's nothing magic about methods. You can call a method m belonging to an object o using the syntax "o.m" any time you would have heretofore used DO m. In fact, you're supposed to. DO <programname> is pretty much passé.
Properties are variables. Except that they're sort of PUBLIC and LOCAL at the same time. Now, instead of using *FOR I = 1 TO N* and finding out two hours later that I was redefined somewhere inside the *FOR...NEXT* loop, you can use *FOR o.I = 1 TO o.N* and be sure. Goodbye to accidentally redefined global variables.

But there are a lot more reasons to use objects besides this. Classes are self-contained. They group all of the local variables (properties) that the functions and procedures (methods) need into one location together with the methods that use them. They compartmentalize tasks and make them easier to manage and debug. They ensure that variables (properties) stay *in scope* while they're needed. Objects and the coding techniques they require do a good job of the things that coding standards used to do badly: *They make it harder to write bad code!* Sidebar: Why did I use "AS RELATION"?. Here's why: Every one of FoxPro's standard classes has a predetermined set of properties, events and methods. You can see for yourself by typing HELP <classname> - e.g., HELP FORM or HELP Label, then following the Properties, Events or Methods hyperlinks; or you can open a form, drop the obscure object of your desire on it, and use the Property Inspector. Some classes (e.g., label) require a container, so you can't use CREATEOBJECT - you have to use ContainerName.AddObject, or add them visually. Others, like CONTAINER, CUSTOM and RELATION, don't need a container. And RELATION is tiny, especially if you assign it a name (who knows why. Steven Black discovered that one, naturally.)

DO <programname> is pretty much passé

To any object that you create you can add as many properties and methods as you want. You can also assign values to its built-in properties, and add code to its existing events and methods. *Your code will be executed in addition to the normal code of the method, unless you include the NODEFAULT keyword.* If you create an object based on one of your classes rather than on a FoxPro base class and then add code to an event or method, your code will be executed instead of the code you placed in your original class, *unless you include the DODEFAULT() function call.*

A technical aside

When you use the createobject function, the first parameter is the class name. The class can either be located inside a .VCX classlib, in which case you must have loaded its table of contents with the SET CLASSLIB TO <name> statement or the SET PROCEDURE TO <proclib (prg file).name> statement. As seen above, you can also define classes below a program that uses them, in the same place where functions and procedures ordinarily live. But additional parameters can be passed to the class as it's being created. They will be intercepted by the PARAMETERS statement that's located in the first line of the object's *Init* method. So if you add this, above

```
PROCEDURE Init
PARAMETERS NewInfo
THIS.Info = NewInfo
ENDPROC
```

then call it with this
x = CREATEOBJECT ("MyClass", "Yo, dude")
the message passed as a parameter at the time of object creation will be the first one displayed. Note that, if you include the statement RETURN .F. in the Init code of a class, it isn't instantiated. This is often used in forms or in the Data Environment of reports when required conditions aren't met.

What's THIS?

There are several new words that are used in class code to make it easier to write generic code. They are THIS, THISFORM, and PARENT. They allow you to refer to other members of a class, or of the class's parent, without knowing the name with which it was instantiated.
This is a little different from what we used to do. To call a function in FoxPro, you just use its name, e.g.
ParseIntoWords(Phrase)
However, this syntax assumes that *ParseIntoWords* is either a standalone PRG or FXP, or a defined function or procedure either inside the current program, a program that called it, or a procedure library previously opened with SET PROCEDURE TO. When you call a method of a class, you write both the object name and the method name, e.g.
MyObjectBasedOnMyStringsClass.ParseIntoWords (Phrase)
It's customary to use an objectname that's similar to the classname but begins with o (for object):
oStrLib = CREA ("MyStringsClass")
oStrLib.ParseIntoWords (Phrase)
To refer to a property or method within the same object, the objectname prefix is required. If you try to just type MethodName and assume that it will look in the last object you created, it won't. But since you don't know the name with which the class will be instantiated, you can just use "THIS" as the prefix. For example, to call the WriteString method from another method in the same class, use THIS.WriteString. If it uses a property from the same class as a parameter, use THIS.WriteString (THIS.String).
For example, I create my own little library of classes for the FoxPro form controls that I use most often. In order to make it easier for the user to see where the cursor is as he/she moves from field to field, my textbox class looks like this:

```
DEFINE CLASS MyText AS TextBox OF MyLib
FontFace = "Courier New"
FontSize = 9
Enabled = .F.

PROCEDURE GotFocus
THIS.ForeColor = RGB ( 255,255,255 )
THIS.BackColor = RGB ( 255,  0,  0 )
ENDPROC

PROCEDURE LostFocus
THIS.ForeColor = RGB (   0,  0,  0 )
THIS.BackColor = RGB ( 255,255,255 )
ENDPROC

ENDDEFINE
```

Arrays as collections

In some languages, you can create object containers, then add things to them. They're like arrays, but without using subscripts. This allows the use of the following syntax
FOR Each ThingName in Collection
  ? ThingName.Value
ENDFOR
You can do something similar with FoxPro. Take this example:

```
DIMENSION X(2)
X(1) = 3
X(2) = 5
FOR EACH Thing in X
  ? Thing
ENDFOR
But we can also have arrays of objects:
DIMENSION X(2)
x(1) = CREA("Dubya", 3)
x(2) = CREA("Dubya", 5)
FOR EACH Texan in X
  ? Texan.Value
ENDFOR

DEFINE CLASS Dubya AS RELATION
Name = "Dubya"
Value = 0
PROCEDURE Init
PARAMETERS V
THIS.Value = V
ENDPROC
ENDDEFINE
```

PRG files containing classes

You can store class definitions either in VCX or in PRG format. If you use VCXs, you can drag and drop objects based on classes in the Form designer. Classes stored in PRGs can only be instantiated through code. They do, however, instantiate faster, and if there's no particular reason for them to be visually displayed, it's a good way to go.
Randy Brown, Calvin Hsia and Matt Oshry have written a set of comprehensive wrapper classes for the Registry. Open registry.prg (probably located in your \Visual Studio\MSDN98\98VSA\1033\SAMPLES\VFP98\CLASSES directory) and look for PROCEDURE. There are dozens, and they cover many if not most of the reasons you'd ever want to use the Win32API. This is an excellent example of a class library that provides ("exposes", if you like their terminology - I don't) functionality available in the Windows OS libraries. You simply write

```
SET PROCEDURE TO Registry
```

Then, depending on which one you need,

```
oReg = CREATEOBJECT ( "Registry" )
oFileReg = CREATEOBJECT ( "FileReg" )
oODBCReg = CREATEOBJECT ( "ODBCReg" )
oFoxReg = CREATEOBJECT ( "FoxReg" )
oINI = CREATEOBJECT ( "oldinireg" )
```

For example, this will find the versions of WinZip on my computer:

```
SET PROCEDURE TO REGISTRY
oIni = CREA ( "oldinireg" )
oIni.LoadIniFuncs()
lcSection = "WinZip"
lcEntry   = "win32 version"
lcIniFile = "C:\WINDOWS\WIN.INI"
lcValue     = ''
oIni.GetINIEntry ( @lcValue, lcSection, lcEntry, lcINIFile )
? lcValue
```

Forms as classes

You probably know that forms are stored in SCX/SCT tables with exactly the same structure as classes, which are stored in VCX/VCT tables. This may have already lead you to the startling conclusion that forms are classes, which they are. This is why you can change form properties on the fly using, for example, THISFORM.BackColor = RGB(192,192,192). THISFORM is a special reference that travels back up the object hierarchy until it comes to the container form. This was done because creating new methods and properties in a form is an ongoing process, and doing it all in code would have created a tech support hell for Microsoft. Note that the Class Browser written by Ken Levy gives you many of the same capabilities for any kind of class.

A very common type of form class is the "flat file editor" class. We'll build one below and show in the process by which classes are often built on the fly while building forms and PRGs.

Say I want to build a form to add, edit and delete records from a flat file named CUSTOMERS. I create a form, add six buttons across the bottom (for Add, Edit, Delete, Save, Cancel and Exit), and start to code the Add button:

```
cmdAdd:Click

SELECT ( "CUSTOMERS" )
=CursorSetProp("Buffering",3)
PUBLIC BeforeAdd
BeforeAdd = RECNO()     && We'll see later why we need this...
APPEND BLANK
THISFORM.Refresh
THISFORM.cmdAdd.Enabled    = .F.
THISFORM.cmdEdit.Enabled   = .F.
THISFORM.cmdDelete.Enabled = .F.
THISFORM.cmdSave.Enabled   = .T.
THISFORM.cmdCancel.Enabled = .T.
THISFORM.cmdExit.Enabled   = .F.
THISFORM.SetAll ( "MyText", "Enabled", .T. )
```

Right about now it dawns on my that all of my forms will need about the same stuff. By creating a class that does these things using generic code, we can write it once, then reuse it over and over again.

To begin with, I create a form class called FlatFileEditor in a class library called Forms:

```
CREATE CLASS FlatFileEditor AS Form OF Forms
```

I create and name the same six buttons, then enter the CLICK code of cmdAdd. I change the SELECT statement to refer to a property called MainTable, which will be initialized by the Programmer in the properties sheet of each form based on this class. To ensure that I don't forget, I write this code in the class's Init:

```
FlatFileForm.Init:

IF EMPTY ( THISFORM.MainTable )
   MessageBox("Programmer error: MainTable property not initialized")
   RETURN .F.
ENDIF
```

BeforeAdd holds the record number the user was looking at before clicking Add, If they cancel the add, I want to be able to return to where they were. It's a perfect candidate for a class property. Imagine if I had several forms open, and had stored the record number before adding a record in one table, then opened another form and stored its initial record number to the same global variable? That's exactly what tended to happen before we had the ability to use form properties that act like variables that are local to their form. So we'll add both TableName and BeforeAdd as properties of the FlatFileForm form class.

During adding, all buttons except Save and Cancel have to be disabled. The same thing is true while Editing, so I'll write it once and use it in both places. Create a form method called ButtonsOff and move all six lines of code there. While we're at it, create another form class method called ButtonsOn, copy the same six lines of code to it, and change all the T's to F's and vice versa. Now we can rewrite our class code for cmdAdd.Click as:

```
FlatFileForm.cmdAdd:Click

SELECT ( THISFORM.MainTable )
=CursorSetProp("Buffering",3)
THISFORM.BeforeAdd = RECNO()
APPEND BLANK
THISFORM.Refresh
THISFORM.ButtonsOff
THISFORM.SetAll ( "MyText", "Enabled", .T. )
We'll use WITH THISFORM to make the code more readable:
FlatFileForm.cmdAdd:Click

WITH THISFORM
  SELECT ( .MainTable )
  =CursorSetProp("Buffering",3)
  .BeforeAdd = RECNO()
   APPEND BLANK
  .Refresh
  .ButtonsOff
  .SetAll ( "MyText", "Enabled", .T. ) && This isn't too good...fix it.
ENDWITH
```

There's just one problem: The last line was meant to enable all of my input fields. However, I may have other types of objects on the screen besides textboxes. And I may add new types of input field classes from time to time, so I want something that's easy to maintain. I finally came up with this generic solution:

```
THISFORM.EnableInputFields:
FOR I = 1 TO THISFORM.ObjectCount
  IF LOWER(THISFORM.Objects(I).Class) $ THISFORM.InputClasses
    THISFORM.Objects(I).Enabled = .T.
  ENDIF
ENDFOR
```

THISFORM.InputClasses, a new property of the FlatFileForm class, contains (in lowercase separated by commas) the names of my classes that I want to enable for input on each form based on this class. If I add a new input class name, nothing breaks. The companion method, THISFORM.DisableFields, which is used in the Save and Cancel routines, is the same code with .T. changed to .F.

You should begin to see a pattern here. You create some code for a form, recognize its generic qualities, add properties to the underlying class to provide the "local public variables" needed to store data in one place and use it somewhere else, then move the code to a new method in the underlying class and replace the place where the code used to be with a call to the new method call. This happens over and over as you develop your classes.

A note on scope: Previously, if you wanted to store a value in one function and use it in another, you either had to PUBLIC it or define it prior to calling the routing in which it was assigned a value, so that it would remain "in scope" wherever it was used. This was more than a little confusing, since the places where the variable was used weren't obvious at the time it was created. By creating a property for a class, we know implicitly that it's used by the methods of that class. (It can also be seen and used by methods in other classes, unless you want to use a Protected or Hidden property. I've never used one.)

The finished FlatFileForm class

Here are the completed class Methods and Properties, and the Click code for the six buttons:

```
FORM CLASS PROPERTIES:

MainTable = ""
BeforeAdd = 0
InputClasses = "myedit,mytext,myspin,myradio,mycombo"

FORM METHODS:

FlatFileForm.Init:
IF EMPTY ( THISFORM.MainTable )
   MessageBox("Programmer error: MainTable property not initialized")
   RETURN .F.
ENDIF

THISFORM.EnableInputFields:
FOR I = 1 TO THISFORM.ObjectCount
IF LOWER(THISFORM.Objects(I).Class) $ THISFORM.InputClasses
   THISFORM.Objects(I).Enabled = .T.
ENDIF
ENDFOR

THISFORM.DisableInputFields:
FOR I = 1 TO THISFORM.ObjectCount
IF LOWER(THISFORM.Objects(I).Class) $ THISFORM.InputClasses
   THISFORM.Objects(I).Enabled = .F.
ENDIF
ENDFOR

THISFORM.ButtonsOff:
THISFORM.cmdAdd.Enabled    = .F.
THISFORM.cmdEdit.Enabled   = .F.
THISFORM.cmdDelete.Enabled = .F.
THISFORM.cmdSave.Enabled   = .T.
THISFORM.cmdCancel.Enabled = .T.
THISFORM.cmdExit.Enabled   = .F.

THISFORM.ButtonsOn:
THISFORM.cmdAdd.Enabled    = .T.
THISFORM.cmdEdit.Enabled   = .T.
THISFORM.cmdDelete.Enabled = .T.
THISFORM.cmdSave.Enabled   = .F.
THISFORM.cmdCancel.Enabled = .F.
THISFORM.cmdExit.Enabled   = .T.

CommandButton EVENT CODE:

FlatFileForm.cmdAdd.Click:
WITH THISFORM
  SELECT ( .MainTable )
  =CursorSetProp("Buffering",3)
  .BeforeAdd = RECNO()
   APPEND BLANK
  .Refresh .ButtonsOff
  .EnableInputFields
ENDWITH

FlatFileForm.cmdEdit.Click:
WITH THISFORM
  SELECT ( .MainTable )
  =CursorSetProp("Buffering",3)
  .BeforeAdd = RECNO()
  .ButtonsOff
  .EnableInputFields
ENDWITH
```

```
FlatFileForm.cmdDelete.Click:
WITH THISFORM
SELECT ( .MainTable )
IF MessageBox ( "Delete this record?" )
   BLANK  NEXT 1
   DELETE NEXT 1
   SET DELETED ON
   GO TOP
   .Refresh
ENDIF
ENDWITH

FlatFileForm.cmdSave.Click:
WITH THISFORM
  SELECT ( .MainTable )
  =TableUpdate(.T.) =CursorSetProp("Buffering",1)
  .ButtonsOn
  .DisableInputFields
ENDWITH
```

Here's what THISFORM.BeforeAdd was for:

```
FlatFileForm.cmdCancel.Click:
WITH THISFORM
  SELECT ( .MainTable )
  =TableRevert(.T.)
  =CursorSetProp("Buffering",1)
  IF BETWEEN ( .BeforeAdd, 1, RECCOUNT() )
    GO ( .BeforeAdd )
  ENDIF
  .ButtonsOn
  .DisableInputFields
ENDWITH

FlatFileForm.cmdCancel.Click:
THISFORM.Release
```

How to use the form class

To use this class, click on Tools, Options, Forms, enter the name of the new class as your Form Template and click on Set as Default. The next time you create a form in your project, if you don't select Form Wizard, it will be based on this form class. Add fields by opening the Data Environment and adding the table that's the main table for the form. (Don't forget to enter the table's name beside the MainTable property in the Property Sheet.)
After using Tools, Options, Field Mapping to tell FoxPro to use your textbox, label and other classes instead of the FoxPro base classes, drag the word "fields" from the top of the table in the DE and drop it in the upper left-hand corner of the form. Rearrange all of the fields as you need them.
Now, click on View, Tab Order from the menu, and select BY ROW. (Make sure that Interactive isn't selected on the Tools, Options, Forms page.) Finally, drag the cmdAdd and cmdEdit buttons to the top of the Tab Order list - you'll see why when you run the form. Bingo, you're done. All of your flat file editing screens are now coded, except for special code you add for each form that's a little different from form to form. That's where you would use DODefault() and a little extra code.
Here's a quick example of how DODEFAULT() is used. It's often necessary to populate a date field with today's date when a record is added. To do this, code the following in your cmdAdd.Click event code:

```
DODEFAULT()
REPLACE NEXT 1 InvDate WITH DATE()
THISFORM.txtInvDate.Refresh
```

This executes the class code, then the two additional lines of code.

Summary

I hope this has convinced you that building and using classes is a natural evolution of the same coding process we've always used, moving from the specific to the general case, but with the tools that we need to do it right. Notice, if you will, that we've managed to write an entire paper about object-oriented programming without analogies, without referring to programming as being like a telephone or a red ball, and without using the terms "polymorphism" or "encapsulation" or even "inheritance"

Les Pinter
www.LesPinter.com
les@LesPinter.com

After writing and marketing the Real Estate Guide, the first nationally-marketed templates for Lotus 1-2-3, Les discovered dBase II and became a database developer. He bought copy number 253 of FoxBASE, and started publishing a monthly newsletter about FoxBASE in 1989. The Pinter FoxPro letter was published for 10 years in the US and 4 years in Russia. Currently, Les continues to publish articles on FoxPro, VB and ASP at his website, www.LesPinter.com.

Les has been a speaker at numerous FoxPro conferences in the US, France, Spain, Mexico, Russia and Canada. He gives seminars in five languages, most recently at the Microsoft TechEd Mexico City. Pinter Consulting is in the process of opening a second office in Mexico City, and is incorporated in Mexico.

Nieuwe redacteur

Mark Blomsma, nieuwe redacteur met aandacht voor Internet-Development.

tools

- ✔ Delphi 6
- ✔ HTML

auteur

Ronald van Aken

IntraWeb – Product Review

Tijdens mijn bezoek aan de Borcon werd mijn aandacht getrokken door IntraWeb. Dit is een door Phoenix Business Enterprises (http://www.pbe.com) gemaakte Delphi tool. In deze review zal ik zowel de technische- als de functionele kant bespreken van dit uiterst originele product.

Wat is Intraweb?

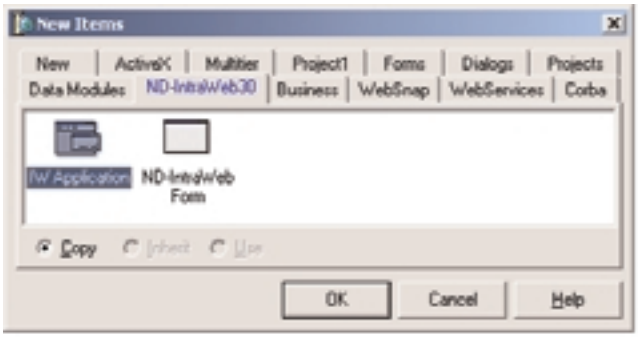
Intraweb is een tool die het mogelijk maakt om Delphi applicaties te renderen in HTML 4.0. De applicatie kan dus, zoals u dat gewend bent, in Delphi gemaakt worden waarna hij vervolgens te zien is in een webbrowser. Er wordt pure HTML code gegenereerd, zonder dat Java of ActiveX wordt gebruikt. Dit heeft als voordeel dat er geen speciale plug-in of andere software door de client gedownload hoeft te worden om het programma te zien.

Belangrijke eigenschappen van Intraweb zijn:

- Bouwen van web applicaties zoals je normaal applicaties maakt in Delphi.
- Geen kennis van HTML code nodig.
- Werkt op zowel Delphi als Kylix.
- Maakt geen gebruik van ActiveX, DCOM of Java.

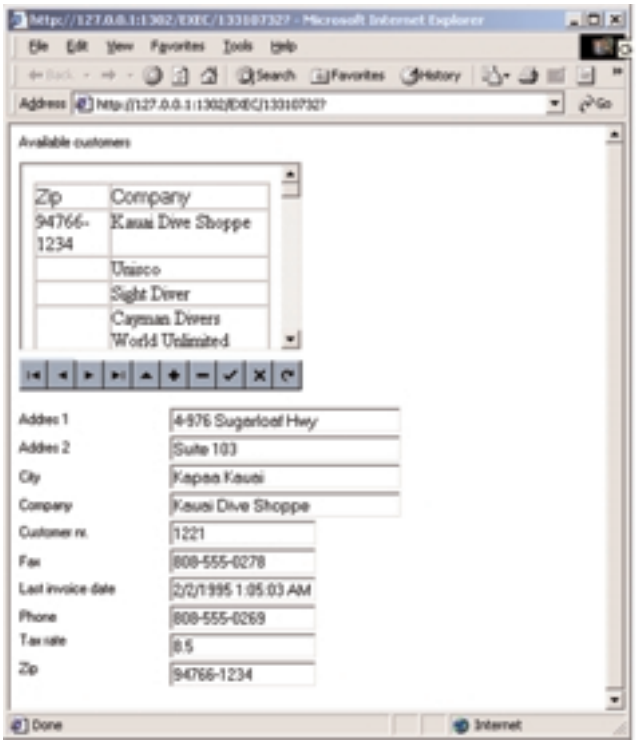
En hoe werkt IntraWeb?

Nadat ND-Intraweb geïnstalleerd is, komt er een tweetal nieuwe tabbladen op uw componentenpalet. De eerste is *Intraweb*. Deze bevat standaard GUI componenten zoals edit's en labels om een applicatie te bouwen. Het tweede tabblad is *Intraweb Data*. Deze bevat componenten om data te integreren in een applicatie. Alle componenten voelen aan zoals we dat gewend zijn in Delphi. Wanneer we een ND-Intraweb applicatie gaan bouwen moeten we beginnen door in Delphi een nieuw project te selecteren. Er wordt tijdens de installatie een nieuwe project wizard toegevoegd om Intraweb applicaties aan te maken.



Wanneer we het tabblad ND-Intraweb 3.0 selecteren krijgen we een tweetal opties: IW Application en ND Intraweb Form. Wanneer voor IW Application wordt gekozen, wordt er een wizard opgestart die vraagt om de directory waar het project moet worden aangemaakt. Er kan hier ook een niet bestaande directory aangegeven worden door de naam van de gewenste directory in te voeren. De wizard maakt de directory en een nieuw projectbestand aan. Opvallend is, dat dit geen normaal .dpr bestand is. Er wordt bijvoorbeeld in plaats van het standaard Application object een IWRun methode gebruikt. Deze zorgt ervoor dat er de applicatie op een bepaalde port gaat luisteren naar binnenkomende http aanvragen hetgeen vergelijkbaar is met een webserver.
De volgende stap is het toevoegen van een form aan ons project. Dit moet wel een form zijn van het type TIWForm die u kunt vinden in de specifieke Intraweb tab in de file new dialoog. In onderstaand voorbeeld is een simpele applicatie gemaakt waarin een aantal edits gekoppeld zijn aan een dataset. Het geheel kostte niet meer dan een paar minuten en het resultaat was een applicatie die te zien was in een browser.

Wanneer de applicatie gestart wordt, verschijnt de Intraweb Project Template Server. Hier vanuit kan de applicatie getest worden. Wanneer we in het menu Run voor de optie execute kiezen wordt de standaard browser gestart en komt direct onze applicatie in beeld. De Project Template Server is ideaal voor het testen van applicaties. Er wordt namelijk direct de standaard browser opgestart met de applicatie die je aan het maken bent in Delphi. We zien dus direct resultaten zonder dat een webserver geconfigureerd hoeft te worden.



In deze modus kan ook gedebugd worden. Wanneer in de code breakpoints gezet worden zal Delphi daar netjes stoppen. Dit is een aanzienlijk verschil met bijvoorbeeld het debuggen van ISAPI of CGI/BIN applicaties die u in Delphi schrijft. We kunnen nu de applicatie installeren, hetzij als een service, hetzij als een standaard applicatie. De wijze van installeren kan met parameters aangestuurd worden. Als de applicatie normaal gestart wordt, verschijnt de template server. Hier is onder andere te zien hoeveel actieve sessies er open staan en op welke port de applicatie luistert.

Als de applicatie met de ***–install*** parameter gestart wordt, wordt hij als een service onder Windows geïnstalleerd. Er verschijnt dan geen status scherm (template server). Intraweb applicaties kunnen op elk willekeurig poortnummer draaien. Indien u geen licentie heeft, zal Intraweb een willekeurig poortnummer pakken. Bij elke executie van de applicatie zal dit poortnummer anders zijn. Na installatie van de applicatie kan deze gestart worden door de juiste URL en poortnummer op te geven.

Denken in HTML

Wanneer u applicaties bouwt met Intraweb moet u er wel degelijk rekening mee houden dat u niet te maken hebt met een standaard Windows applicatie. U merkt het verschil al als u de properties ziet van een IWForm. We kunnen een oneindig aantal forms opnemen in onze applicatie maar we hebben niet de beschikking over dialogen of forms die showmodal gebruiken. Wanneer u dit doet, dan zal dit de hele webserver in een oneindige loop brengen. De reden hiervoor is dat de server de html naar de client verstuurt en geen persistente connectie heeft. Voor mensen die ervaring hebben met het ontwikkelen van web applicaties geen nieuws onder de zon. Voor beginnende webdevelopers is dit iets waar ze aan zullen moeten wennen. Wanneer u visuele componenten wilt plaatsen op het form dan moet u gebruik maken van de door ND-Intraweb aangeleverde componenten of zelf een component bouwen dat is afgeleid van TIWBaseControl. De source code voor de visuele componenten wordt meegeleverd in de evaluatieversie en u kunt hier dus direct mee aan de slag gaan. U kunt trouwens wel normale VCL componenten op de forms plaatsen maar deze worden niet naar HTML geconverteerd.

Integratie met data in applicaties

Databasetoegang kan verkregen worden door gebruik te maken van de standaard Delphi data access componenten zoals tables, queries, etc. U zult er alleen weer rekening mee moeten houden dat het resultaat in een browser terechtkomt. Het gebruik van TTables bij grote tabellen zou dus bijvoorbeeld niet zo slim zijn aangezien alle data in stringvorm dan overgestuurd wordt naar de client. U zult hier waarschijnlijk meer gebruik moeten maken van Client-Server technieken. Een ander aandachtspunt is dat u er rekening mee moet houden dat honderden clients simultaan uw applicatie kunnen gebruiken. Naast alle normale multi-user aspecten met betrekking tot databases heeft dit ook impact op hoe er met de data access componenten omgegaan wordt. Het belangrijk-

Er zijn al bedrijven die .NET gebruiken. Kijk op www.microsoft.com/business/casestudies/b2c/dollarrentacar.asp voor een voorbeeld daarvan.

ste hier is dat u ervoor moet zorgen dat er van sessies gebruik gemaakt wordt aangezien de applicatie in een aparte thread wordt opgestart, zoals dat ook gebeurt bij een ISAPI DLL.

Cosmetische aspecten

We hebben gezien dat we vrij simpel een webapplicatie kunnen bouwen, met de snelheid zoals we dat gewend zijn in Delphi. Maar u wilt ook dat uw applicatie er fraai uitziet of misschien wel integreert in de bedrijfsstijl zoals die gevoerd wordt op een website. Hiervoor kunt u in IntraWeb gebruik maken van templates. Dit zijn HTML bestanden, waarin webontwikkelaars hun stijl kunnen toevoegen zoals zij die op de hele site doorvoeren. Voorbeelden van dit soort templates worden bij het product meegeleverd. In de huidige versie van Intraweb wordt het gebruik van Delphi frames en form inheritance niet ondersteund. Dit is dus iets waar rekening mee gehouden moet worden.

Conclusie

Intraweb is uiterst geschikt als tool om snel en simpel web applicaties te bouwen in de taal waarin we ons als Delphi ontwikkelaars thuis voelen. Het is een tool waarmee je vrij snel vertrouwd raakt en die goed aanvoelt wanneer je er mee aan de slag gaat. Ik vind wel dat de bijgeleverde componenten, in het bijzonder de component editors, wat beter zouden kunnen worden afgewerkt. Dit doet echter niet af aan de kwaliteit die je over het algemeen ziet in dit product. Jammer is wel dat bestaande applicaties niet overgezet kunnen worden naar Intraweb applicaties. Iets wat ook opmerkelijk is bij Nevrona is de geboden ondersteuning voor gebruikers van hun product. Ik nam de vrijheid om door de newsgroups heen te bladeren en vrijwel elk bericht heeft wel een reactie van mensen van Nevrona. Ik heb alle testen gedaan met een standaard Delphi 6 en niet echt getest hoe het pakket samenwerkt met andere 3rd party database access providers zoals DirectAccess voor Oracle of software die gebruikt maakt van DAO/ADO. Gelukkig kunt u een volledig functionele evaluatieversie downloaden voor Delphi 5 en 6 en er mee spelen zoveel als u wilt. (zie hiervoor www.pbe.com/intraweb)

Ronald van Aken is software engineer bij LEAD. Ronald werkt met Delphi sinds het product bestaat. Je kunt Ronald bereiken via rvaken@lead.nl

auteur
Peter van Ooijen

Anatomie van een automation object

(Dr. Delphi snijdt dieper)

Inleiding

Met de komst van .NET heeft Microsoft een beetje de neiging COM als een verouderde welhaast achterhaalde techniek te presenteren. In de praktijk blijft COM, ook in de ogen van MS, een heel belangrijke techniek. Waarom ? In de eerste plaats zal niemand zo gek zijn om al zijn applicaties in één keer helemaal opnieuw te schrijven voor .NET. Als poort naar de "legacy"-buitenwereld biedt .NET een technologie die ze interop noemen. Die komt er op neer dat je COM objecten/interfaces gebruikt om .NET code te laten communiceren met niet .NET code.
Maar ook binnen de .NET tools wordt COM gebruikt. Je vindt het onder zijn drie bekendste namen (COM, ActiveX en OLE-automation) terug in docs en specs. Zo vind je in Jscript.Net de methode CreateActiveXObject. Waarmee je een klassiek automation object aanmaakt en door je .NET script laat aansturen. Niets nieuws onder de zon dus.
Office ontwikkelomgevingen als Word, Excel en Outlook zijn ook nog niet in .NET opgenomen. De manier om deze applicaties met andere stukken software te laten communiceren zijn nog steeds COM interfaces.
Een goede kennis van COM, en met name van automation, is dus nog steeds niet weg, zeker nu dit ook nog eens de brug lijkt naar de (eventuele) toekomst. In de praktijk werkt dat laatste nog niet altijd even lekker, volgens sommigen werkt de COM koppeling in beta2 van .NET zelfs minder goed dan in beta1. Een reden te meer om nog een keer COM-pje onder te gaan.
Het is nu nog "Delphi dot.net not yet" (al schijnt daar wel hard aan gewerkt te worden). Delphi blijft echter mijn favoriete tool om COM objecten mee te bouwen. In dit verhaal wil ik proberen OLEautomation te gaan ontleden door gebruik te maken van alle Object Pascal mogelijkheden die ook in de Delphi VCL zijn gebruikt. We zullen Iunknown en Idispatch in detail voorbij zien komen en al gaande begrippen als early en late binding opnieuw leren kennen.

Simpele server maken

Met de Delphi wizard maak ik eerst een nieuwe ActiveX-library aan en noem die "SDGNanatomy". Daarna maak ik een nieuw automation object genaamd "ExpServer". Dit object krijgt één methode en die noem ik "koekoek". Hiermee zal ik verder gaan experimenteren. Het levert de volgende declaratie op:

```
type
  TExpServer = class(TAutoObject, IExpServer)
  protected
    procedure Koekoek; safecall;
  end;
```

Nog even op een rijtje: in een (COM)interface worden methodes en hun signature (aantal en types van de parameters) beschreven. Wie deze methodes uitvoert en hoe dat gedaan wordt laat een interface zich niet over uit. In de declaratie van een class wordt opgegeven welke interfaces deze class implementeert.

Delphi dot.net not yet ?

Zowel een interface als een class kunnen overerven. Als je een interface van een andere interface af laat stammen dan neem je de belofte over de ouder zijn methodes te ondersteunen. Als je een class van een andere class laat overerven dan neem je niet alleen de belofte, maar ook de daadwerkelijke implementatie van die belofte, over. In het stukje code staat dat onze class tExpServer afstamt van de VCL class tAutoObject en dat hij een implementatie van de interface IexpServer verzorgt. Als je in de VCL sources kijkt dan zie je in de declaratie van tAutoObject dat deze de implementatie van Idispatch verzorgt. En Idispatch is de COM interface die automation verzorgt.

Overriden Iunknown methodes

Alle interfaces stammen af van Iunknown. De class tAutoObject heeft erg interessante voorouders. Hij stamt af van tTypedComObject welke weer afstamt van tCOMobject. In de VCL kan je zien dat deze laatste de implementatie van Iunknown verzorgt.

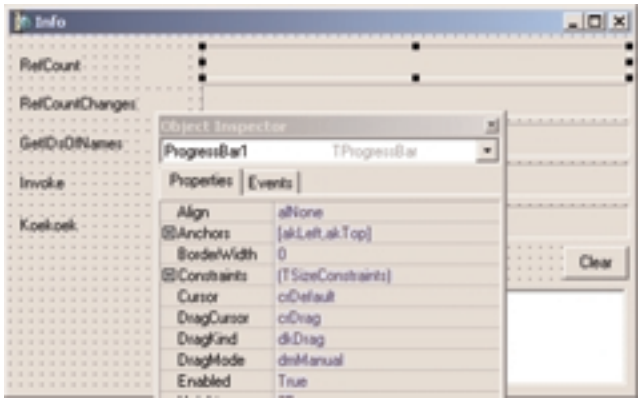
```
TComObject = class(TObject, IUnknown, ISupportErrorInfo)
...
protected
  { IUnknown }
  function IUnknown.QueryInterface = ObjQueryInterface;
  function IUnknown._AddRef = ObjAddRef;
  function IUnknown._Release = ObjRelease;
  ...
public
  function ObjAddRef: Integer; virtual; stdcall;
  function ObjQueryInterface(const IID: TGUID; out Obj):
    HResult; virtual; stdcall;
  function ObjRelease: Integer; virtual; stdcall;
  ...
end;
```

Iunknown heeft maar drie methodes. Met Queryinterface krijg je de één of andere interface naar een implementerend object te pakken. Het duo addref en release zorgen samen voor het lifetime management van het imple-

menterende object. Een COM object heeft een refcount property, die vertelt door hoeveel clients het object gebruikt wordt. Met de addref methode wordt de refcount opgehoogd en met de release methode weer verlaagd. En zodra de refcount op o komt zal het COM object zichzelf vernietigen.
In de VCL tCOMobject class worden de methodes geïmplementeerd door public virtual methodes. Virtual wil zeggen dat je ze kan overriden. Dat laatste verzwijgt de Delphi help file. Het is natuurlijk gevaarlijk dit te doen, maar laten we het toch eens met het addref / release duo gaan proberen. Dat levert de volgende code op.

```
type
  TExpServer = class(TAutoObject, IExpServer)
  private
    fInfoForm : tInfo;
  public
    procedure AfterConstruction; override;
    procedure BeforeDestruction; override;
    function ObjAddRef: Integer; override;
    function ObjRelease: Integer; override;
  ...
```

Om iets zichtbaars in mijn implementatie te doen laat ik tExpServer een form aanmaken van het type tInfo. Op dat formpje kan ik dan info over het aangemaakte object kwijt. Op de form zet ik een aantal progressbars met bijbehorend label. In de afterconstruction (dat is een methode van tObject) wordt het form aangemaakt:



Een form met wat info over het automation object

```
procedure TExpServer.AfterConstruction;
begin
  inherited; fInfoForm:= tInfo.Create(nil);
  fInfoForm.Show;
end;
```

En in de beforedestruction wordt het weer opgeruimd.

```
procedure TExpServer.BeforeDestruction;
begin
  fInfoForm.Free;
end;
```

Zolang mijn autoobject bestaat is er nu een form te zien. In mijn implementatie van ObjAddRef en ObjRelease kan ik dat gebruiken

```
function TExpServer.ObjAddRef: Integer;
begin
  fInfoForm.ShowRefCount(True);
  result := inherited ObjAddRef;
end;
```

Het enige wat deze methode doet is, middels de methode ShowRefCount, wat info op het form zetten en de inherited het echte werk op laten knappen. Ik kan nu volgen wat de refcount is en ook volgen hoe vaak die verandert. Op zich is dit allemaal niet zo schokkend, maar het autoobject ben ik nu wel binnen.
Ik zei net al dat het life-time management van een automation object wordt beheerd door zijn refcount. Wat dus ook betekent dat het object blijft bestaan zolang er iemand of iets een verwijzing naar heeft. Deze code is dus te gebruiken als debug-monitor om bugs van het laatste soort op te sporen.

Overriden Idispatch methodes

Automation wordt verzorgd door de methods van Idispatch. Idispatch stamt af van Iunknown en voegt vier methodes aan de interface toe. GetTypeInfoCount en GetTypeInfo vormen de ingang naar de TypeInfo in de Typelibrary van het object. Zij vormen een hele wereld op zich, aan het eind van dit artikel ga ik nog even op typelibraries in. Deze twee methodes laat ik nu voor wat ze zijn. Het duo GetIDsOfNames en Invoke zijn de motor van het automationobject. Met de eerste vraag je of het object een methode met een bepaalde naam kent en met de tweede voer je de methode daadwerkelijk uit. Het zou leuk zijn ook hier in te kunnen grijpen. En dat kan! In tAutoObject zijn zowel GetIDofNames als Invoke als virtual gedeclareerd. En ook dit wordt in de help file verzwegen. Maar ik ga ze overriden:

```
function GetIDsOfNames(
 const IID: TGUID; Names: Pointer; NameCount,
 LocaleID: Integer; DispIDs: Pointer): HResult;override;
function Invoke(
 DispID: Integer; const IID: TGUID; LocaleID: Integer;
 Flags: Word; var Params; VarResult, ExcepInfo,
 ArgErr: Pointer): HResult; override;
```

In eerste instantie schrijf ik een simpele implementatie. Dat de methode is aangeroepen geef ik weer door aan de infoform en de inherited mag de rest weer doen.

```
function TExpServer.GetIDsOfNames(
 const IID: TGUID; Names: Pointer; NameCount,
 LocaleID: Integer; DispIDs: Pointer): HResult;
begin
  fInfoForm.GetIDsTick;
  result := inherited GetIDsOfNames(
    IID, Names, NameCount, LocaleID, DispIDs);
end;
```

De Invoke implementatie ziet er ook zo uit. En ook de methode koekoek laat ik vertellen dat die is aangeroepen. Nu heb ik een hele simpele automation server gebouwd, deze heeft één methode die eigenlijk helemaal niets doet, maar het object laat op een formpje wel heel duidelijk zien wat er met hem gebeurt. Hiermee kunnen we met verschillende clients verder experimenteren.

Delphi client

Als eerste maak ik een client in Delphi. Deze laat ik de typelib gebruiken en nu ga ik op vier verschillende manieren met objecten van mijn Expserver werken. Om te beginnen declareer ik vier variabelen

```
fEarly   : IexpServer;
fLate    : OleVariant;
fDisp    : IexpServerDisp;
fDispatch: Idispatch;
```

De types IExpServer en Iexpserverdisp staan in de typelibrary, Idispatch en OleVariant zijn windows-brede types. Elk van deze variabelen gaat verwijzen naar een ExpServer-object waarna de methode "Koekoek" aan wordt geroepen.
De eerste manier is via early binding:

```
fEarly := COexpServer.Create;
fEarly.KoeKoek;
```

Onze eigen CoClass Expserver laten we een object aanmaken, wat een interface van het type IexpServer oplevert. Via deze interface wordt koekoek uitgevoerd. Bij het uitvoeren van deze code zullen we in de infoform zien dat de refcount van het object verandert en dat de methode koekoek wordt aangeroepen. GetIdsOfNames en Invoke zijn niet nodig, de Delphi compiler heeft de aanroep van koekoek rechtsreeks aan de vTable van het tExpServer automation object kunnen binden.

De tweede manier is via late binding:

```
fLate := CreateOleObject('SDGNanatomy.ExpServer');
fLate.Koekoek;
```

De compiler kan hier erg weinig mee. Als je typefouten maakt in de parameter van CreateOleObject dan zal je dat pas bij het uitvoeren van de code merken. Begrijpelijk, het is in een string. Maar ook het stukje *fLate.Koekoek* wordt pas runtime gevalideerd. Als je hier *fLate.RenKoekoek* had geklopt dan was het volgens de Delphi compiler ook goed geweest.

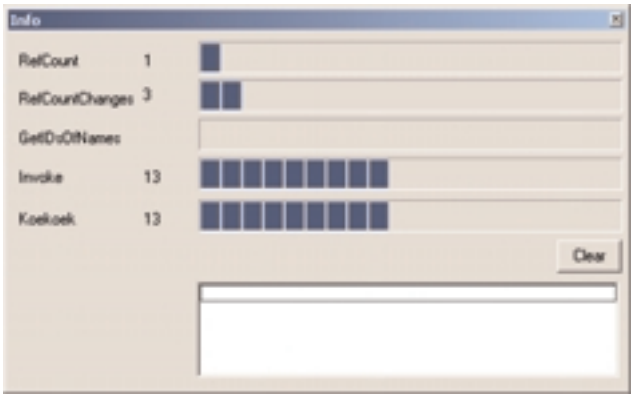
Bij het uitvoeren van de code zien we nu GetIDsOfNames en Invoke in actie komen. Bij elke aanroep gaat eerst GetIDsOfNames af om de juiste info voor Invoke op te halen, daarna gaat invoke af om de methode uit te voeren en daarna gaat koekoek af. Je kan je voorstellen dat dit allemaal wat langer duurt dan early binding.

Een reden te meer om nog een keer COM-pje onder te gaan

In de derde manier gebruiken we een tussenweg. In de .pas versie van de type library zie je naast de IsExpServer ook een IexpServerDisp staan. Achter elke methode staat hier een dispid. Het is een zogeheten dispinterface. Daar moet iets mee te doen zijn. Het levert de volgende code op

```
TempDisp := CreateOleObject('SDGNanatomy.ExpServer');
Tempdisp.QueryInterface(IexpServerDisp, fDisp);
fDisp.Koekoek;
```

TempDisp is een tijdelijke tussenvariabele van het type OleVariant. Net als in de vorige manier laat ik hem een ExpServer object aanmaken. Maar daarna haal ik met QueryInterface de dispinterface op en roep daarop Koekoek aan. De compiler kan nu iets meer doen. Je zal zien dat hij alleen bestaande methodes van IexpServerDisp toelaat. Voer je de code nu uit dan zie je dat GetIDsofNames niet meer uitgevoerd wordt maar Invoke en Koekoek nog wel. Wezenlijk ander gedrag dus.



Een dispinterface in actie

Bij de vierde manier wil ik nog flexibeler met een late-bound object om gaan. Uitgaande van een OleVariant genereerde de Delphi compiler (achter de schermen) aanroepen naar GetIdsOfNames en Invoke gebruik makend van de string "KoeKoek". Maar dat zou ik zelf ook moeten kunnen. In mijn declaratie ben ik iets specifieker, ik declareer een Idispatch variabele en geen OleVariant. Ik kan het object net zo aanmaken als de OleVariant, maar het aanroepen van GetIdsOfNames en Invoke kost iets meer moeite.

```
var NameArray  : array[0..0] of WideString;
    DispIdArray: array[0..0] of integer;
    ParamRecord: DispParams;
begin
  fDispatch := CreateOleObject('SDGNanatomy.ExpServer');
  NameArray[0] := 'Koekoek';
  ParamRecord.cArgs := 0;
  ParamRecord.cNamedArgs := 0;
  if fDispatch.GetIDsOfNames(
    GUID_NULL, @NameArray, 1, LOCALE_USER_DEFAULT,
    @DispIdArray) = S_OK then
    fDispatch.Invoke(
      DispIdArray[0], GUID_NULL, LOCALE_USER_DEFAULT,
      DISPATCH_METHOD, ParamRecord, nil, nil, nil);
```

Beide methodes hebben nogal wat parameters.

GetIDsOfNames

- De eerste is "reserverd for future use" en wil altijd GUID_NULL meekrijgen
- De namen waarvan je de Id's wil weten geef je door in een pointer naar een array van WideStrings. (COM werkt altijd met widestrings)
- De dispIDs krijg je terug in een array van integers. Voor beide arrays moet je zelf zorgen, ik declareer ze hier als locale variabelen. In het eerste element van de namen array stop ik de string "koekoek", waarna GetIDsOfNames kan worden aangeroepen
- Met de 1 geef ik aan dat ik maar één naam opvraag
- de LOCALE_USER_DEFAULT wordt gebruikt om meertaligheid in automation servers te kunnen implementeren. Doe ik niet aan

- Als GetIdsOfNames S_OK als resultaat oplevert dan heeft die een dispid voor koekoek gevonden en kan ik deze gaan gebruiken in Invoke

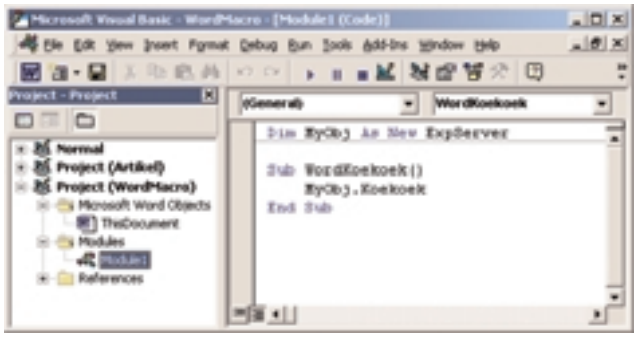
Invoke

- De parameters voor Invoke worden meegegeven in een dispparams record. Gelukkig is dat type al in de VCL gedeclareerd
- het voldoet een 0 als aantal parameters en een 0 als aantal named parameters (named parameters worden gebruikt bij een aanroep als *MyDocument.Save options := none*) op te geven
- DISPATCH_METHOD wil zeggen dat ik hier een methode aanroep en geen property set of get

Het resultaat is precies hetzelfde als bij het gebruik van een OleVariant. Het verschil is nu wel dat ik run time in mijn Client de naam van de aan te roepen methode zou kunnen bepalen. Daar gaan we straks nog leuke dingen mee doen.

Welke clients gebuiken wat

Met de Delphi client hebben we nu kunnen zien wat de verschillende manieren van gebruik voor invloed hebben op de interne werking van het object. De volgende stap is te onderzoeken hoe verschillende bekende gebruikers van automation objecten met mijn ExpServer zullen omgaan.



De server in Word

Als eerste pak ik Word. In VBA voeg ik in tools | references de SDGNanatomy server toe. In de macro module declareer ik een nieuwe instantie van de server

```
Dim MyObj As New ExpServer
```

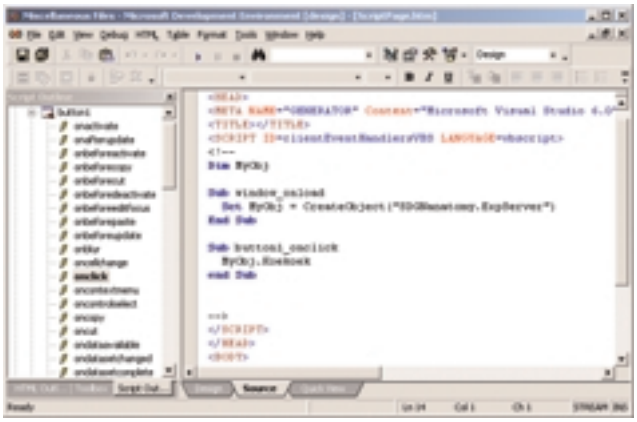
en maak een macro die koekoek zegt:

```
Sub WordKoekoek()
  MyObj.Koekoek
End Sub
```

Als deze macro wordt uitgevoerd dan zien we in het infoform dat noch GetIDsOfNames, noch Invoke afgaat. Je had het ook wat luier kunnen coderen

```
Dim MyLateObject As Object
Sub WordLateKoekKoek()
  If MyLateObject Is Nothing Then
    Set MyLateObject =
CreateObject("SDGNanatomy.expServer")
  End If
  MyLateObject.Koekoek
End Sub
```

Nu zie je bij elke aanroep GetIDsOfNames en Invoke wèl afgaan. VBA is kennelijk goed in staat informatie uit de typelibrary om te zetten in efficiënte code.



De server in een web pagina

Als tweede gebruiker neem ik een stukje VBScript in Internet Explorer. De HTML pagina laat ik bij laden een ExpServer object aanmaken. De pagina heeft één button, als je hem klikt dan zegt het object weer koekoek. En hier zien we dat bij iedere aanroep GetIDsOfNames en Invoke voorbij komen. Dit is typisch script, alles wordt geïnterpreteerd.

Om een gebruiker van dispinterfaces te vinden kom je bij een oudere versie van VB en wie weet wat voor andere tools terecht. Deze werken met dispinterfaces, ze kennen van de automation objecten de dispId's en roepen alleen Invoke aan. In de tijd dat dit dè ontwikkeltools waren werd het gebruik van dispinterfaces late binding genoemd. Run time werd er nog eens bepaald wat de parameters van de methode waren en werd hier de dispparams structure mee gevuld. Het nieuwe VBA kent nog steeds zaken als optionele en named parameters. Maar blijkbaar gebruikt VBA de Invoke methode niet meer om daar wat mee te doen.
Tegenwoordig denk je bij late binding meer aan een client als VBScript, een client die bij iedere aanroep weer (opnieuw) zal vragen wat de ID van een methode is.

Runtime nieuwe properties maken

Gegeven al het voorafgaande is er nu een heel spannende mogelijkheid ontstaan. Ik weet dat een echte latebound client iedere keer aan GetIDsOfNames een ID zal vragen. En iedere keer dat die voorbijkomt passeert deze mijn Delphi code. En waarom zou ik niet naar eigen inzicht in kunnen grijpen en run-time nieuwe properties aan mijn object toe gaan voegen? Dat valt te proberen.

Als eerste geef ik ExpServer een nieuwe methode. Deze noem ik "learn" en hij krijgt twee parameters. De eerste is de naam van de nieuwe property, de tweede is zijn waarde. Intern ga ik deze properties opslaan in een stringlist en zal ze laten zien in een listbox op de infoform.

De stringlist maak ik, net als de form zelf, aan in de AfterConstruction en ruim ik op in de BeforeDestruction. De property sorted van de stringlist gaat op True zodat je er makkelijk in kan zoeken. Mijn properties voeg ik met AddObject aan de stringlist toe, de stringlist zelf bevat de naam, het object is een wrapper om zijn waarde. De implementatie van Learn kan er dus zo uit gaan zien:

```
type
  tLearnedProp = class(tObject)
    PropValue: Variant;
  public
    constructor CreateEx(TheValue: Variant);
end;

constructor tLearnedProp.CreateEx(TheValue: Variant);
begin
  Create;
  PropValue := TheValue;
end;

procedure TExpServer.Learn(PropName, PropValue: OleVariant);
var I : integer;
    NewProp : tLearnedProp;
begin
  {Property was al bekend, krijgt nieuwe waarde}
  if fLearned.Find(PropName, i) then
  begin
    tLearnedProp(fLearned.Objects[i]).PropValue :=
      PropValue;
    fInfoForm.ListBox1.Items[i] :=
      PropName + ' : ' + PropValue;
  end
  else begin
    fLearned.AddObject(PropName,
      tLearnedProp.CreateEx(PropValue));
    fInfoForm.ListBox1.Items.Add(
      PropName + ' : ' + PropValue);
  end;
end;
```

Dat is heel recht toe recht aan. De propertynaam wordt opgezocht in de stringlist. De waarde wordt als object in de stringlist bewaard.
Aan GetIDsOfNames wordt gevraagd of een property bestaat. Ik laat ze nu in deze stringlist zoeken

```
function TExpServer.GetIDsOfNames(
 const IID: TGUID; Names: Pointer; NameCount,
 LocaleID: Integer; DispIDs: Pointer): HResult;
var MyName: WideString;
    MyIndex: integer;
begin
  fInfoForm.GetIDsTick;
  result:= S_FALSE;
  if NameCount = 1 then
  begin
    MyName:= pWideString(Names)^;
    if fLearned.Find(MyName, MyIndex) then
    begin
      pinteger(DispIDs)^:= MyDispIDbase + MyIndex;
      result:= S_OK;
    end;
  end;
  if result <> S_OK then
    result := inherited GetIDsOfNames(
      IID, Names, NameCount, LocaleID, DispIDs);
end;
```

De methode gaat er in eerste instantie van uit dat de property niet bekend is. Eerst wordt gekeken of de client maar van één property de info wil hebben. Zo ja, dan wordt de propertynaam opgehaald en opgezocht in de stringlist. Stel nu dat die gevonden wordt. Dan moet ik een ID teruggeven die mijn implementatie van Invoke ook weer herkent. De gewone dispids in de typelib van ExpServer beginnen met 1 en worden consequent doorgenummerd. Dat zal betekenen dat wat hogere waardes in principe beschikbaar zijn. MyDispIDbase is een con-

stante die ik op 512 zet. Kan ik even vooruit. Bij deze constante tel ik de index in de stringlist op en het resultaat geef ik terug in de parameter. Als resultaat van GetIDsOfNames kan ik nu S_OK teruggeven. Komt de naam niet voor, of wordt er meer dan 1 naam gevraagd, dan zal de inherited implementatie het op moeten lossen.

Als GetIDsOfNames een S_OK heeft opgeleverd dan komt mijn client terug met een aanroep van Invoke. De aanpassing in de implementatie hiervan is simpel:

```
function TExpServer.Invoke(
 DispID: Integer; const IID: TGUID; LocaleID: Integer;
 Flags: Word; var Params; VarResult, ExcepInfo,
 ArgErr: Pointer): HResult;
begin
  fInfoForm.InvokeTick;
  if DispID >= MyDispIDbase then
  begin
    pVariant(VarResult)^ := tLearnedProp(
     fLearned.Objects[DispID - MyDispIDbase]).PropValue;
    result:= S_OK;
  end
  else result := inherited Invoke(
    DispID, IID, LocaleID, Flags,
    Params, VarResult, ExcepInfo, ArgErr);
end;
```

Ik test eerst of de dispid door mijn implementatie van GetIDs is verzonnen door te kijken of die groter of gelijk is dan de constante MyDispIDbase. Zo ja, dan kan ik de dispid (na de constante er af te hebben getrokken) gebruiken als index in de stringlist. In de result parameter wordt de bewaarde waarde teruggeven. Invoke krijgt een pointer naar een variant aangeboden. Bij het behandelen van het gebruik van Invoke zal ik hier nog iets verder op ingaan.

Werkt dit ?

Goed, nou heb ik wel iets vreemd in elkaar geknutseld, maar de vraag is of het ook werkt. In eerste instantie ga ik dit testen met mijn Delphi client, die biedt tenslotte de meeste mogelijkheden. Om het te laten werken moet ik weer latebound gaan werken. Net hebben we al gezien hoe je GetIDsOfNames gebruikte, daar ga ik mee verder. De learn methode van ExpServer heeft twee parameters. Om die mee te geven moet ik iets meer moeite doen dan zonet:

```
procedure TForm1.ActionLearnExecute(Sender: TObject);
var NameArray  : array[0..0] of WideString;
    DispIdArray: array[0..0] of integer;
    ParamRecord: DispParams;
    Params     : array[0..1] of Variant;
begin
  NameArray[0] := 'Learn';
  Params[0] := Edit2.Text;
  Params[1] := Edit1.Text;
  ParamRecord.cArgs := 2;
  ParamRecord.rgvarg := @Params;
  ParamRecord.cNamedArgs := 0;
  if fDispatch.GetIDsOfNames(
    GUID_NULL, @NameArray, 1,
    LOCALE_USER_DEFAULT, @DispIdArray) = S_OK then
    fDispatch.Invoke(
      DispIdArray[0], GUID_NULL, LOCALE_USER_DEFAULT,
      DISPATCH_METHOD, ParamRecord, nil, nil, nil);
end;
```

De aanroep van GetIDsOfNames is precies hetzelfde, nu stop ik de string "Learn" in het eerste array element en ik ►

zal de dispid van de methode learn terugkrijgen. De extra moeite zit hem in het meegeven van de parameters. Die komen in een array van variants, die declareer ik lokaal en vul ik vanuit twee editbox-jes. Waar je hier op moet letten is dat de parameters in omgekeerde volgorde mee moeten worden gegeven. De eerste parameter van learn (in Edit1) bevat de propertynaam, de tweede (in Edit2) bevat de waarde. Echter de tweede parameter stop ik element 0 van de array en de eerste in element 1. Het paramrecord wordt verteld dat er twee parameters zijn en nog steeds 0 named parameters. ParamRecord.rgvarg is een pointer naar de werkelijke parameters. Met @Params wijst die nu naar mijn lokale parameter array. Dit werkt, al klikkend zal je de listbox op het inform zien groeien.

Maar nu wordt het spannend. Kent mijn object nu ook de properties die het heeft geleerd ? De code om dit te testen lijkt weer sprekend op hetgeen we nu al een paar keer hebben gedaan

```
var InvResult: Variant;
begin
  NameArray[0] := Edit1.Text;
  ParamRecord.cArgs := 0;
  ParamRecord.cNamedArgs := 0;
  if fDispatch.GetIDsOfNames(...) = S_OK
  then fDispatch.Invoke(GUID_NULL,  LOCALE_USER_DEFAULT,
      DISPATCH_PROPERTYGET, ParamRecord, @InvResult, nil, nil);
  if not VarIsEmpty(InvResult) then
    Label1.Caption := InvResult;
```

Aan GetIDsOfNames verandert helemaal niets. Invoke geef ik een vlag DISPATCH_PROPERTYGET mee. Niet dat die daar in dit geval nou zo veel mee zal doen, maar het is wel netjes. Het grote verschil is dat de via Invoke aangeroepen property nu een resultaat heeft. Dit resulaat wordt teruggeven in een variant. Deze variant moet je zelf declareren, dat doe ik als InvResult. Invoke gebruikt een pointer naar de variant, daarom geef ik @InvResult mee. Ook dit werkt! De ExpServer leert al klikkend nieuwe properties.

Werkt het ook in script ?

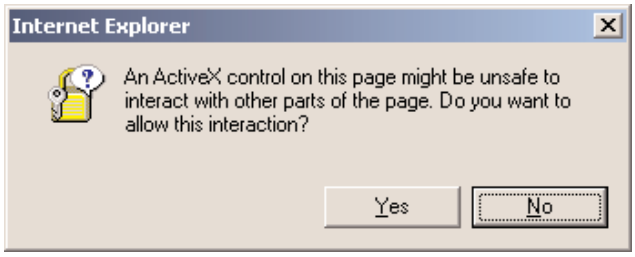
Ik ga terug naar MSE (in Word, onder Tools de keuze Script Editor) om op de eerder gemaakte webpagina het ExpServer object verder te testen. Van MSE krijg ik minstens zo veel jeuk als van VBA, maar het lukt me wel om hiermee snel een pagina met wat knoppen en invoervelden te maken. Het kan niet vaak genoeg gezegd worden, wat zijn we toch gezegend met de IDE van Delphi! Zo, de welhaast verplichte sneer naar MS zit er weer op en ik kan weer aan het werk. Op de pagina maak ik een leerknop met twee invoervelden en een vraagknop met twee velden. Onder de leerknop hang ik dit scriptje:

```
Sub button2_onclick
  MyObj.Learn text1.value, text2.value
End Sub
```

En onder de vraag knop dit scriptje:

```
Sub button3_onclick
  on error resume next
  text4.value = Eval("MyObj." & text3.value)
End Sub
```

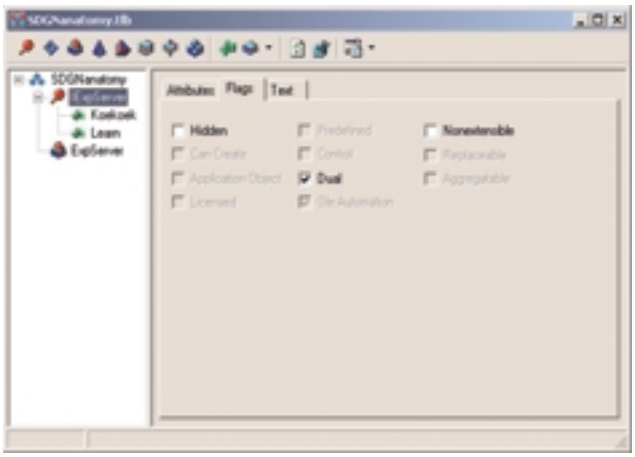
On error resume next is nou niet bepaald de mooiste foutafhandeling die er is, maar het is het enige dat VBScript te bieden heeft. Er is in dit geval geen "next", maar zo voorkom je wel dat allerlei debuggers gaan zeuren als er iets mis gaat in de volgende regel. Daar gebruik ik een hele krachtige script functie. De parameter van de functie is een string die door de Eval functie als script wordt geëvalueerd. En het resultaat van die evaluatie is het resultaat van de *Eval* functie.



Is het veilig ?

Als ik de pagina open zal IE merken dat er een automation object (hij noemt het een ActiveXControl) wordt aangemaakt. Voor de veiligheid, je weet maar nooit wat het object allemaal met je machine uit gaat vreten. Als MyObj mag worden aangemaakt kan ik runtime willekeurig properties aan MyObj toevoegen en weer opvragen. Het blijkt dat de ExpServer ook in een scriptomgeving werkt.

Vlaggen in typelib



De typelibary van IExpServer

Ik wil nog even terugkomen op de typelibrary van de ExpServer. Ik heb me helemaal geconcentreerd op de methodes GetIDsOfNames en Invoke van Idispatch. Ik ben niet ingegaan op de twee typeinfo methodes: GetTypeInfoCount en GetTypeInfo. Deze methodes bieden toegang tot de informatie in de typelibrary van de ExpServer via een interface van het type ITypeInfo. Ook ITypeInfo heeft een GetIDsOfNames en Invoke methode. De interne implementatie van GetIDsOfNames en Invoke in Idispatch doet niet veel meer dan de aanroep doorspelen naar de gelijknamige methodes in de type-

info. Er gebeurt dus nog veel meer met de typelibrary dan je misschien had gedacht.
Een typelibrary heeft een hoop vlaggen die extra info bevatten. Twee van de vlaggen van de IExpServer interface zijn in ons geval interessant.

- De eerste is de dual vlag. Default staat die aan. Als je deze uit zet en vervolgens een nieuwe versie van de SDGNanatomy_tlb.pas genereert zal je merken dat de dispinterfaces zijn verdwenen. Dual wil zoveel zeggen als èn early bound èn late bound. Maar dan wel late bound via een dispinterface en niet op de manier dat VBScript het doet. De Delphi client via dispinterfaces zal niet meer werken, hij zal niets een compileren. De VBScript client echter heeft geen enkele moeite met deze versie van de ExpServer

- Net heb ik met success het aantal en de naam van properties zo maar zitten veranderen. Dat lukte wel maar je kan je af vragen of dat "mag". Nou, in de typelibrary is er de nonextensible vlag, die gaat hier over. Nonextensible wil zeggen dat de properties en methodes run-time niet zullen veranderen. Als ik die vlag aan zou zetten dan zou mijn properties kunstje het dus niet meer mogen doen. De vlag blijkt niet veel meer dan een belofte, VBScript negeert hem en laat me rustig MyObj verder verbouwen

Tot slot

Al knutselend met Object Pascal heb ik een hoop gekke dingen kunnen doen met COM objecten. En al doende heb ik een veel beter idee gekregen hoe ze werken en vooral ook hoe de gebruikers van de objecten werken. Op de website staan de sources bij dit verhaal. Ik nodig iedereen uit om er eens mee te spelen, met name ben ik

Met Delphi kan je heel diep snijden, maar voor je het weet raak je iets vitaals

benieuwd hoe de vele COM clients met de ExpServer om gaan. Een waarschuwing vooraf lijkt me niet overbodig, met Delphi kan je heel diep snijden, maar voor je het weet raak je iets vitaals. Windows NT (4/5) redt je vaak uit de nesten maar speel hier niet mee op een W9x machine. Tenzij je dol op blauw bent.

Peter van Ooijen is een Delphi ontwikkelaar met een grote interesse in COM. Op zijn website www.Gekko-Software.nl kan je wat meer over hem te weten komen.

auteur
Frits Bosschert

Inleiding in PHP

Dit is het eerste deel van een cursus over PHP. PHP is een populaire en snelgroeiende scripttaal voor op het web. PHP is server-side, besturingssysteem onafhankelijk en kan eenvoudig worden gebruikt met opmaaktalen zoals HTML en andere scripttalen zoals JavaScript.

Geschiedenis van PHP

PHP staat voor "PHP: Hypertext Preprocessor" en is bedacht in 1994 door de in Groenland geboren Rasmus Lerdorf. De allereerste versies werden alleen voor zijn eigen homepage gebruikt en waren niet voor andere bedoeld. In het begin van 1995 kwam de eerste versie uit die door anderen gebruikt mocht worden onder de naam "Personal Home Page Tools". De eerste versie ondersteunde alleen de simpelste toepassingen, zoals een gastenboek of een bezoekersteller. In vergelijking met Perl was dit dan ook nog geen bijzondere taal.

In de zomer van 1995 kwam er een nieuwe versie uit van PHP, namelijk PHP/FI Versie 2. Rasmus Lerdorf had de parser, datgene wat de code omzet, volledig herschreven en een ander pakket toegevoegd. Dat andere pakket heette "Form Interpreter" en de afkorting daarvan was FI, zoals de naam van de tweede versie al zegt. FI kan samenwerken met gegevens van formulieren in HTML. Daarnaast voegde hij ook nog de ondersteuning van mSQL toe, waardoor er tegenwoordig gecommuniceerd kan worden met databases. Veel mensen begonnen deze taal te gebruiken en sommige stuurden zelfs eigen verbeteringen in, waardoor de taal ineens gigantisch populair werd.

Rond 1997 kwam er een georganiseerd team dat zich bezig hield met het bijhouden en ontwikkelen van nieuwe PHP versies. Bij de derde versie werd de parser weer volledig herschreven, dit keer niet door Rasmus Lerdorf zelf, maar door Zeev Suraski en Andi Gutmans. Sindsdien werd PHP alleen nog maar populairder en tegenwoordig wordt de taal door meer dan 5 miljoen sites gebruikt. Met de nieuwste versie, PHP 4, lijkt dit aantal alleen nog maar te groeien.

Waarom PHP?

PHP heeft een groot aantal voordelen, dat is ook een van de redenen dat het zo populair geworden is. Hieronder één opsomming van de voordelen:

- PHP is besturingssysteem onafhankelijk. Dit betekent dat PHP op elk besturingssysteem kan draaien, dit in tegenstelling tot bijvoorbeeld ASP. PHP kan bijvoorbeeld op Windows NT draaien én op Unix.
- PHP is Open-Source. Dit betekent dat iedereen mee kan helpen aan het ontwikkelen en verbeteren van PHP.
- PHP is gratis te downloaden van Internet.
- PHP ondersteunt een groot aantal databases, waaronder:
  - dBase
  - mSQL
  - MySQL
  - ODBC
  - Oracle
  - PostgreSQL

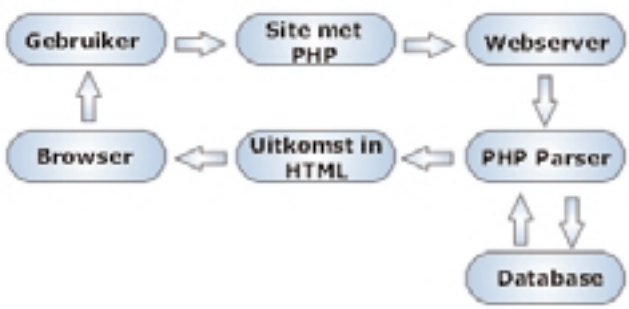
  Dit is nog maar een kleine opsomming, hiernaast worden nog vele andere databases ondersteund door PHP.
- PHP is makkelijker dan programmeertalen zoals JAVA, C of Perl.
- PHP is gebaseerd op de grotere programmeertalen zoals JAVA, C en Perl. Voor de programmeurs die die talen al volledig kennen, is PHP (nog) makkelijk(er) te leren.
- PHP is makkelijk te combineren met opmaaktalen zoals HTML en CSS.
- Er is veel documentatie over PHP te vinden, vooral op Internet.
- PHP heeft evenveel mogelijkheden als andere scripttalen.
- De code hoeft niet gecompileerd te worden, zoals andere talen, bijvoorbeeld JAVA. Bij PHP gebeurt dit op de server en kost het de schrijver en/of gebruiker geen moeite.

PHP kan alleen gebruikt worden voor websites

Dit waren de grootste voordelen. Het enige nadeel van PHP is dat het een scripttaal is en geen programmeertaal. Dat wil zeggen dat PHP alleen kan gebruikt worden voor websites en niet voor aparte programma's.

De werking van PHP

Om PHP scripts te draaien is een webserver, met daarop een PHP parser geïnstalleerd, noodzakelijk. Een PHP Parser is een programma dat de PHP code verwerkt en omzet naar HTML, zodat de browser het kan omzetten naar de pagina die men te zien krijgt. Hier volgt een diagram met de werking van PHP en daaronder een toelichting:



Figuur 1: Werking van PHP

De browser van de gebruiker stuurt een verzoek naar de webserver om een PHP file te bezoeken, de webserver reageert hierop en schakelt de PHP parser in. De PHP parser wisselt eventueel gegevens uit met één of meerdere databases, als dat is aangegeven in het script. Daarna worden de gegevens verwerkt en de uitkomst in HTML weer teruggestuurd naar de browser van de gebruiker.
Alles rond PHP gebeurt op de webserver. De gebruiker merkt er in principe niets van, dit in tegenstelling tot bijvoorbeeld Flash. Daarbij heeft de gebruiker een speciale plugin nodig om de Flashsites te bekijken. Bij PHP hoeft de browser van de gebruiker niet aan bijzondere eisen te voldoen.

Dit in tegenstelling tot de webserver. De webserver moet dus een PHP Parser geïnstalleerd hebben om de PHP code te verwerken en te kunnen vertalen. De meeste professionele hostingbedrijven ondersteunen tegenwoordig PHP. Als dat niet het geval is, zijn er alternatieven. Een alternatief zou kunnen zijn om je site te hosten op een gratis webserver met PHP ondersteuning. Een aantal sites waar dit kan zijn:

- http://www.f2s.com/
- http://www.portland.co.uk/

Daarnaast is er ook nog een ander alternatief. Je kunt je eigen webserver opzetten. Dit is niet zo moeilijk als je wellicht zou denken. Windows-gebruikers kunnen van Internet een programma downloaden dat automatisch een Apache webserver installeert met daarop PHP en MySQL geïnstalleerd. Daarnaast is er ook PHPMyAdmin bij geïnstalleerd, waarmee men makkelijk de gegevens uit de database kan bewerken. Dit pakket is te downloaden op: http://sourceforge.net/projects/phptriad/
Wil je zelf handmatig de Apache Webserver met PHP en MySQL installeren? Op de volgende URL is hierover een goed artikel te vinden:
http://members.home.nl/rvanmil/faq/Installatie%20Webserver.html

De programmeur heeft geen aparte benodigdheden nodig om PHP te kunnen schrijven. Een tekstverwerker, zoals Kladblok, is al genoeg om een PHP script te maken. Er zijn natuurlijk legio programma's die het schrijven van PHP scripts een stuk makkelijker maken, door bijvoorbeeld bepaalde codes kleuren te geven, waardoor ze ►

beter opvallen. Een voorbeeld van zo'n programma is: PHPed, te downloaden op:
http://soysal.com/download2.html

Er zijn op Internet ook engines te downloaden. Dat zijn complete sites die al helemaal voor zijn geprogrammeerd. Daar hoeft men alleen de inhoud en lay-out te veranderen, maar de code is al geschreven voor hen. Een goed voorbeeld daarvan is PHP-Nuke. PHP-Nuke is te downloaden op: http://www.phpnuke.org/. Natuurlijk is dat wel handig, maar het is veel leuker om je eigen code te schrijven.

Alles rond PHP gebeurt op de webserver

Aan de slag met PHP

Als we een webserver hebben gevonden die PHP ondersteunt, kunnen we aan de slag met het schrijven van een PHP script. Een eenvoudig voorbeeld van een PHP script zie je hieronder:

```
<?
Echo ("Dit is een voorbeeld van een PHP script");
?>
```

Als dit wordt opgeslagen als een .php file en wordt uitgevoerd, moet dit op het scherm komen te staan:
Dit is een voorbeeld van een PHP Script.

Het script kan ook opgeslagen worden als bijvoorbeeld .php3 of .phtml. Hieronder wordt met commentaar uitgelegd, hoe het script werkt.

```
<?
/* Dit geeft aan dat de tekst hieronder PHP is. */
Echo ("Dit is een voorbeeld van een PHP script");
/* Echo is een functie om iets te laten zien. Dit kan ook
zonder de haakjes. Zoals bijvoorbeeld hieronder:
Echo "Dit is een ander voorbeeld om iets te laten zien";
Dit komt dus niet op het scherm, omdat het allemaal in de
commentaar regels wordt geschreven.
*/
// De tekens hieronder geven het einde van de PHP codes aan.
?>
```

De /* staan voor het aangeven van commentaar. Met */ geef je weer aan dat de commentaar regels is afgelopen. Dit is dus voor meerdere regels, zoals te zien is in het voorbeeld script.
// zijn de tekens voor één regel commentaar. Dat is de reden dat er geen tekens aan het einde van de regel hoeven te staan, om het einde aan te geven.
In plaats van de functie echo, kan de functie print ook gebruikt worden. Zie het voorbeeldscript hieronder:

```
<?
Print "Dit is een andere manier om iets op het scherm af te laten drukken";
?>
```

Dit werkt op dezelfde manier als de functie echo.

Nu gaan we een HTML file koppelen aan een PHP script. Stel we hebben een formulier met daarin een tekstvak en een submit knop. Als men op de Submitknop (Verzendknop) drukt, moet het script test.php aanroepen. Maak een HTML file met de volgende code aan:

```
<form action="test.php" method="post">
Wat is uw achternaam: <input type="text" name="Achternaam"><BR>
<input type="Submit" name="Submit">
</form>
```

Binnen de "form"-tag geeft de "action"-tag aan welk PHP programma aangroepen moet worden na het drukken op de submit-button. Verder is het belangrijk dat het inputveld behalve een type ook een "name"-tag heeft.
We willen nu in het PHP Script de tekst die werd ingevuld in het tekstvak met de naam "Achternaam" op het scherm laten afdrukken. Dit kan met behulp van het volgende script:

```
<?
Echo ("Dit werd ingevuld: $Achternaam");
?>
```

De tekst die dus werd ingevuld in het tekstvak, wordt dus opgeslagen in de variabele "$Achternaam". Een formulier gemaakt in HTML koppelen met een PHP script is dus helemaal niet zo moeilijk.
Hopelijk kan iedereen na deze introductie aan de slag met PHP. Volgende keer meer over zaken als "If, Else statements" en formvalidatie.

Frits Bosschert is een van de twee oprichters van www.dutchdevelopers.nl, dé Nederlandse site voor programmeurs, scripters, designers en andere computergeïnteresseerden. Bereikbaar voor vragen en commentaar op: frits@dutchdevelopers.nl

FoxPro Tip

SQL-SELECT in VFP 7

In Visual Foxpro 7 is het SQL SELECT statement uitgebreid met een NOFILTER clause.
Deze nieuwe NOFILTER clause creëert een cursor die in volgende queries m.b.v. SQL SELECT gebruikt kan worden. In eerdere versies van (Visual) FoxPro, moest men een extra constante of expressie opnemen om dit met zekerheid te bereiken, b.v. "SELECT *, .T. AS lDummi FROM customers INTO CURSOR myquery". Als men dit niet deed, was het mogelijk dat (Visual) FoxPro intern een extra index aanlegde op een bestaande tabel/cursor i.p.v. echt een nieuwe cursor aan te maken. Met de NOFILTER clause is dit niet meer nodig. Deze zorgt ervoor dat er altijd een tijdelijke tabel op schijf gecreëerd wordt, waarmee verder gewerkt kan worden. Merk op, dat dit invloed kan hebben op de performance.

auteur
ANTHONY SMITH

In memory databases – Indexing

Preface

Having already made an "in memory database" that acts as a lookup list, collection class, array handler etc, adding indexes add much more flexibility to the class, and allows us to start using the class in many more places where it was previously limited.

Enhancements

I have also added the ability to store BLOB data and Objects in a field as well.
Various other small enhancements have also been made; refer to the attached .AEF for sources.

Balanced B-tree indexes

Balanced B-tree indexes have been used in this case, all data is consistently and at all times balanced. This does however have performance degradation over other indexing methods, but I feel that this is not a problem as it is still much faster than using disk alternatives. However seeking of data is extremely fast as the indexes are balanced. Creating them in theory is pretty simple, let's look at an example set of data to index.
Values: 4, 9, 3, 2, 5, 8

1. Firstly allocate enough memory to store all the index data.
2. Then start with the first number "4" and write it into the first byte of memory.
3. Perform a seek on the existing data, and write the "9" after the "4" which results in "49"
4. Perform a seek and move the "49" on 1 place and insert the "3" at the beginning of the numbers resulting in "349"
5. Keep performing the same until the resultant data is "234589"

I however feel that this is not a problem as it is still much faster than using disk alternatives

As I said, pretty simple, the only problem is that when you use large amounts of records (10's of thousands or more) you could end up moving a LOT of memory around, which is not always very efficient. The advantage is that if this is to be used a lot for lookups then the performance loss in adding items is really worth it, as the indexes are balanced and really fast with seeks.
Once again I have used numeric field position referencing and not strings. To make your code more readable is easy, just use a DEFINE for the field names eg:

```
DEFINE CUST_NAME := 1
DEFINE CUST_CODE := 2
```

This way you get the best of both worlds.
I have not allowed any "Code-block" or loose typing to create indexes. Rather I have created a field that is the indexing expression.
For example: if you want an index on "Surname + Firstname" then create a field for the "RAMDBF" that contains the sum of both the "Surname" and the "Firstname" and call it something like "FULLNAME" eg:

```
ACCESS Surname as STRING STRICT CLASS MyClass
RETURN Left( SELF:FieldgetString(1), 1, 20 )

ACCESS Firstname as STRING STRICT CLASS MyClass
RETURN Substr( SELF:FieldgetString(1), 21, 20 )
```

Performing seeks would then take the following format.
oDbf:Seek(DEF_FULLNAME, "Smith", 0)
where DEF_FULLNAME is a DEFINE for the field number. The last parameter in the seek method is the amount of bytes to match in your seek. If you leave it "0" then a full match will be performed to the length of the field.

Every time you add a record the value is firstly written away to the table, thereafter a seek is performed on the index and index data is moved around if necessary, before the new data is written to the index in the correct ordinal place.

```
CLASS IndexedRamDbf INHERIT RAMDBF
  PROTECT _ptrIndexes         AS DWORD PTR
  PROTECT _lHasIndexes        AS LOGIC
  PROTECT _ptrIndexPosition   AS BYTE PTR

~"ONLYEARLY+"
  DECLARE METHOD UpdateIndex
  DECLARE METHOD Seek
  DECLARE METHOD IndexedFieldsWritten
~"ONLYEARLY-"
```

I have used a block of allocated memory with the following size: (DEF_Max_Fields * 4) to store references to each index. NB: remember that this is only a reference the position in memory where the index is stored.
For example: if you had an index on the 1st field and the 3rd field then byte position of SELF:_ ptrIndexes would

start at 0-4 and 9-12 and would contain the references to the actual block of memory containing the index.
As an index is created from one of the fields, the number of indexes you can have is limited to the amount of fields you use, i.e.: basically unlimited.

Seeking for each value in a loop (10000 times) takes an additional 0.01 seconds. Pretty fast!

In addition to having the ability to index a field, you can also define an index as being DEF_FieldConsecutive, which boosts performance when adding indexes dramatically. Adding 10000 records with no indexes will take 0.1 seconds ("RAMDBF"). Adding 10000 records with an index on a field that has been declared with DEF_FieldConsecutive takes only 0.14 seconds ("IndexedRamDbf"). Seeking for each value in a loop (10000 times) takes an additional 0.01 seconds. Pretty fast!
However if you are creating a string index containing +- 30 characters that do not follow consecutively as you add it, then adding this data will take +- 0.85 seconds; moving all the memory around and the additional seeks make it a lot slower.

The setup method of this class calls it parents Setup method, allocates a block of memory for actual index pointers and sets each field structure to contain whether the field is indexed and whether the field has an incremental index or not. The Array that is passed to the Setup method of this class looks as follows:

```
aStruct := { ;
      {#CUST_CODE  , DWORD    ,    , DEF_Indexed , ;
                                  DEF_Consecutive } , ;
      {#CUST_NAME  , STRING   , 30 , DEF_Indexed } , ;
      {#CUST_NAME  , STRING   , 30 } , ;
      {#CUST_BAL   , REAL8         } , ;
      {#CUST_Addr1 , STRING   , 30 } , ;
      {#CUST_Addr2 , STRING   , 30 } , ;
      {#CUST_Addr3 , STRING   , 30 } , ;
      {#CUST_Addr4 , STRING   ,  4 } , ;
      {#Active     , LOGIC    } , ;
      {#Joined     , DATE     } }

oDbf := IndexedRamdbf{}
oDbf:Setup( aStruct )
```

Please note the Additional 2 fields that have been passed to the first 2 fields. They denote that the fields are indexed and that the first field has an index that is consecutive.

```
METHOD Append() AS LOGIC STRICT CLASS IndexedRamDbf

LOCAL lAddedBuffer                  AS LOGIC
LOCAL ptrIndexBuffer        AS BYTE PTR
LOCAL ptrNewIndexBuffer     AS BYTE PTR
LOCAL ptrOffset             AS DWORD PTR
LOCAL strucField            AS Fields
LOCAL dwFieldPos            AS DWORD
LOCAL strPointer            AS STRING
LOCAL dwTemp                        AS DWORD
LOCAL pszPointer            AS PSZ
~"ONLYEARLY+"
lAddedBuffer := SUPER:Append()

IF lAddedBuffer .and. SELF:_lHasIndexes

  FOR dwFieldPos := 0 UPTO SELF:_FieldCount - 1
    strucField := SELF:_aStruct[ dwFieldPos + 1 ]

    IF strucField.Indexed == TRUE
      ptrOffset  := SELF:_ptrIndexes + dwFieldPos
      strPointer := Mem2String( ptrOffset , 4 )
      ptrIndexBuffer := PTR( _CAST, Bin2DW( strPointer ) )
      dwTemp := DEF_Buffer_Size * ( strucField.Length + 8 )
      ptrNewIndexBuffer := MemAlloc( SELF:_dwUsedBuffers * ;
                                     dwTemp )
      MemCopy( ptrNewIndexBuffer, ptrIndexBuffer , ;
               ( SELF:_dwUsedBuffers -1 ) * dwTemp )
      MemFree( ptrIndexBuffer )
      dwTemp := DWORD( _CAST, ptrNewIndexBuffer )
      pszPointer := PSZ( DW2Bin( dwTemp ) )
      MemCopy( SELF:_ptrIndexes + dwFieldPos, pszPointer,4)
    ENDIF

  NEXT

  SELF:IndexedFieldsWritten( FALSE )
ENDIF

RETURN lAddedBuffer
~"ONLYEARLY-"
```

In the Append method, we first perform a normal append to the table calling the SUPER:Append() method. If the return result of that method is TRUE (a new buffer was allocated) then a new buffer for each index must also be added. Go into a loop scanning each field to see whether it has an index on it or not. If there is one, then Allocate a buffer to the size of the existing buffer size + 1000 * (size

The number of indexes you can have is limited to the amount of fields you use

of index data to be written + 4 bytes for the record number and 4 bytes for the starting record position of the data). Next, copy all the data from the existing buffer into the new buffer. Delete the old memory buffer that the index was contained in and link the new index pointer to the list of indexes that is used as a reference to index buffers.

```
METHOD FieldPutDWord( dwFieldPos AS DWORD, ;
                      dwValue AS DWORD) AS VOID STRICT ;
       CLASS IndexedRamDbf
LOCAL strucField  AS Fields

~"ONLYEARLY+"
strucField := SELF:_aStruct[ dwFieldPos ]
MemCopyString( SELF:_ptrRecordStartAt + ;
  strucField.Offset ,DW2Bin( dwValue ),strucField.Length)

IF strucField.Indexed == TRUE
  SELF:UpdateIndex( strucField, DW2Bin( dwValue ) )
ENDIF

RETURN
~"ONLYEARLY-"
```

Next add the new value to the table via a Fieldput() method. See the source code for a complete listing of the various fieldput() methods. If this field we are updating is an indexed field, then call the UpdateIndex() method that updates the index.

```
METHOD UpdateIndex( strucField AS Fields, ;
                    strData AS STRING ) AS VOID STRICT ;
       CLASS IndexedRamDbf

LOCAL ptrIndex        AS BYTE PTR
LOCAL ptrTemp         AS BYTE PTR
LOCAL dwOffset        AS DWORD
LOCAL strPointer      AS STRING
LOCAL ptrOffset       AS DWORD PTR
LOCAL dwCurrPosition  AS DWORD
LOCAL dwMoveBytes     AS DWORD
LOCAL ptrSource       AS BYTE PTR
LOCAL ptrDest         AS BYTE PTR
LOCAL dwEndingPos     AS DWORD
LOCAL dwLength        AS DWORD

~"ONLYEARLY+"
  ptrOffset  = SELF:_ptrIndexes + strucField.Position -1
  strPointer := Mem2String( ptrOffset , 4 )
  dwOffset := ( SELF:_dwLastRec - 1 ) * ;
                    ( strucField.Length + 8 )
  ptrIndex := PTR( _CAST, Bin2DW( strPointer ) )

  IF strucField.IndexWritten == TRUE
  ENDIF

  IF strucField.Consecutive != TRUE
    SELF:Seek( strucField.Position, strData , 0 )
    ptrIndex := SELF:_ptrIndexPosition
    dwLength := ( SELF:_dwRecNo - 1 )*(strucField.Length +8)
    dwEndingPos := Bin2DW( strPointer ) + dwLength
    dwCurrPosition := DWORD( _CAST, ptrIndex )

    IF dwEndingPos > dwCurrPosition
      ptrSource := ptrIndex
      ptrDest := ptrSource + strucField.Length + 8
      dwMoveBytes := dwEndingPos - dwCurrPosition
      MemMove( ptrDest, ptrSource, dwMoveBytes )
    ENDIF

    dwOffset := 0
  ENDIF


  ptrTemp := ptrIndex + dwOffset
  strData := PadR( strData, strucField.Length )
  MemCopyString( ptrTemp ,strData , strucField.Length )

  ptrTemp := ptrIndex + dwOffset + strucField.Length
  MemCopy( ptrTemp, PTR( _CAST, DW2Bin( SELF:_dwRecNo)),4)
  ptrTemp := ptrIndex + dwOffset + strucField.Length + 4
  MemCopy( ptrTemp, PTR( _CAST, DW2Bin( DWORD( _CAST, ;
          SELF:_ptrRecordStartAt ) ) ), 4 )
RETURN
~"ONLYEARLY-"
```

Firstly retrieve the pointer to the actual index in memory via the list of indexes (an allocated block of memory that contains a list of pointers to where the actual index data resides in ram).

Take arrays to the Max, Ditch them!

Next, if this value has already been written to the index and you want to update it then you must first remove it from the index, before adding it again. (I have not had time to complete this section yet, but by the time the article is published you can download the updated version) Check to see if the index is NOT a consecutive one. If it is, then the following section is not necessary as it only results in overhead.
Perform a seek to determine the position at which to insert the new index data.

If necessary, move the data that lies between the insertion point and the end of the index onwards by the length of the data being written + 4 bytes for the Record number and 4 bytes for the data pointer position.
Now its simple: copy the actual data, record number and data pointer to the new position.

```
METHOD Seek( dwFieldPos AS DWORD, uValue AS USUAL , dwLength AS;
DWORD ) AS LOGIC STRICT CLASS IndexedRamDBF
LOCAL lFound AS LOGIC
LOCAL dwMiddleElement AS DWORD
LOCAL dwOldMiddleElement AS DWORD
LOCAL strucField AS Fields
LOCAL ptrOffset AS DWORD PTR
LOCAL     strPointer AS STRING
LOCAL ptrIndex AS BYTE PTR
LOCAL dwSeekPosition AS DWORD
LOCAL ptrTemp AS BYTE PTR
LOCAL siCompare AS SHORTINT
LOCAL pszSearchFor AS PSZ
LOCAL dwMoveElements AS DWORD

~"ONLYEARLY+"
strucField := SELF:_aStruct[ dwFieldPos ]
IF dwLength == 0
  dwLength := strucField.Length
ENDIF

IF strucField.Indexed == TRUE
  ptrOffset  := SELF:_ptrIndexes + strucField.Position -1
  strPointer := Mem2String( ptrOffset , 4 )
  ptrIndex := PTR( _CAST, Bin2DW( strPointer ) )

  IF IsString( uValue )
    pszSearchFor := String2Psz( uValue )
  ELSE
    DO CASE
    CASE strucField.Type == DATE
      pszSearchFor := String2Psz( Date2Bin( uValue ) )
    CASE strucField.Type == LOGIC
      pszSearchFor := String2Psz( Logic2Bin( uValue ) )
    CASE strucField.Type == STRING
      pszSearchFor := String2Psz( PadR( uValue, dwLength ) )
    CASE strucField.Type == PSZ
      pszSearchFor := String2Psz( PadR( AsString( uValue ), ;
                                  dwLength ) )
    CASE strucField.Type == DWORD
      pszSearchFor := String2Psz( DW2Bin( uValue ) )
    CASE strucField.Type == WORD
      pszSearchFor := String2Psz( W2Bin( uValue ) )
    CASE strucField.Type == REAL8
      pszSearchFor := String2Psz( Real82Bin( uValue ) )
    CASE strucField.Type == REAL4
      pszSearchFor := String2Psz( Real42Bin( uValue ) )
    CASE strucField.Type == LONGINT
      pszSearchFor := String2Psz( L2Bin( uValue ) )
    CASE strucField.Type == SHORTINT
pszSearchFor := String2Psz( I2Bin( uValue ) )
    CASE strucField.Type == BYTE
      pszSearchFor := String2Psz( I2Bin( uValue ) )
    CASE strucField.Type == FLOAT
      pszSearchFor := String2Psz( F2Bin( uValue ) )
    ENDCASE
  ENDIF

  dwMiddleElement := DWORD( SELF:_dwLastRec / 2 )
  dwOldMiddleElement := 0
  DO WHILE TRUE
    dwSeekPosition := dwMiddleElement * ( dwLength + 8 )
    ptrTemp := ptrIndex + dwSeekPosition

    siCompare := MemComp( pszSearchFor, ptrTemp , dwLength )

    IF siCompare == 0
      lFound := TRUE
      SELF:_ptrIndexPosition := ptrTemp
      EXIT

    ELSEIF siCompare == -1
      IF dwMiddleElement == 0
        SELF:_ptrIndexPosition := ptrTemp
        EXIT
      ELSE
```

```
          IF dwMiddleElement > dwOldMiddleElement
            dwMoveElements := dwMiddleElement - dwOldMiddleElement
          ELSE
            dwMoveElements := dwOldMiddleElement - dwMiddleElement
          ENDIF

          dwMoveElements := DWORD( dwMoveElements / 2 )
          IF dwMoveElements == 0
            IF dwOldMiddleElement - dwMiddleElement >= 1
              dwMoveElements := 1
            ELSE
              SELF:_ptrIndexPosition := ptrTemp
              EXIT
            ENDIF
          ENDIF

          dwOldMiddleElement := dwMiddleElement
          dwMiddleElement := dwMiddleElement - dwMoveElements
        ENDIF

      ELSEIF siCompare == 1

        IF dwMiddleElement == SELF:_dwLastRec - 1
          SELF:_ptrIndexPosition := ptrTemp
          EXIT
        ELSE
          IF dwMiddleElement > dwOldMiddleElement
            dwMoveElements := dwMiddleElement - dwOldMiddleElement
          ELSE
            dwMoveElements := dwOldMiddleElement - dwMiddleElement
          ENDIF

          dwMoveElements := DWORD( dwMoveElements / 2 )
          IF dwMoveElements == 0

            IF dwOldMiddleElement - dwMiddleElement >= 1
              dwMoveElements := 1
            ELSE
              SELF:_ptrIndexPosition := ptrTemp
              EXIT
            ENDIF

          ENDIF

          IF dwMiddleElement + dwMoveElements == dwOldMiddleElement
            SELF:_ptrIndexPosition := ptrTemp + ( dwLength + 8 )
            EXIT
          ELSE
            dwOldMiddleElement := dwMiddleElement
            dwMiddleElement := dwMiddleElement + dwMoveElements
          ENDIF
        ENDIF
      ENDIF
    ENDDO
  ENDIF

  RETURN lFound
  ~"ONLYEARLY-"
```

The seek criteria is converted into a PSZ for the comparison. The seek method starts by determining the middle point of the index (Lastrec/2) and does a comparison between the index value and the seek criteria. If the result is "0" a match has been found. If "-1" the value is smaller than the index value and a value of "1" is greater than the index value. This process continues until a match has been found or BOF, EOF or no more matches are possible. In addition to returning the seek result the position at which the next Seek criteria should be inserted is stored in SELF:_ptrIndexPosition. This is not needed by a conventional seek, but when the class performs a seek to determine the next insertion point of data it is needed.

METHOD ZapBuffers() has also been modified to delete all index data as well.

NB: remember that if you are using very little memory like 128MB then Windows is already swapping to disk and has probably already used up most of the available memory. Adding a few hundred or 1-2 thousand records in a limited memory scenario will work perfectly, but as soon as you add 10's of thousands of records then Windows cannot allocate enough VM for the additional data in time and your program will crash.

This means that there is always enough available RAM for the program to allocate extra when it needs to

I have happily however added more than 1GB of data to a table over and over with a computer that only has 512MB ram with absolutely no problems at all. The reason for this is that Windows starts swapping out memory when it has used up 50% of available RAM. This means that there is always enough available RAM for the program to allocate extra when it needs to. Windows merely swaps out to disk and the program continues to operate at the same speed, reliably.

Conclusion

This class now has the capability of performing all the conventional tasks you would normally use arrays for, both dimensioned and ragged. In addition you have a lot of extra functionality like blazing speed, indexes, a familiar interface, the ability to store objects and BLOB data without having to worry about pointers changing when the garbage collector kicks in. I see this class's primary role as a collection class, used for lookups of frequently used static data. Storing temporary data or data that is constantly being changed before being posted to a database, logging purposes etc.

I intend to optimize this class a little, add a few more features and then add a client server model with multi-threading on the server side, perhaps even an RDD.
"Take arrays to the Max, Ditch them!"

Anthony Smith was born and raised in South Africa, the country in which he learned Visual Objects in different settings amongst that having it's own software-shop. In June 2001 he and his wife Tania moved to Holland where he is now working for Innovact in Amstelveen. There he learned about SDGN for which he hopes to write a lot more articles in the future.

auteur
Marco Hemmes

Objectoriëntatie: hoe passeer je de ouder

Of: hoe kun je in een overschreven methode de methode van de ouder overslaan, maar die van de grootouder wel uitvoeren.

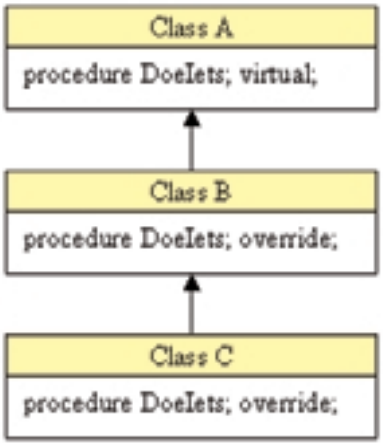
Dit artikel beschrijft een methode, die bij het gebruik van objectorientatie erg handig kan zijn, maar eigenlijk niet gebruikt mag worden. Als de hier gepresenteerde oplossing nodig is, betekent het altijd dat er een fout in het objectoriëntatie ontwerp zit.

Waarom presenteer ik hier dan een oplossing, die niet gebruikt dient te worden? Omdat het bijvoorbeeld niet (meer) mogelijk of gewenst is om het ontwerp te wijzigen. Daarnaast is het gewoon erg interessant en leerzaam; na het lezen van dit artikel weet je wat een virtual-method-table is en hoe het overschrijven van een virtual method intern wordt geregeld.

In dit artikel vind je eerst een situatieschets, daarna een uitleg van de virtual-method-table en daarna twee oplossingen om je doel te bereiken.

Situatieschets

In de figuur rechts zie je drie classes met pijlen getekend. De pijlen betekenen dat Class B erft van Class A en dat Class C erft van Class B. Class A is dus de ouder van Class B en de grootouder van Class C. Class C is het kind van Class B en het kleinkind van Class A.



Class A heeft een procedure DoeIets gedefinieerd die de eigenschap 'virtual' heeft, dus een class die erft van Class A mag de procedure overschrijven. Class B heeft de procedure DoeIets overschreven, die hetzelfde doet als Class A maar dan met nog een paar extra handelingen.

Deze procedure ziet er abstract gezien als volgt uit:

```
procedure TClassB.DoeIets;
begin
  // Voer de handelingen van de ouder uit.
  inherited DoeIets;
  // Voer nog wat extra handelingen uit.
  DoeNogWatExtraHandelingen;
end;
```

Hierboven is te zien dat met het woordje 'inherited' een procedure van de ouder kan worden uitgevoerd, ook al is die procedure in het kind overschreven. Op deze manier kan de werking van de procedure van een ouder dus worden uitgebreid met extra functionaliteit.

Class C heeft op zijn beurt weer de procedure DoeIets van Class B overschreven, maar wil niet dat de extra handelingen worden uitgevoerd die in de procedure DoeIets van Class B staan, maar wil wel dat de handelingen worden uitgevoerd die in de grootouder, Class A, staan. Hier moet nogmaals een opmerking gemaakt worden dat deze wens een fout in het ontwerp van de objectoriëntatie is. Er is namelijk een regel die zegt dat een class die erft van een andere class, alleen functionaliteit mag toevoegen, nooit mag weghalen. Door de functionaliteit van Class B niet te willen uitvoeren, haal je dus functionaliteit weg uit Class B.

Het zou mooi zijn als je de procedure DoeIets van Class C als volgt kon schrijven:

```
procedure TClassC.DoeIets;
begin
  // Voer de handelingen van de grootouder uit.
  inherited inherited DoeIets;  // Compiler fout
end;
```

'Helaas' geeft de Delphi compiler een foutmelding op deze regel, wat natuurlijk terecht is, omdat dit tegen de objectoriëntatie theorie in gaat. Misschien overweeg je om de procedure dan maar als volgt te schrijven:

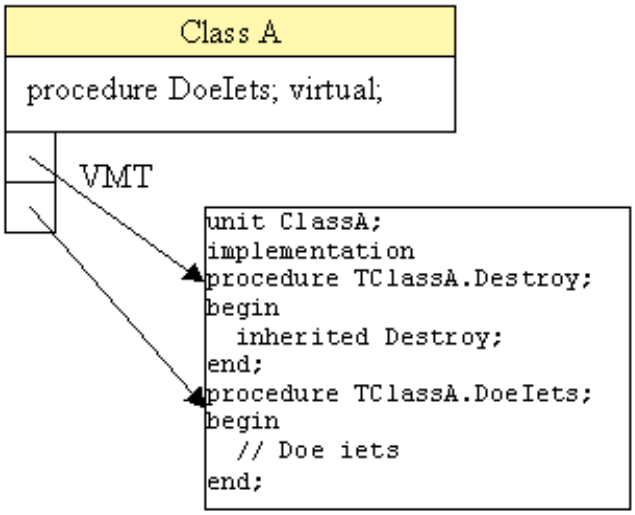
```
procedure TClassC.DoeIets;
begin
  // Voer de handelingen van de grootouder uit.
  TClassA(Self).DoeIets;  // Recursieve aanroep, stack overflow
end;
```

Hiervan kan ik je (uit ervaring) meedelen dat dit niet gaat werken. Omdat het hier om een virtual procedure gaat, bepaalt niet het type dat je in de typecast gebruikt welke procedure wordt uitgevoerd, maar bepaald het object zelf, aan de hand van zijn ClassType, welke procedure wordt uitgevoerd. Elk object heeft namelijk een property ClassType van het type TClass, die aangeeft tot welke class het object behoort. De bovenstaande procedure voert daarom niet de procedure DoeIets van Class A uit, maar de procedure DoeIets van Class C: een recursieve aanroep!

Virtual Method Table

Als het met een typecast niet werkt, hoe dan wel? De oplossing moet gezocht worden in een Virtual-Method-Table of afgekort VMT .

Zo'n VMT is een tabel, behorend bij een class, waarin verwijzingen staan naar alle procedures die 'virtual' worden gedeclareerd. Als je tijdens uitvoering van een programma een virtual procedure van een class uitvoert, wordt in de VMT van die class opgezocht wat het adres van de procedure is, waarna de procedure wordt uitgevoerd. Schematisch is dit te zien in de figuur hieronder.



Elke class heeft zijn eigen VMT. Een class die erft van een andere class, heeft tenminste dezelfde grootte en betekenis als de VMT van de ouder, eventueel aangevuld met extra virtual procedures. Als de procedure DoeIets van Class A de tweede regel in de VMT van Class A is, dan weet je zeker dat de procedure DieIets van Class B ook op de tweede regel in de VMT van Class B staat. Een class die erft van een andere class, bevat altijd een kopie van de VMT van de ouder, alleen zijn de verwijzigen anders voor alle virtual procedures waar een 'override' procedure voor bestaat.

Deze VMT techniek stelt je in staat om een variabele van het type TClassA te hebben, waar een instantie van Class B in zit. Als je nu een virtual procedure van het object in die variabele aanroept, dan wordt niet de procedure van Class A uitgevoerd, maar wordt eerst in de VMT van Class B gekeken en wordt de procedure uitgevoerd die daar staat vermeld.

Oplossing 1: verander het ClassType van een object

Eerder in dit artikel heb ik al verteld dat het ClassType van een object bepaald welke VMT wordt gebruikt voor het opzoeken van de implementatie van een virtual procedure. De property ClassType is een readonly property; logisch natuurlijk, een object is een instantie van een bepaalde class en daarom kun je de class niet wijzigen.

Het is echter wel mogelijk om de ClassType van een object te veranderen, maar pas als je weet dat de eerste positie (4 bytes) in het geheugenblok van een object de ClassType van het object bevat. Met deze kennis kan ik het probleem van het aanroepen van een procedure van de grootouder oplossen. Zie onderstaande procedure DoeIets van Class C, die met behulp van het type PClass de ClassType wijzigt in die van de grootouder, dan de procedure DoeIets aanroept en daarna de ClassType weer terug zet in de originele ClassType. Let erop dat je nu niet de inherited DoeIets aanroept, dus inherited weglaten.

```
type
  PClass = ^TClass;

procedure TClassC.DoeIets;
begin
  // Wijzig ClassType in de grootouder TClassA
  PClass(Self)^ := TClassA;
  try
    // Voer de handelingen van de grootouder uit. Geen inherited!
    DoeIets;
  finally
    // Zet de originele ClassType weer terug
    PClass(Self)^ := TClassC;
  end;
  // Voer nog wat extra handelingen uit.
  ShowMessage('TClassC.DoeIets');
end;
```

Oplossing 2: verander de VMT van de class van een object

Bij oplossing 1 wordt het ClassType gewijzigd, waardoor een andere VMT wordt gebruikt. Het alternatief is om niet een andere VMT te gebruiken, maar de eigen VMT te wijzigen. Oplossing 2 is niet beter dan oplossing 1, maar wel interessant om nog meer inzicht te krijgen in de VMT.

Voor oplossing 2 maak ik gebruik van de unit RxHook die onderdeel is van de RX library; een vrij te gebruiken library van componenten en handige functies. De RX library is te downloaden vanaf http://www.rxlib.com/ die op het moment van schrijven niet bereikbaar is, omdat de maandelijkse bandbreedte bereikt is; begin volgende maand nog maar eens proberen dus. De RX library werkt niet

Nieuwe redacteur

Johan Parent, nieuwe redacteur met aandacht voor Delphi.

met Delphi 6, maar op http://www.oxygensoftware.com/ kun je een poort voor Delphi 6 downloaden. De unit RxHook bevat de functie `FindVirtualMethodIndex` die gegeven een ClassType en een pointer naar een method van die class de index van die method in de VMT terug geeft. De tweede functie die ik gebruik is `SetVirtualMethodAddress` waar je een ClassType, een index in de VMT en een pointer naar een method aan meegeeft; de functie zet de pointer in de VMT.

Het is belangrijk dat deze procedure virtual is, zodat de procedure in de VMT wordt opgenomen

Class C krijgt er een extra virtual procedure bij, die ik hier even procedure ClassA_Doelets noem. Het is belangrijk dat deze procedure virtual is, zodat de procedure in de VMT wordt opgenomen. De VMT voor de procedure ClassA_Doelets wordt gewijzigd zodat die procedure naar de procedure Doelets van Class A verwijst.

Bedenk dat een VMT bij een class hoort. Het wijzigen van een VMT heeft dus gevolgen voor alle objecten die een instantie van deze class zijn. Om deze reden wordt het wijzigen van de VMT niet in een procedure van de class gedaan, maar in de initialization van de unit waar de class in geïmplementeerd is. De initialization van de unit van Class C in dit voorbeeld ziet er als volgt uit:

```
initialization
  SetVirtualMethodAddress(
    TClassC,
    FindVirtualMethodIndex(TClassC, @TClassC.ClassA_DoeIets),
    @TClassA.DoeIets);
```

Hier staat dat de VMT van TClassC wordt gewijzigd. De index in de VMT is die van de extra procedure ClassA_Doelets. Deze plaats in de VMT wordt vervangen door een pointer naar de procedure Doelets van TClassA.

De procedure Doelets van Class C kunnen we nu als volgt schrijven:

```
procedure TClassC.DoeIets;
begin
  // Voer de handelingen van de grootouder uit.
  ClassA_DoeIets;
end;
```

In deze procedure wordt dus in plaats van ClassA_Doelets de procedure Doelets van Class A gestart, waarmee we hebben bereikt dat de procedure Doelets van Class B wordt overgeslagen.

Marco Hemmes noemt zich IT Professional bij Bergler ICT Meer informatie over Bergler en Marco is te vinden op www.bergler.nl

auteur
Mike French

Understanding Certificates

Up until a few months ago I knew very little about certificates. Sure, I had some vague idea about what certificates were. I had heard of companies like VeriSign, and even had a number of friends who work in the security fields who lived and breathed certificates. But being a developer of PC software in a fairly small company, I didn't feel that I had a real need to know much more.

What changed this was working with Microsoft's new Office XP. Amongst the myriad of minor improvements that Microsoft has made to the Office suite with the XP release is a much greater emphasis on security. In fact, it could be said that Microsoft has gone from being completely lax with security to going overboard with security.

So having better security is good right? Well it is if it doesn't stop you doing your work.

One of the biggest security concerns is of course viruses. Most viruses come in the form of executables, dll's, or VB script that are transferred by mechanisms such as e-mails and Word for Windows documents. In a bid to make Outlook and the rest of the office suite more secure, Microsoft has placed tremendous restrictions on what can be done with these sorts of files. For example, as developers it is common for us to e-mail an executable to a customer or associate. A simple process that

Microsoft has gone from being completely lax to totally overboard

many of us perform even daily. With Outlook XP and even Outlook 2000 with Service Pack 2, this is no longer possible without an understanding of the new security mechanisms Microsoft has put in place. What happens now is that the recipient of your application will not be able to execute the application or open the VB script even if it is saved outside of Outlook. In fact, with Windows XP, even if you copy the file to another machine, you will still not be able to run that application.

The second problem I hit was with regard to a small Microsoft Word add-in that our company develops. While the Word add-in component is only a small part of our product, it is an important feature that our customers greatly appreciate. The plug-in is in the form of a VBA script or macro. When we attempted to run this plug-in in Word XP we received a message notifying us that macros were disabled and that before we could execute this macro, we would have to enable them. My first reaction was "Well, that's not so hard." I selected 'tools', 'macros,' 'security', and set the security level to low. The screen says that using that setting will allow even unsafe macros to run. However, our plug-in still wouldn't work.

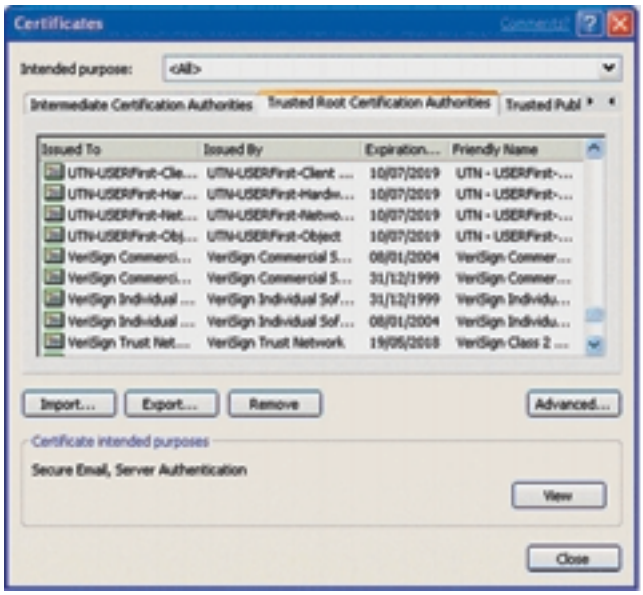
Understanding certificates was going to become necessary regardless of whether or not I needed this sort of protection

By now I was starting to realise that understanding certificates and the associated security concepts, was going to become necessary regardless of whether or not I felt I needed this sort of protection. Unfortunately, I have since discovered that while the concepts of certificates and encryption are fairly simple to understand, putting them into practice is another thing all together. In this article, I will introduce you to the fundamentals of certificates and security. In later articles, I will go into more depth as to how to get your own certificates and utilise them effectively.

Certificate Trees

Certificates, or digital ID's as Microsoft sometimes calls them, are designed to overcome two major obstacles. Authentication i.e. establishing that the person you are talking to online is in fact who they say they are, and encryption that is ensuring that you can communicate with that person without your messages or code being intercepted or modified.

The authentication process works through what is called a certificate tree. Essentially you start with a trusted source such as a security vendor like VeriSign. They create for themselves a special root certificate. This certificate is encrypted and contains details of what that certificate can be entrusted for. Eg E-mail protection, code signing, secure IP etc. These root certificates have signatures which are shipped with Microsoft Windows and various applications and are referred to as trusted root certificates.



If you would like to have a certificate that establishes your identity, you can then apply to a company like VeriSign and ask them to issue you with your own digital certificate. For this you pay a fee and this covers the cost of the certificate issuer doing checks to establish that you are who you say you are. In much the same way as the government performs checks on you to establish your identity before issuing you with a passport. Depending on the level of security you require, the requirements on you to establish your identity may vary. Once the certificate issuer issues you with your certificate, they are vouching for your credentials. Obviously it is important for these companies to maintain a high level of credibility and so these checks can sometimes be quite thorough.

When they issue you with your certificate, that certificate will include in it, all other certificates necessary to get back to the original root certificate. In essence, a tree is created with each lower level of the tree depending on the level above it for its authentication integrity. When you install your certificate on your machine the system checks the root certificate in the certificate tree within your certificate against the trusted root certificate signatures that were shipped with your applications. If the root certificate in your certificate matches one of the trusted root certificates, then your certificate is verified as authentic and your identity therefore trusted. In the same way that you presenting a foreign government with your passport will generally be sufficient to establish your identity through the fact that your passport was issued by your government.

Encryption

An essential part of securing both the certificates themselves and communications once your identification has been established is the process of encryption. Certificates use a technique called public/private key pairs. A public/private key pair are two special numbers each a certain, though usually different number of bits in length eg. 40 bits, 128 bits, 1024 bits etc. that are used to encrypt and decrypt data.
Essentially they work like this: one of pair of keys is ued to encrypt data and the other can then be used to decrypt it. If for example you encrypt an element of data with a public key, you need the matching private key to decrypt that bit of data. Conversely, if you encrypt that

FoxPro Tip

Nogmaals SQL-SELECT in VFP 7

In Visual Foxpro 7 is het SQL SELECT statement uitgebreid met een READWRITE clause.
Als je in eerdere versies van (Visual) FoxPro een cursor wilde creëren waarin je vervolgens wilde schrijven, dan moest je de cursor met een truc opnieuw openen. Stel dat de gecreëerde cursor query heet, dan kon je m.b.v. het statement "USE DBF('query') IN 0 AGAIN ALIAS query_rw" de cursor query onder de nieuwe naam query_rw openen, en in deze alias kon wel geschreven worden. Het gevolg was dat de wijzigingen ook in de cursor query terecht kwamen.
In VFP7 is dit niet meer nodig dankzij de READWRITE clause, waarmee je kunt aangeven dat de te creëren cursor ook voor schrijven toegankelijk gemaakt moet worden.

data with the private key you need the public key to decrypt it. Private keys cannot be used to decrypt data encrypted with that same private key, nor can a public key be used to decrypt data encrypted with the same public key.
If you therefore encrypt something with your private key and provide that data to somebody else, they must poses your corresponding public key in order to decipher it. The trick is that while your may provide other people with your public key (in fact there are public registries where you can locate other peoples public keys), your private key should always remain confidential and well guarded.
How then, do you exchange information with somebody else? If they need your public key to decrypt data encoded with your private key and you are making your public key widely available. The answer is if you are sending data to somebody else you don't use your own private key to encrypt data at all. Instead you exchange public keys with the party at the other end. Any data you are sending to them, you encrypt with their public key. Even if somebody else has access to their public key and intercepts your message, they cannot decrypt that data without the other persons private key. Conversely, when the other person send you information they utilise your public key to encrypt the data and when you receive that data, your private key is used to decrypt it. In this way, we can establish secure two way communications.

This public/private key process is also used to protect the certificate chain itself to ensure that people can verify the authenticity of a certificate chain while at the same time preventing them from decrypting and hence forging a certificate chain. Since the protection of these private keys is so essential to the security process, the keys for root certificates such as those that VeriSign posses, are guarded by multi-million dollar security systems since anybody gaining access to them, could compromise the entire validation tree.

In future articles, I'll explain the details of purchasing and implementing certificates for e-mail and code signing.

Mike is the CEO and Chief Product Architect at Direct Marketing Software. Mike can be contacted by e-mail at mike@dmsw.com.au.

Tip

Wil je meer weten over certificaten?

Kom dan **13 december naar de CttP**. Mike French is dan aanwezig om je alle details uit de doeken te doen!

auteur
Paul Maskens & Andy Kramek

COM-monsense Design

Earlier published in FoxTalk

This month Paul and Andy are discussing the issues that arise when designing a class that has to be capable of being instantiated directly in Visual FoxPro and also as a COM Object.

Andy: I need some help with a rather tricky design problem that I have right now. Here is the situation. I need to design a class that can be used within a Visual FoxPro application to consolidate data which comes from multiple databases. This data can be manipulated in the application and can then be output in a variety of formats (e.g. either as Visual FoxPro data, XML or as an ADO RecordSet). The final complication is that the same class needs to be capable of being instantiated as a COM object too. Here is a high level diagram of the design requirement for our 'Data Controller Class':

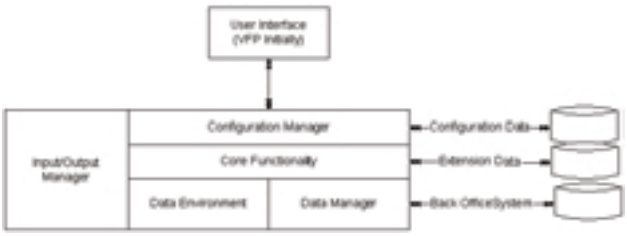


Figure 1:High Level Design

Paul: The 'Back Office System' that you have there can, I assume, be anything at all?

Andy: Exactly, it might be a Visual FoxPro application like "AccountMate", or an SQL based system like "Oracle Financials" as far as this class is concerned it is simply a data source. The Extension data is going to be a local Visual FoxPro database (and so is the 'Configuration ' data).

Paul: I see, so the 'Core Functionality' is to combine data from the back office system with data which is held locally in the 'Extension Data'. I presume, from the name, that the extension data provides additional information for specific functionality which is either not present in the back-office system or is not standardised between different systems. The Visual FoxPro front end is then going to be used to manipulate this data and export the combined results through the Input/Output Manager. And since you have named this an Input/Output Manager I guess that there is going to be data coming the other way too, or is this just an export process?

Andy: Oh yes, data will be coming the other way too. We could be receiving data in as many different formats as we need to export it. The only limitation is that we will not be attempting to update back office data directly - the only thing that we need to import updates for will be the extension data. An additional wrinkle is that this data needs to be configurable, and so the Configuration database is going to be used to keep track of what fields are being used and how they are mapped to the interface tables.

Paul: But if this is all a Visual FoxPro application I cannot see the need for instantiating the thing as a COM component at all, unless you are proposing that some other interface be used to manipulate the extension data. Oh, I bet you want to be able to put this up on a corporate intranet at some point?

Andy: Exactly right. Although the initial intention is that this should be a Visual FoxPro application, there could come a point when the data would have to be made available to a wider community. Since the possibility is known to exist we should not be closing any doors in our design. Now, the public interface consists of only four methods as follows:

- OpenDataSet()
- CloseDataSet()
- FetchData()
- SendData()

The functions are pretty self-explanatory I think. The "Open" and "Close" dataset methods establish a connection to a back end and (using the configuration data) link in the extension data while the "Fetch" and "Send" methods are used to handle the link to core functionality and the Input/Output functions. All of the various different parts of the core functionality and the different Input/Output options will be handled by separate classes so that for a given request I can just instantiate the relevant "functional object". Now here is my specific problem.

The class uses Visual FoxPro cursors in its internal data environment. So when we are connected to a Visual FoxPro front end there is actually no need for us to reformat data for the interface (after all Visual FoxPro can use these cursors directly). In this situation all I need to do is to return a reference to the data. But when this class is instantiated as a COM component there is no guarantee that the interface will be able to use VFP cur-

sors and, anyway, the object should be stateless and should not be holding on to properties or data between calls to its functions and so must return the actual data, not just a reference to it. How can I reconcile these two requirements?

Paul: Firstly you have to ensure that the interface can provide sufficient information to enable the class, however it was instantiated, to carry out its function. Unless you know some magic that will tell you whether the class was instantiated by VFP's CREATEOBJECT() or as a COM Object using VB's CREATEOBJECT() you must define how the data is to be returned. Now, I'd do that by just passing a parameter which specifies the type of data that the calling object is expecting. It is then as simple as passing either 'VFP' or 'VB'?

Of course you will also need to pass something which identifies what the request is actually for. Presumably you will also need parameters to tell you which configuration data to use and where to return the results to.

Andy: We can do it all with parameters then - that works for me! As you said, that means that I will need:

- The specific data that is required. This defines which functional object will be instantiated.
- The configuration Dataset to use.
  This defines where we will get the data from.
- The format which we are using.
  This defines the specific I/O format that we need.
- A backward reference to the calling object so that we can return the results.

Paul: To me this seems an ideal place to use a Strategy pattern. Incidentally, I don't see why your input/output manager, has to be a single object. In the same way that your data manager uses multiple connection managers (I read your data classes article <g>) then surely you'd want to avoid an 'itdoesabsolutelyeverything' I/O manager? I think that you would need that even if it is only to simplify the maintenance. So when, six months down the track, the clients ask for a additional type of output you only need to update the control table.

Andy: What control table? I wasn't actually planning on data driving this part - although I can see why you might want to. My intention was to name the methods in the I/O manager after their output type. For example the VFP method would be named 'IOVFP' and we would have 'IOVB' and 'IOXML' for VB and XML output respectively. Then, when called with the format, the controller simply appends that format code to 'IO' and looks for the appropriate method.

Paul: That would work, though I'd rather instantiate different classes than use methods. When you need to add another output type (and you surely will <g>) there is no need to change the actual code - merely add another class. The only objection to that approach is that you then have to handle the VFP Errors that arise when the specified method (or class) doesn't exist. Whereas if you use the table based approach then you only need to handle a SEEK() failure. It's just personal preference though.

Andy: Seems that either way will work just fine. But I take your point on table-driving it - I will do it that way. My next question is where, in this I/O manager, is the functionality going to be?

Paul: In the IOVB class or IOVFP class is my first thought. But that's not how your data classes work. You have the functionality split between the Data Manager, the Connection objects and the Behaviour objects.

Andy: Right, so what's your answer?

Paul: Well I'd say that the data classes have the better design, not just because it's yours <g>. It is of course more complex and takes more time and effort to get the design right, but it's worth it in the long run. I can see at first there is a temptation to put all the functionality into the IOVB and IOVFP classes, particularly for those to insist on "Design At The Keyboard". Hmm, have I just condemned myself as a DATKer?

I'm going to stick my neck out here and say that you ought to apply your "must, should, could" rules that you keep quoting to me.

Andy: Very well, for a start the I/O manager must define the public interface (used by the configuration manager). It should handle the common behaviours necessary to handle the data and present it to I/O formatter. It should create the appropriate I/O formatter instance. It could handle things like transferring files.

Paul: I definitely think that last one ought to be a separate class, or in fact set of classes. Your data transfer is likely to range from a simple copy in the same computer to FTP transfer to a remote host. They may not even be in the same platform (Unix, Linux, NT, or even Mac). Unlike your data classes though, I think you only need two layers, not three.

we can agree with their use of terminology

Andy: Yes, there's no requirement for a component analogous to the connection object in the data classes. I think we can now draw an implementation diagram (Figure 2). As you can see, the interface object determines which functional component to instantiate

when a request is received, based on the type of request. The functional component communicates with the data source and passes the results back to the interface as a VFP cursor. The interface object then hands this cursor off to the I/O manager for formatting and that object instantiates the appropriate formatter for the requested data type. Thanks Paul, that has helped clarify things for me.

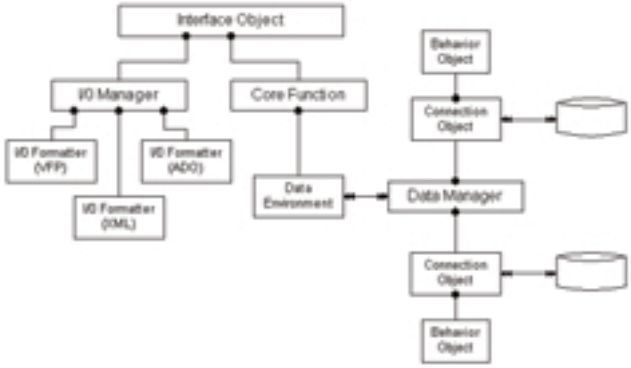


Figure 2: The Implementation of the High Level Design

Paul: No problem. Now, I have a follow-on question for you. Earlier you said "the object should be stateless". That's something that I really don't understand well enough. State has to be stored somewhere after all. For example, in the interface we have been discussing the same parameters are passed every time and must, therefore, be coming from, and hence stored by, the client. Some of these values could be stored by the object, but I suppose that that would mean that only one client can ever use the object. So is the object 'stateless' because the state is being stored by the client, or is it because the object is stateless that the client has to store the state?

Andy: I'm going to quote Rick Strahl from his book "Internet Applications with Visual FoxPro 6.0" (Hentzenwerke Publishing, 1999). He defines a stateless server as one that "does not retain any context for the client. The server does not know what action the previous request performed and does not make any assumptions about the current environment."

So I guess that this is a stateless object because it does not store anything about the client, and does not make any assumptions about the current environment. Any client that wants to interact with this object must, therefore, maintain its own state information.

Paul: Well, that is what Microsoft (elsewhere) refers to as "Client-managed state" and it looks as though, for once, we can agree with their use of terminology.

Andy: That's a novelty <g> To my way of thinking, they key to all of this is the concept of "persistence". A stateless object "does it once, and throws it all away". While a stateful object keeps information in case it's useful next time.

Paul: Ah, but as you have described it, your object keeps the configuration data internally. Now isn't that "state" information in some sense?

Andy: Only in the context of the object itself. The configuration data has nothing to do with any individual client. It is only used to support the internal operations of our object. It therefore is created when the object is first instantiated and shares the lifetime of the object.

always good when writing this column to reduce my ignorance level

Paul: If the object does not change that data, and in its operation doesn't rely on index orders, record pointer values and other states that it gets into while processing a request from a client - I can see your point.

Andy: It definitely does not rely on any of those things. It will always look up the values it has been passed as parameters. We are only holding this data as an optimisation to avoid the necessity of reopening the same tables in the same workspace for each request the object receives. This data is, in the context of the lifetime of the object, static.

Paul: Thanks! it's always good when writing this column to reduce my ignorance level.

Andy Kramek is an old FoxPro developer, FoxPro MVP, independent contractor and occasional author originally English but now based in Akron, Ohio, USA.
Email: andykr@compuserve.com

Paul Maskens is a VFP specialist and FoxPro MVP, working as Internet Systems Manager for Euphony Communications Ltd, he's based in Oxford, England.
Email: pmaskens@compuserve.com

auteur
Mark Vroom

RI builder in de bocht

Na grondig onderzoek vanwege problemen in onze import routines blijkt er helaas weer een fout te zitten in de RI-Builder. Ik heb in vorige SDGN magazines al eens eerder over problemen met betrekking tot de RI builder geschreven en er zijn ook wel diverse tips over geplaatst. Dat had onder andere te maken met de kans op 'Orphan' records, child records zonder parent en op het niet in vullen van Integer velden waarop een relatie ligt in de DBC met een parent tabel.

Inmiddels is gebleken dat zich ook een bug bevindt in de RI-Builder in de Insert en Update methods waar je last van krijgt met 'vrijwel gelijknamige' tabellen.

Om inzicht te krijgen in de bug is het belangrijk om eerst te begrijpen hoe de code van de RI-Builder in grote lijnen werkt. De door de RI-Builder gegenereerde code zorgt ervoor dat bij het aanmaken van nieuwe records, het verwijderen van records en het updaten van records constant een test plaatsvindt of bij een child record wel een overeenkomstig parent record bestaat. Afhankelijk van de gekozen actie (Delete/Add/Update) worden een of meerdere parent records behandeld.

Voor elke tabel worden een drietal functies aangemaakt in de Stored Procedures van de DBC met als naam __RI_<actie>_<tabel>. Voor <actie> wordt in geval van toevoegen, wijzigen en verwijderen respectievelijk 'Add', 'Update' en 'Delete' ingevuld. In het geval van bijvoorbeeld een Update functie voor een tabel 'relaties' zal de functienaam dan dus __RI_Update_Relaties worden. Op die manier worden er voor de Delete, Add en Update acties telkens drie functies aangemaakt voor elke tabel.

Aangezien de bug zich bevindt in de Update en Add functies zullen we ons beperken tot het uitleggen van de werking daarvan. De __RI_Update en __RI_Add functies zijn bovendien vrijwel identiek, de bug is dan ook in beide methods aanwezig en we behandelen beide functies dus even alsof ze gelijk zijn.

De beide functies moeten voor een opgegeven child tabel altijd een of meerdere parent tabellen controleren. Zo kan aan een klant bijvoorbeeld een faktuur gekoppeld zijn. Nu bestaat de mogelijkheid dat de faktuur tabel niet open is. Daarvoor heeft de RI-Builder een generieke functie 'RiOpen' gegenereerd. In die functie worden alle benodigde parent tabellen bij een child geopend. Elke tabel die tijdelijk geopend wordt krijgt een alias naam, bestaande uit __RI<workarea> waarbij <workarea> het nummer is van het werkgebied. Tevens wordt er een string pcRiCursors bijgehouden waarin alle tabellen staan die geopend zijn, gevolgd door een * en het workarea nummer. Die string zou er als volgt uit kunnen zien voor de tabel 'relaties':

```
pcRiCursors: fakturen*2     offertes*3     contactregistraties*4
```

De string bevat zoals gezegd alle tabellen die tijdelijk geopend zijn, per tabel gevolgd door het werkgebied nummer. Om nu te voorkomen dat de environment vol loopt met tijdelijk geopende tabellen moet de RiEnd functie er voor zorgen dat alle tabellen uit de string pcRiCursors weer netjes gesloten worden. Om dat te krijgen wordt elk * teken in de string opgezocht en wordt het getal daarachter uit de string herleid. Vervolgens wordt de tijdelijke naam opnieuw samengesteld (in geval van 'fakturen' is de tijdelijke naam '__RI2' geweest, de tijdelijke alias naam __RI gevolgd door werkgebied 2) en wordt er een USE IN <workarea> (USE IN 2) gegeven om die tabel te sluiten.

De bug die optreedt vindt alleen plaats bij vrijwel gelijknamige tabellen. Onderstaande voorbeeld string zal bijvoorbeeld mis gaan:

```
onderwijselementen*2     type_cfi_schoolsoorten*3     cfi_schoolsoorten*4
```

De eerste opdrachtregel van RiOpen() is als volgt:

```
lnInUseSpot=atc(tcTable+"*",pcRIcursors)
```

Dit commando geeft de problemen. Wanneer je hier als tcTable 'cfi_schoolsoorten' naar binnen stuurt zal lnInUseSpot de positie van 'type_cfi_schoolsoorten' aan gaan geven als tabel die blijkbaar gezocht wordt. Dat komt omdat type_cfi_schoolsoorten* op zich hetzelfde is als cfi_schoolsoorten* voor een ATC() functie. Microsoft heeft nog wel een * toegevoegd achter tcTable, maar in bovenstaand voorbeeld zal dat geen verbetering opleveren. Met behulp van de foutieve waarde lnInUseSpot zal vervolgens in werkgebied 3 (van type_cfi_schoolsoorten) gezocht gaan worden naar parent records, wat in veel gevallen vervolgens trigger foutmelding gaat opleveren, omdat het eenvoudigweg niet de juiste tabel is die gebruikt zou moeten worden!
Dit probleem is eigenlijk alleen maar op te lossen door de string pcRiCursors anders te gaan samenstellen. Eigenlijk zou elke tabelnaam ook voorafgegaan moeten worden door een * in plaats van alleen aan de achterzijde. Hoe zit het resultaat er dan uit?

```
*onderwijselementen*2    *type_cfi_schoolsoorten*3    *cfi_schoolsoorten*4
```

De eerste opdrachtregel van RiOpen zou dan moeten worden:

```
lnInUseSpot=atc("*"+tcTable+"*",pcRIcursors)
```

In dat geval zal wanneer tcTable 'cfi_schoolsoorten' bevat gezocht worden naar *cfi_schoolsoorten* en zal de juiste positie worden geretourneerd.

RiOpen()

De RI-Builder is als project file beschikbaar in de zip file XSOURCE.ZIP. Door deze project file te openen en de class 'Ribuilder.vcx' te openen kan in het 'Click' event van de 'OK' button gezocht worden naar de regel 'PROCEDURE riopen'. Vlak onder die regel staat de opdrachtregel

```
lnInUseSpot=atc(tcTable+"*",pcRIcursors)
```

Markeer of verwijder die regel en plaats de volgende regel daarvoor terug:

```
lnInUseSpot=atc("*"+tcTable+"*",pcRIcursors)
```

Deze regel voegt het beschreven extra * karakter toe aan de string pcRiCursors. Vervolgens moet de regel onder 'if pnerror=0' als volgt worden vervangen:

```
*   pcRIcursors=pcRIcursors+upper(tcTable)+"?"+STR(SELECT(),5)
pcRIcursors=pcRIcursors+'*'+upper(tcTable)+"?"+STR(SELECT(),5)
```

Ook hier wordt opnieuw het * teken voor tcTable geplaatst. Hiermee is het gedeelte aangepast wat aangeroepen wordt als een tabel tcTable niet in gebruik is en om die reden in gebruik moet worden genomen. Dan is er ook nog de situatie dat tcTable al in gebruik is, het gedeelte na het nu volgende 'Else' commando:

```
*  lcNewWkArea=val(substr(pcRIcursors,lnInUseSpot+len(tcTable)+1,5))
lcNewWkArea=val(substr(pcRIcursors,lnInUseSpot+1+len(tcTable)+1,5))
*  pcRIcursors = strtran(pcRIcursors,upper(tcTable)+"*"+str(lcNewWkArea,5),;
*   upper(tcTable)+"?"+str(lcNewWkArea,5))
pcRIcursors = strtran(pcRIcursors,"*"+upper(tcTable)+"*"+str(lcNewWkArea,5),;
  "*"+upper(tcTable)+"?"+str(lcNewWkArea,5))
```

Ook hier wordt opnieuw een * voorafgaand aan tcTable geplaatst.
Nu de volledige RiOpen functie is aangepast moet RiEnd nog worden aangepast, omdat deze functie op een soortgelijke manier gaat proberen om alle tijdelijk geopende tabellen weer af te sluiten.

RiEnd()

In de RiEnd functie worden alle tijdelijk geopende tabellen weer gesloten. Daarvoor wordt, zoals eerder beschreven, het werkgebied nummer uit pcRiCursors gehaald. Daarvoor wordt een For/Next lus gebruikt op basis van het aantal gevonden * tekens. Door de aanpassing in pcRiCursors is het aantal * tekens twee keer zo groot geworden. Om die reden is de For/Next lus als volgt aangepast:

```
*FOR lnXX=1 TO occurs("*",pcRIcursors)
FOR lnXX=2 TO occurs("*",pcRIcursors) step 2
```

Op deze manier begint de For/Next lus met het 2e gevonden * teken (dus het * teken áchter de eerste tabelnaam en stapt vervolgens naar het 2e daarop volgende * teken. Op die manier wordt elke tabel uit pcRiCursors netjes uit de string gehaald en kan die tabel worden gesloten.

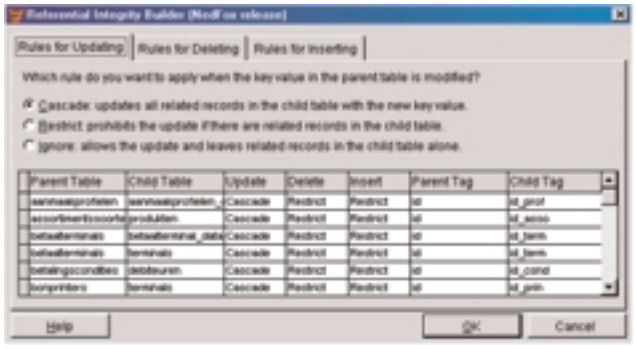
RiReuse()

Als laatste aanpassing moet de functie RiReuse() nog worden herschreven. Deze functie wordt op diverse plekken in de Ri-Builder software gebruikt om een ? karakter te vervangen door een * karakter. De reden hiervoor ontgaat mij enigszins, maar het is in ieder geval belangrijk dat pcRiCursors nog steeds consequent van een voorloop * wordt voorzien, zodat ook deze functie goed zal werken.

Conclusie

De gerapporteerde bug is niet een die je snel tegen zal komen. Ook binnen de Foxpro user groups heb ik hier eigenlijk nog nooit iets over gehoord. Zoals je hebt kunnen lezen zal de bug ook alleen maar onder bepaalde condities optreden, maar wij hadden toevallig net die twijfelachtige eer.

De RiBuildr.app file moet geplaatst worden in de directory \program files\microsoft visual studio\vfp98\wizards waarna FoxPro opnieuw opgestart moet worden.
Mocht je de RI-Builder op de juiste manier geplaatst hebben, dan herken je dat doordat de titelbalk van het RI-Builder scherm nu de tekst 'NedFox release' aangeeft in plaats van de standaard Visual FoxPro tekst.



De aangepaste RI-Builder software waarin ook de in eerdere SDGN Magazines beschreven tekortkomingen zijn opgelost is te downloaden van de SDGN website, ribuilder69.zip.

Mark Vroom is werkzaam bij NedFox B.V. te Emmeloord als programmeur. NedFox houdt zich bezig met het ontwikkelen en ondersteunen van POS software (Point Of Sale). Na de Euro conversie eind van dit jaar zal begin volgend jaar gestart worden met de implementatie van Pos4All, de Windows versie van NedFox Kassa bij onder andere GroenRijk, Europatuin en Tuinspectrum tuincentra in Nederland en Belgie. mark@nedfox.nl

Questions and Answers

Internet

q Hoe moet ik de namespace binnen een xmldocument wijzigen ?

a Soms is het wenselijk om ten behoeve van schemavalidatie de namespaces binnen een XMLdocument te veranderen.
Binnen DotNet kan dit eenvoudig met de volgende code: Na opslaan ziet je XML er zo uit:

```
   XmlDocument objXMLDoc = new XmlDocument();
   objXMLDoc.LoadXml("<test
xmlns='123'><item>abc</item></test>");

objXMLDoc.DocumentElement.SetAttribute("xmlns","urn:123");
```

<test xmlns='urn:123'><item>abc</item></test>

Je zult moeten op opslaan en herladen om tegen het nieuwe schema te valideren.

Delphi

q Hoe kan ik de gebruikte ADO-versie te weten komen?

a Er zijn 2 methoden:
1. programmatisch via de version property van het ADO-connection component.
2. Dowloand de microsoft ComCheck utility van

```
ShowMessage(ADOConnection1.version);
```

http://a118.ms.a.microsoft.com/f/118/1611/2h/download.microsoft.com/download/dasdk/Utility/2.61/WIN98Me/EN-US/cc.exe .

Deze utility scant de computer oo inconsistency, verkeerde en ontbrekende files en register waarden en rapporteert de MDAC versie.
Het is nuttig om het register op fouten te controleren. Meer weten over deze utility: ga naar http://www.microsoft.com/data/download.htm

Visual FoxPro

Visual FoxPro instellingen bewaren

q Het 'Options' tabblad in VFP bevat allerlei instellingen die in de registry worden opgeslagen.
Helaas kan je niet aangeven op welke plaats die gegevens terecht moeten komen, ze worden allemaal bewaard in HKEY_CURRENT_USER in Software\Microsoft\VisualFoxPro\6.0\Options.

Wanneer je echter werkt met meerdere projecten kan het wel eens nodig zijn om een aantal specifieke instellingen per project te bewaren. Denk hierbij vooral aan het onderdeel 'Field mappings' wanneer je nog te maken hebt met oude en nieuwe baseclasses. Met onderstaande leesregistry.prg en writeregistry.prg kan je alle instellingen van VFP uit de registry opslaan in een INI file en kan je de INI file weer terugzetten naar de registry. Door per project een INI file aan te maken kan je vervolgens met weinig handelingen VFP per project eigen instellingen laten toepassen.

a

```
* writeregistry.prg

lparameters tcINI_file

#include 'c:\program files\microsoft visual ~
  studio\vfp98\ffc\registry.h'

* De INI files staan standaard in de Windows directory.
* Geef als parameter de volledige padverwijzing op naar de te
* parsen INI file. De INI file kan bijvoorbeeld ook op diskette
* gezet worden.

? 'Openenen INI file...'
lnReadHandle=fopen(tcINI_file)
if lnReadHandle=-1
  wait wind 'oh oh, probleem bij openen INI file'
  return -1
endif

set class to 'c:\program files\microsoft visual studio\vfp98\ffc\registry'
oReg=createobject('Registry')
oINI=createobject('OldIniReg')

lcSection='Software\Microsoft\VisualFoxPro\6.0\Options'

* Zoek de sectie lcSection op in de INI file en tel het aantal regels
* wat daar onder staat
lnRegels=0 && Aantal regels in sectie gedeelte
llSectieStart=.F.  && Indicatie start van gevonden sectie gedeelte
? 'Scannen INI file...'

do while !feof(lnReadHandle)
  lcRegel=fgets(lnReadHandle)
  if llSectieStart and (empty(lcRegel) or substr(lcRegel,1,1)='[')
    * Sectie einde of start nieuwe sectie terwijl oude nog actief was,
    * sectie is blijkbaar gevonden
    exit
  endif


  if lower(lcRegel)=lower('['+lcSection+']')
    * Sectie begint
    llSectieStart=.t.
    loop
  endif

  if llSectieStart
    * Schrijf deze regel in de registry
    lcSleutel=substr(lcRegel,1,at('=',lcRegel)-1)
    lcWaarde=substr(lcRegel, at('=',lcRegel)+1)
    ? 'Wegschrijven sleutel '+alltrim(lcSleutel)+;
      ' met waarde '+alltrim(lcWaarde)+' naar registry...'
      =oReg.SetRegKey(lcSleutel, lcWaarde, lcSection,;
      HKEY_CURRENT_USER, .T.)
  lnRegels=lnRegels+1
  endif
enddo

? 'Sluiten INI file...'
=fclose(lnReadHandle)

* Lees de registry settings nu opnieuw in zodat de INI file
* actief wordt
? 'Opnieuw inlezen registry settings...'
=sys(3056)
? 'Gereed...'

lparameters tcINI_file

#include 'c:\program files\microsoft visual studio\vfp98\ffc\registry.h'
```

```
* Parameter: Naam van de aan te maken INI file. Geef hierbij
* beslist geen drive en/of pad op, de INI file komt altijd in de
* Windows directory te staan!

* Deze functie opent de registry instellingen en bewaard die in een
* aan te geven INI file. Hierdoor wordt het mogelijk om met
* verschillende projecten te werken die ieder hun eigen specifieke
* VFP omgeving nodig hebben.
* Vooral bij het werken met veroudere classlibraries voor bepaalde
* projecten kan het handig zijn om bijvoorbeeld de fieldmappings
* snel om te kunnen zetten naar de oude class libraries. Door in
* dit geval te werken met een aparte INI file waarin deze
* instellingen staan kan snel worden doorgewerkt aan
* het verouderde project.

set class to 'c:\program files\microsoft visual studio\vfp98\ffc\registry'
oReg=createobject('Registry')
oINI=createobject('OldIniReg')

dime aTest[1]
=oReg.eNumOptions(@aTest,; ~
  "Software\Microsoft\VisualFoxPro\6.0\Options",
  HKEY_CURRENT_USER)

* Bepaal lengte van array en scan alle elementen
aArrayLengte=alen(aTest,1)

? 'Scannen registry...'
for lnTeller=1 to aArrayLengte
  lcKey=aTest[lnTeller,1]
  luValue=aTest[lnTeller,2]
  ? 'Wegschrijven sleutel '+alltrim(lcKey)+;
    ' met waarde '+alltrim(luValue)+' naar INI file...'
    oINI.WriteINIentry(luValue,;
    'Software\Microsoft\VisualFoxPro\6.0\Options', lcKey, tcINI_file)
endfor

rele oINI
rele oReg
? 'Gereed...'
```

Deze sourcecode is ook te downloaden van de SDGN website onder de naam VFPSETUP.ZIP

Office 2000

q Is there an easy way to do a Word mailmerge for a subset of Outlook contacts?

a Outlook 2000 provides a new feature that allows you to do a mail merge with your contact list from within Outlook, whereas in previous versions of Outlook the mail merge had to be done from Word, and you had no way to filter or set up any type of merge criteria based on Outlook categories or other filtering options.
With Outlook 2000, you can pick specific contacts to merge. Word will still perform the merge, but the set up and filtering of your contacts takes place in Outlook.
Suppose you want to do a mailmerge for a couple of your contacts. After opening the Contacts folder, you could select the specific contacts by holding down the Ctrl key and select the wanted contacts. Then choose Mail Merge on the Tools menu. You will be presented a dialog where you can select several mail merge options, and after clicking OK, Word will be started and from this point on you will be working in Word, where you can use its regular mail merge functions. Use the Insert Merge Fields options to select from the Outlook contact fields.

Internet Explorer

q Are there ways to speed up entering of addresses of websites?

a Yes. One way to speed up the entering of an address, is to just enter the main part, e.g. "microsoft", in the Address field, and then press Ctrl+Enter. IE will automatically add "http://www." as a prefix and ".com" as a suffix.

Agenda 2001//2002

Conference to the Point, het SDGN eindjaars event *De Reehorst in Ede*	**13 December**
SDGN Magazine nr. 70	**15 februari**
Conference to the Point + ALV	**22 maart**
Tech Ed 2002 USA, New Orleans	**9 t/m 13 april**
SDGN Magazine nr. 71	**12 april**
Conference to the Max 2002 *Koningshof in Veldhoven*	**13 en 14 mei**
BorCon 2002 USA, Anaheim CA	**19 t/m 22 mei**
SDGN Magazine nr. 72	**7 juni**
Tech Ed 2002 Europe, Barcelona	**1 t/m 5 juli**

Genoemde data onder voorbehoud

In memoriam

In de maand september van dit jaar is de heer **Joop Sargentini** plotseling komen te overlijden.

De heer Sargentini was een van de oprichters van de Dfudg, beter bekend als de FoxPro Usergroup, welke later is overgegaan in de SDGN als Foxpro Sectie.

Joop is ook binnen SDGN vanuit de Dfudg achtergrond nog enige tijd actief geweest. Iedereen die hem heeft gekend zal hem bestempelen als een aimabel mens die je desgewenst met raad en daad ter zijde stond.

Wij wensen de famillie en vrienden alle sterkte toe.

Ed van Akkeren

EEN HALF TOONTJE HOGER

Temidden van die zondvloed aan acroniemen, drie- en zelfs vierletterwoorden die de ICT nog altijd teistert heeft C altijd een bijzondere plaats ingenomen. Het was gewoon C, meer niet; de taal was al moeilijk zat. En toen ze C nog een beetje ingewikkelder wisten te maken werd het geen Turbo, Visual of ander superwhizzbang epitheton, maar gewoon plus plus, als een Maleis meervoudje.

Wat dus te doen met een nog verdere complicering van dat woud van accolades en boter-kaas-en-eieren spelletjes? De jongens van C hadden immers een naam op te houden. Nou, men is er weer in geslaagd om origineel te blijven. Ze hebben leentjebuur gespeeld bij de muziekwereld: C#, op zijn Engels C-sharp. De Cis: niet meer, maar hoger. Gaf C je een touw om je mee op te knopen en C++ een hele bundel, met C# gaan we dat allemaal nog een half toontje hoger doen.

Tenminste, als het aan MickeySoft ligt, want die positioneren deze nieuwste loot aan het programmeerpalet als het alternatief voor Java. Sun is daar uiteraard niet blij mee: hun Java zou toch de toekomst worden. Iedereen was het daarmee eens. Maar ja, MicroSoft heeft nu eenmaal lak aan iedereen. U koopt maar gewoon wat wij willen dat U koopt. Vrees niet: onze commerciële mensen zijn de besten ter wereld. U zult zich straks niet eens meer kunnen herinneren dat U ooit in Java hebt geloofd.

MicroSoft verkoopt C# natuurlijk als Het Antwoord op al Uw objectgeorienteerde problemen in de-omgeving-die-U-al-zo-goed-kende. The truth, the whole truth and nothing but the truth so help us Gates? Zeg nu zelf: zo 'anders' ten opzichte van C++ was Java toch ook weer niet? En Java had nog het voordeel dat het geen legacy uit een vorige incarnatie (C) met zich mee hoefde te slepen.

Sun maakt zich terecht zorgen: technologische superioriteit ten opzichte van de marketing-machine uit Redmond zegt immers niets. Maar erger is nog dat wij ontwikkelaars opgezadeld worden met weer een nieuwe programmeertaal. Niet gegroeid uit een professionele behoefte, nee, gewoon uit marktpolitieke overwegingen.

Maar wat ik me na al die tijd nog steeds afvraag is waarom er zoveel in C++ (al dat niet visueel aangekleed) wordt geprogrammeerd. Ja, ik weet dat de doorgewinterde C-veraar nu gaat roepen hoe krachtig en snel die taal wel niet is, maar ik heb C++ programma's eigenlijk nog nooit iets zien doen wat ik niet in een andere, door mij eenvoudiger te doorgronden taal, ook had kunnen doen. Het blijven toch combo-, list- en andere boxen, een kwestie van die database volploppen en dan weer het hemd van het lijf queriën. Dat kan toch overal mee? Wat maakt C++ zo bijzonder dat het een groot commercieel offensief waard is?

Volgens mij is het juist die complexiteit. Hoe kan je als klopper je riante salaris verdedigen als iedere onbenul na een tweedaags cursusje je code kan lezen. Nee, als het moeilijk was om te schrijven, dan moet het ook moeilijk te begrijpen zijn. Ziedaar het succes van C en vooral C++: welke ontwikkelomgeving kan een novice bij een 'Hello World' zo'n vijftien errors en nog veel meer warnings garanderen?

The MicroSoft strikes back!

Wacht even. Eureka! Dat ge-C-mel is gewoon onderdeel van een nieuwe marketingstrategie om de schuld van al die protectiefouten bij de programmeurs te kunnen leggen. Want waren het niet die keyboard-junks die de hetze tegen het glorieuze MicroSoft begonnen? En zijn het niet die sites waar die ontwikkelaars na werktijd inloggen waar de meeste anti-Bill moppen worden geboren? Tijd voor actie.
Als je de foutmeldingen maar cryptisch genoeg maakt (en daar zijn wij van MicroSoft natuurlijk meesters in), dan zullen ze natuurlijk nooit meer met zekerheid kunnen beweren dat ze uit Windows komen en niet uit hun eigen broddelwerk. En de officiële legende wordt uiteraard dat het nooit aan het besturingssysteem kan liggen, want Windows is toch al sinds versie 3 het toonbeeld van stabiliteit?

Momentje. Die meneer daar op de achterste rij! Die daar zo net in een Homerisch gelach uitbarstte. Gooi hem eruit! Hij heeft het gewaagd de woorden van de Here in twijfel te trekken, de Schepper van het cyberleven, de Gates Billus, die zo ruw door Pontius Reno op de Berg Capitol aan het kruis werd genageld, voor ons, codeklopers! Of, nee, nog beter: meneer Woytila, kunt U ons nog eens uitleggen hoe dat nu werkte, die Inquisitie?

The MicroSoft strikes back!