Delphi OplossingsCourant

Vol 2. No. 3. Een publicatie van het TAS Advanced Technologies Delphi OplossingsCentrum- http://www.tas-at.com/doc

Best, 25 april 2000,

Op het moment dat ik dit welkomstwoord schrijf zijn we nog slechts enkele dagen verwijderd van de Conference to the Max in Noordwijkerhout. Ik zie er met veel plezier naar uit om daar met het TAS-AT Delphi Oplossings-Centrum (DOC) te vertellen en laten zien hoe wij met Delphi en gerelateerde tools en technieken omgaan. Het laatste nieuws betreft het Inprise Enterprise Solutions Partnership van TAS Advanced Technologies, waardoor we nog beter in staat zijn om advies en projecten rondom technisch hoogwaardige oplossingen aan te kunnen bieden. Voor meer informatie verwijs ik naar het officiële pers-bericht.
Deze nieuwsbrief, de Delphi Oplossings-Courant, geeft een klein inzicht in waar onze mensen zich (soms zelfs als hobby) mee bezighouden. De Delphi OplossingsCourant verschijnt eens in de twee maanden, en dit is al weer het derde nummer van het nieuwe jaar (en daarbij het eerste *publieke* nummer). Mocht u als lezer geen (toekomstig) nummer willen missen, dan kunt u zich vrijblijvend aanmelden door een e-mailtje te sturen naar doc@tas-at.com onder vermelding van "abonnement DOC". Na verloop van tijd zal de inhoud ook op onze TAS-AT DOC website verschijnen te http://www.tas-at.com/doc

Behalve deze week op de CttM 2000 in Noordwijkerhout, kunt u ons in de nabije toekomst ook vinden op de Delphi Conferentie in London (DCon2000 - 19 t/m 20 juni 2000) en natuurlijk op de BorCon in San Diego van 8 t/m 12 juli 2000.

Inhoudsopgave

De **TAS-AT Delphi OplossingsCourant** is een produktie van het TAS Advanced Technologies Delphi OplossingsCentrum, met medewerking van Arjan Jansen, Arnim Mulder, Chris van Aart, Micha Somers, Rick Beerendonk en Ruurd Pels.

Eindredactie: Bob Swart - e-mail: doc@tas-at.com

Integratie met Windows

Auteur: **Rick Beerendonk**

Als we een applicatie schrijven die een eigen documenttype heeft, dan kunnen we het bestandsbeheer voor de gebruiker zo eenvoudig mogelijk maken door te integreren met Windows. In dit artikel zullen we zien hoe we bestandstypen bekend kunnen maken bij Windows.

De bestandentypen die Windows kent worden geregistreerd in de registry. Als we een eigen type willen toevoegen dan maken we een nieuwe key onder de key HKEY_CLASSES_ROOT met als naam *.ext* oftewel een punt en de gewenste extensie. De default waarde van deze key is een zogenaamde File Type Identifier. Deze identifier moeten we de naam geven van een andere key die we ook onder HKEY_CLASSES_ROOT gaan plaatsen. Deze laatste key zal de verbinding regelen tussen bestanden met onze extensie en onze applicatie. De naam van de identifier kun je zelf bepalen, maar het is handig wat richtlijnen aan te houden. De identifier kunnen we het beste een combinatie laten zijn van onze bedrijfsnaam, de naam van het programma en eventueel een versienummer en het datatype (voor het geval ons programma meerdere bestandstypen kent). De identifier key heeft exact dezelfde naam als de File Type Identifier die we bij de extensie hebben vermeld. De default-waarde van deze key moet een begrijpelijke omschrijving zijn van ons bestandstype. In de verkenner zal deze omschrijving ook in de Type kolom achter bestanden met onze extensie verschijnen.
Onder de identifier key maken we twee subkeys. De eerste noemen we DefaultIcon. Deze key zal aangeven waar de verkenner het pictogram voor dit bestand vandaan moet halen. Als we hiervoor hetzelfde pictogram als de applicatie willen gebruiken, dan vullen we het pad van de applicatie in met daarachter een komma en een nul. De nul geeft eigenlijk aan dat het eerste pictogram uit de applicatie moet worden gehaald. Als de applicatie meerdere pictogrammen heeft, dan kunnen we die ook gebruiken door andere getallen in te vullen.
De tweede subkey die we maken heet Shell. We kunnen hier commando's aangeven die in het popup menu van onze bestanden in de verkenner verschijnen. Het eerste commando zal bovendien de default actie zijn, de actie die dus zal worden uitgevoerd als er met de linkermuisknop op een bestand wordt geklikt.

Als default commando willen we natuurlijk dat het bestand wordt geopend. Onder de Shell key maken we daartoe een nieuwe subkey met als naam Open. De default-waarde zal de naam zijn die in het popup menu verschijnt. Als we hier &Open invullen, dan zal de O ook meteen als sneltoets dienst doen. Rest ons nog aan te geven welk programma ons bestand moet openen. Onder de Shell key maken we dan een nieuwe subkey met als naam Command. De default-waarde hiervan zal het pad van onze applicatie zijn met daarachter, gescheiden door een spatie, %1. Windows zorgt er in dit geval voor dat onze applicatie wordt aangeroepen met als argument het pad van het bestand dat geopend moet worden.

Windows weet nu voldoende om onze bestandstype te herkennen. Het enige wat ons te doen staat is te zorgen dat onze applicatie na het opstarten eventuele argumenten uitleest en als het argument een pad van een document is dit document opent.

Hieronder staat uitgeschreven hoe we een bestand met als extentie TAS kunnen toevoegen. Als extra zien we hier hoe we nieuwe, lege documenten kunnen laten maken door de verkenner. Dit wordt mogelijk door onder de key met de extensie (.tas) een subkey ShellNew aan te maken met daarin een String Value genaamd NullFile (die gelijk is aan een lege string). In het popup menu van de verkenner zal nu bij menu item Nieuw een nieuw document met de extensie tas kunnen worden aangemaakt.

```
HKEY_CLASSES_ROOT\.tas
  (Default) = "TASFile"
HKEY_CLASSES_ROOT\.tas\ShellNew
  NullFile = ""
HKEY_CLASSES_ROOT\TASFile
  (Default) = "TAS Advanced Technologies"
HKEY_CLASSES_ROOT\TASFile\DefaultIcon
  (Default) = "C:\Demo\Tas.exe,0"
HKEY_CLASSES_ROOT\TASFile\Shell\Open
  (Default) = "&Open"
HKEY_CLASSES_ROOT\TASFile\Shell\Command
  (Default) = "C:\Demo\Tas.exe %1"
```

De basis van het integreren met de Windows Shell hebben we hier gezien. Misschien dat we in de toekomst op geavanceerdere zaken kunnen ingaan.

Delphi-based Intelligent Agents

Auteur: **Chris van Aart**

Ervaringen hebben uitgewezen dat het moeilijk en duur is om grote, complexe systemen te ontwerpen, te bouwen, te onderhouden en te controleren. Neem bijvoorbeeld de problemen met verschillende versies van onderdelen van een systeem, onoverzichtelijke communicatie tussen systeemcomponenten, of veel verschillende gebruikers met daarbij eveneens verschillende behoeften.

Ontwikkelen met Intelligent Agents oftewel Agenttechnologie biedt een andere manier van kijken naar problemen. Een zelfstandige interface/informatie-agent is een stuk software dat in staat is om zelfstandig (zonder dat anderen zich ermee bemoeien) taken uit te voeren om een bepaald doel te bereiken. Voor gebruikers kunnen intelligent agents hun dienst bewijzen als digitale assistenten. Agents zijn terug te vinden in gebieden als scheduling en productieplanning, e-commerce, fraude detectie, supply chain management, effecten beheer, kennis management en diverse web-based applicaties. In deze systemen, oftewel *multi-agent systemen*, werken meerdere agents samen om zo een gezamenlijk doel te bereiken. Elke agent heeft één of meerdere specialismen. Hierdoor hoeft niet een complex systeem gebouwd te worden, maar juist kleinere systemen met een beperkte functionaliteit. Samenwerking wordt gerealiseerd door agents in een structuur te plaatsen waarin elke agent een bepaalde rol vervult. Een voorbeeld hiervan is een organisatie met een baas - een manager-agent - die werk delegeert naar een werknemers, worker-agents. Technisch gezien kunnen verschillende zelfstandige Delphi Applicaties gaan samenwerken dmv sturen van berichten om zo een werkelijk intelligent gedrag te krijgen. Dit kan door communicatie-technieken zoals bijvoorbeeld sockets (tcp/ip), (D)COM+ of CORBA, en de implementatie van (COM) interfaces kan hierbij een belangrijke rol spelen.

Een voorbeeld van een agent structuur of agent community is het zogenaamde *black-board* concept. Dit komt overeen met een groep van wetenschappers die gezamenlijk naar een probleem kijkt dat op een schoolbord (black board) staat geschreven.

Elke wetenschapper bekijkt het probleem vanuit zijn vakgebied en schrijft een deeloplossing op het bord of vult een andere aan. Naarmate de deeloplossingen groeien, vormt er een uiteindelijke oplossing. Het schoolbord wordt gebruikt als een gezamenlijke opslagplaats van probleem-beschrijvingen en (deel)oplossingen.

Binnen een multi-agent systeem wordt de kennis van één wetenschapper of expert in één agent gemodelleerd. Elke agent wordt als afzonderlijk systeem gebouwd. Met behulp van het onderling sturen van berichten wisselen de agents gegevens uit. Middleware die hierbij gebruikt kan worden kan ofwel zelf ontwikkeld worden in Delphi (gebruikmakend van MIDAS bijvoorbeeld), ofwel op CORBA gebasseerd zijn. Deze laatste keuze biedt het voordeel dat de agents makkelijker cross-language en cross-platform kunnen opereren (bijvoorbeeld ontwikkeld in Delphi en C++Builder maar ook JBuilder of straks Kylix).

Voor het adequaat inzetten van agent-technologie zijn de volgende vuistregels van toepassing:

- Moeten verschillende bestaande en nieuwe systemen met elkaar gaan werken?
- Zijn er veel verschillende kennis of informatiebronnen?
- Worden er veel verschillende technieken gebruikt?
- Zijn er veel verschillende ontwikkelaars?
- Wordt de oplossing op meerdere plaatsen tegelijk ontworpen?
- Zijn er veel verschillende gebruikers?

Ontwikkelomgevingen, zoals Delphi bieden steeds meer technieken om gedistribueerde oplossingen te bouwen, zoals met MIDAS en CORBA (VisiBroker for Delphi). Voor agent-gebaseerd ontwikkelen is distributie een rand-voorwaarde. Agentgebaseerde E-commerce, bijvoorbeeld, zal niet alleen plaats gaan vinden op basis van webpagina's maar juist via middleware die internettechnologie (zoals MIDAS met DCOM of CORBA) gebruikt als communicatiekanaal. Met behulp van agent-technologie kunnen virtuele marktplaatsen gecreëerd worden waarbij zowel de koper als verkoper vertegenwoordigd worden door respectievelijk koop-agents en verkoop-agents die als afzonderlijke gedistribueerde systemen ontwikkeld zijn.

Manipulating services with Delphi

Auteur: **Arjan Jansen**

Ever wanted to know how to manipulate services from within Delphi? This article will give a brief introduction into the functions you will need to call in order to access the services in Windows NT.

First, let me explain what a service is. A service is a program in Windows NT, which is run either at system startup or is activated by the end-user. This program provides a certain kind of functionality, e.g. the OracleServiceORCL is a service which when started will startup an Oracle DBMS. The main difference between a service and a regular program within NT is that a service can run even if no user is logged in. In the case of the OracleServiceORCL this could mean that we can startup this machine as a database server (without having to log in to the system), and then connect to this database from a remote machine in order to access the data. If you normally would like to start or stop services you go to the Settings/Control Panel/Services.

In this series of articles we will be building a small application, which provides a similar functionality to the services panel. In order to do this we need to query the status of all services on the local machine, and be able to start, stop, pause and continue these services.

This time we will be using two functions to open the service control manager database, and query the status of all services on the local machine. They are:

- OpenSCManager. This function opens a handle to the service control manager, which like the name says controls all the services on the machine.
- EnumerateServicesStatus. This function returns a list of all services handled by the service control manager, as well as their statuses.

To retrieve a handle to the service control manager we call the OpenSCManager function. The function expects three parameters, namely the machine on which the service control manager is running, the database we want to open, and the type of access we require. Since we want to open the service control managers default database on the local machine the first parameter can be an empty string and the second parameter can be a nil pointer.

We want a list of services on the local machine, so SC_MANAGER_ENUMERATE_SERVICE is the access type we need:

```
SCMHandle := OpenSCManager('', nil,
  SC_MANAGER_ENUMERATE_SERVICE);
```

The resulting handle is stored in the variable SCMHandle, which is of type SC_HANDLE. We can use this handle to the service control manager (SCM) to get a list of all services on the local machine. The function we need to call is EnumerateServicesStatus, which will give us a list of all services handled by the SCM. The function expects the following arguments:

- SC_HANDLE hSCManager, this is a handle to the SCM, which we opened by calling the OpenSCManager function.
- DWORD dwServiceType, this parameter specifies the type of services we want to enumerate. Since we do not want to enumerate drivers at this point, we will pass SERVICE_WIN32 to this parameter. If we wanted to enumerate all drivers, we should have passed SERVICE_DRIVER to this parameter.
- DWORD dwServiceState, this parameter specifies the state of the services we want to enumerate. It can be SERVICE_ACTIVE, SERVICE_INACTIVE or both. Since we want to enumerate all services on the local machine we will pass SERVICE_ACTIVE OR SERVICE_INACTIVE to this parameter.
- LPENUM_SERVICE_STATUS lpServices, the lpServices must be a pointer to an array of ENUM_SERVICE_STATUS records. After calling the EnumerateServicesStatus this array contains a list of records containing the internal name of the service, the display name of the service, and the status of the service. We will need to allocate this buffer before calling the EnumerateServicesStatus function.
- DWORD cbBufSize, this parameter should be set to the size of buffer as specified in the previous parameter.
- LPDWORD pcbBytesNeeded, if the buffer that is passed to store the services is not large enough, this parameter is set to the additional number of bytes required to retrieve all services after calling EnumerateServicesStatus.
- LPDWORD lpServicesReturned, this parameter is set to the number of services that are returned in the services buffer after calling the function.

- LPDWORD lpResumeHandle, if not all services could be returned due to the fact that the buffer is too small, then this parameter will be set to the handle of the next service we want to retrieve, and the function result will be false. lpResumeHandle can be used in subsequent calls to EnumerateServicesStatus to get the status of the rest of the services.

The result of the function call is a boolean, true if the function completes successfully, and false if either the buffer we allocated was too small, or something else went wrong. In the latter case, we need to call the getLastError function for more info. If the getLastError returns an ERROR_MORE_DATA we need to make subsequent calls to EnumerateServicesStatus. Otherwise an unexpected error occurred.

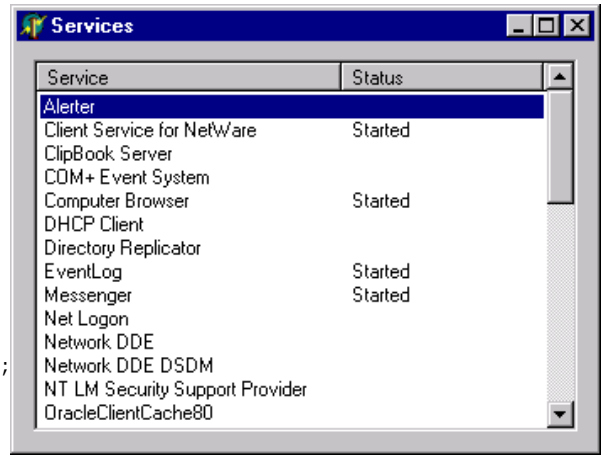
Now we will focus on the GUI we need to build to display these services. We start by making a form, called TfrmServices. On this form drop a Tlistview called lvServices. This listview should get two columns, namely Service and Status. Make sure that the SortType of the listview is set to stText. This should be enough for now.

Add a private procedure CheckServiceStatus to this form and call it in the OnCreate of the form. The CheckServiceStatus procedure will call the EnumerateServicesStatus, and will store all services in the lvServices listview of the form. We make a buffer of at most 50 services at a time, and pass this to EnumerateServicesStatus. After the call we need to check the result of the function to see if there are more services to be retrieved through subsequent calls. All data that is received in the buffer, is then parsed and stored in the lvServices listviews items. The display name of the service will be stored in the caption part of the listitem, the internal name of the service in the data part. We need to convert the service status code to a readable string, which is done in the function StateToString. This is continued until all data is received or an unexpected error occurred. The code for CheckServiceStatus and StateToString is as follows:

```
function StateToString(const CurrentState: DWORD): String;
begin
  case CurrentState of
    SERVICE_RUNNING         : result := 'Started';
    SERVICE_STOPPED         : result := '';
    SERVICE_PAUSED          : result := 'Paused';
    SERVICE_STOP_PENDING    : result := 'Stop Pending';
    SERVICE_START_PENDING   : result := 'Start   Pending';
    SERVICE_PAUSE_PENDING   : result := 'Pause Pending';
    SERVICE_CONTINUE_PENDING: result := 'Continue Pending';
  else
    result := '';
  end;
end;

procedure TfrmServices.CheckServiceStatus;
const
  MAXSERVICES = 49;
  MachineName = '';
type
  TSList = array[0..MAXSERVICES] of TEnumServiceStatus;
  PSList = ^TSList;
var
  allOk                                : Boolean;
  serviceList                          : PSList;
  bNeed, nServ, resHandle, StartupType : Cardinal;
  SCMHandle, I, lastError              : integer;
  ListItem                             : TListItem;
begin
  SCMHandle := OpenSCManager(MachineName, nil,
    SC_MANAGER_ENUMERATE_SERVICE);
  if SCMHandle = 0 then ShowError(GetLastError)
  else
  try
    resHandle := 0;
    new(serviceList);
    lvServices.Items.Clear;
    try
      repeat
        allOk := EnumServicesStatus(SCMHandle,
                   SERVICE_WIN32,
                   SERVICE_ACTIVE or SERVICE_INACTIVE,
                   serviceList^[0],
                   sizeof(serviceList^),
                   bNeed, nServ, resHandle);
        lastError := GetLastError;
        if allOk or (lastError = ERROR_MORE_DATA)
        then
        begin
          for i := 0 to nServ-1 do
          begin
            ListItem := lvServices.Items.Add;
            ListItem.Caption :=
              StrPas(serviceList^[i].lpDisplayName);
            ListItem.Data :=
              StrNew(serviceList^[i].lpServiceName);
              ListItem.SubItems.Add(StateToString(
               serviceList^[i].ServiceStatus.
                 dwCurrentState));
          end;
        end;
      until allOk or (lastError <> ERROR_MORE_DATA);
      if lvServices.Items.Count <> 0 then
        lvServices.Selected := lvServices.Items[0];
    finally
      Dispose(serviceList);
    end;
  finally
    CloseServiceHandle(SCMHandle);
  end;
end;
```

The figure below shows the screen output:



Next time we will add the startuptype of the service to the listview, as well as buttons to start, stop, pause and continue the services.

Component Based Development

Auteur: **Micha Somers**

Flexibility, a shorter time-to-market, less development costs, increasing productivity, higher quality, and maintainability are often mentioned as arguments for introducing Component Based Development (CBD) in an organization.

In reality however, CBD does not automatically lead to these advantages. CBD asks for investment in order to organize things carefully. CBD can be setup in several ways where the support of automated systems may vary. However, knowledge about the organization, culture, and the size of the organization, are important for finding the right procedures and tools to use CBD.

What is CBD?

CBD can be described as:

An approach for system development where systems or parts of systems are developed by assembling, configuring and integrating reusable software components.

In this context a component means:

A reusable software unit that provides particular functionality, can be deployed independently, and can easily be integrated with other components and applications.

Furthermore, components

- are big enough to provide useful functionality,
- are small enough to use and maintain them easily,
- have some characteristics (variables or properties),
- show some behavior (methods)
- use explicit specifications

Often Lego building blocks are used for illustrating how CBD works. It shows how small and simple building blocks are used for building a larger functional system.

Why CBD?

There are a number of reasons why CBD is interesting. One of the most important reasons is reusability; we do not want to invent the same thing again and again.

Quality, reliability, and maintainability become more and more important as the complexity of applications increases.

Modularization and reuse of components can help with this.

In this dynamic world where the market and the number of competitors grow, organizations have to become more flexible. They have to take care of:

- shorter lifecycle of products
- shorter time-to-market
- flexible products
- changes in business processes

In order to handle this, software systems that support the business processes need to become more flexible. This is where reusability and a modular approach are essential.

CBD provides all the advantages of reusability in general, such as shorter time-to-market, lower development costs, increasing productivity, consistency, etcetera.

Besides that, CBD can also lead to:

- Flexibility: components can easily be replaced or extended.
- Splitting up complexity in clear organized pieces.
- Better quality and maintainability.

New technical possibilities and the increasing use of the Internet will make CBD even more important. For example e-business and e-commerce solutions ask for a flexible and efficient approach.

By using component middleware (such as CORBA, DCOM, COM+), thin-clients, and server-side components it becomes possible to efficiently implement e-business and e-commerce solutions.

C++ lost+found #13

Auteur: **Ruurd Pels**

Now for something completely different. Normally I do C++(Builder) stuff, this time I'll do Delphi stuff and take it from there to C++ stuff permitting you do not have C++Builder Enterprise but still want to use the idiom. What idiom? The **CORBA bind in a property** idiom.

Use CORBA wizard

C++Builder Enterprise 4 comes with Visibroker. It has a wizard that let's you use a CORBA object in a form. It uses a property that you use to get at the methods exposed by a CORBA server object. To find it, right click in the toolbar and turn on the CORBA toolbar. The icons with the blue bubble says 'Use CORBA Object'. If you use that wizard, it will lead you through a number of options. What you may end up with is a property in your form. From the following IDL

```
module Conference
{
  interface Session
  {
    attribute string Name;
  };
};
```

it can make this:

```
class TFormMain: public TForm
{
private:
  Conference::Session_ptr __fastcall GetSession();
  Conference::Session_var FSession;
  void __fastcall
    SetSession(Conference::Session_ptr _ptr);
public:
  __property Conference::Session_ptr Session =
  {
    read=GetSession,
    write=SetSession
  };
};
```

The implementation in C++ is as follows:

```
Conference::Session_ptr __fastcall
  TFormMain::GetSession(void)
{
  if (FSession == NULL)
  {
     FSession =
       Conference::Session::_bind("ConferenceSession");
  }
  return FSession;
}

void __fastcall
  TFormMain::SetSession(Conference::Session_ptr _ptr)
{
  FSession = _ptr;
}
```

If you use the property, it will automagically bind to the server object for you. Also, since it uses a Conference::Session_var as a variable, the object will be released automatically.

Enter Visibroker for Delphi

Inprise recently released a tool for Delphi with which you can create CORBA clients. However, no wizard is included in this package. No such luck… yet. Maybe in the next issue of Delphi. It is more or less a command line tool that generates the client stubs in Delphi. However, you might use the same method to bind to server objects in your own forms:

```
type
  TConfClntForm = class(TForm)
    EditConfName: TEdit;
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
    FSession: Conference_i.Session;
    function GetSession: Conference_i.Session;
    procedure SetSession(iface: Conference_i.Session);
  public
    { Public declarations }
    property Session: Conference_i.Session
      read GetSession
      write SetSession;
  end;

var
  ConfClntForm: TConfClntForm;

implementation

{$R *.DFM}
function TConfClntForm.GetSession: Conference_i.Session;
begin
  if (FSession = nil) then
    FSession :=
      TSessionHelper.Bind('ConferenceSession');
  Result := FSession;
end;

procedure TConfClntForm.SetSession(iface: Conference_i.Session);
begin
  FSession := iface;
end;
```

If we want to use it:

```
procedure TConfClntForm.FormActivate(Sender: TObject);
begin
  EditConfName.Text := Session.Name;
end;
```

'Back to C++, straight this time

If you don't want to use C++Builder like C++, you also can wrap client calls to CORBA objects in the following way. It sort of emulates the __property extension:

```
class Wrapper
{
public:
  Conference::Session_ptr Session()
  {
    if (sv == 0)
    {
      sv =

Conference::Session::_bind("ConferenceSession");
           }
         return sv;
  }
  void Session(Conference::Session_ptr ptr)
  {
    sv = ptr;
  }
private:
  Conference::Session_var sv;
};
```

It works for me! See you next time...

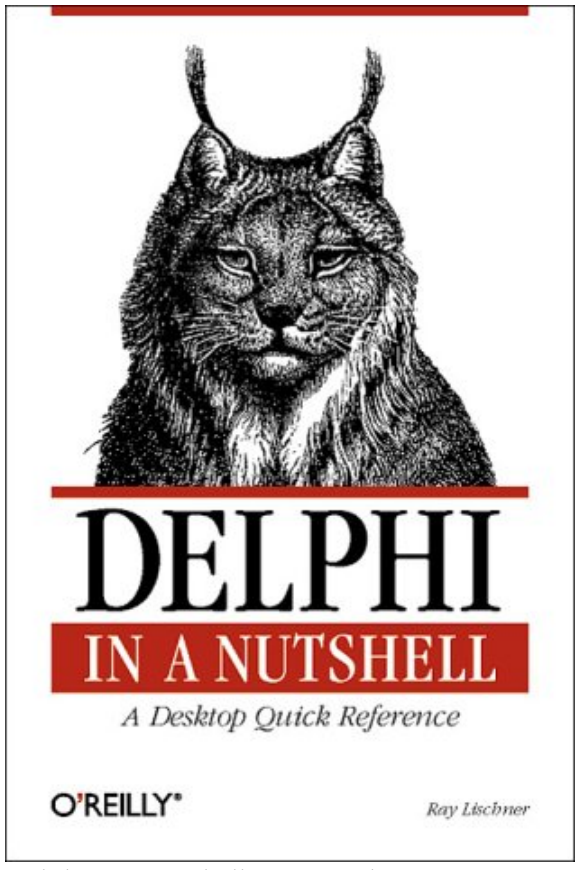
Review: Delphi in a Nutshell

Auteur: **Arnim Mulder**

Ik weet nog toen ik de overstap maakte van Pascal naar Delphi dat ik op zoek was naar goede Delphi boeken. Elk boek bevat wel een of twee interessante hoofdstukken, maar het is Ray Lischner gelukt om een heel groot deel van de taal te documenteren. Zo heeft dit boek alle keywords, functies, procedures en compiler directives gedocumenteerd. Alle hoofdstukken zijn interessant voor iedere programmeur, van welk niveau ook. Het gaat immers over onderwerpen waar je dagelijks mee te maken hebt. Het boek begint met een korte beschrijving van de structuur van een Delphi project. Daarna komt het Delphi Object Model aan bod. In dit hoofdstuk wordt aandacht besteed aan classes, objecten en interfaces (COM). Dan is er nog een hoofdstuk over RunTime Type Information en een over Threads. Maar het grootste deel van het boek beslaat de language reference (dus geen VCL reference). De reference is niet zomaar een kopie van de online-help van Delphi, maar een zeer gedetaileerde beschrijving van alle keywords, functies en procedures. In veel gevallen is er een voorbeeld opgenomen om het geheel te verduidelijken. Tips&tricks is een vast onderdeel. In deze sectie worden veel nuttige zaken over het onderwerp behandeld. Wanneer moet je nu een bepaalde functie of procedure wel gebruiken, wat voor foutcodes kun je verwachten als de aanroep niet lukt en waar moet je op letten als je er gebruik van maakt? In de sectie Tips&Tricks worden deze vragen beantwoord. Naast deze onderwerpen zijn ook de runtime errors, variant-typen en operators gedocumenteerd. Omdat de DOS-prompt nog steeds aanwezig is in Windows is ook een hoofdstuk gewijd aan de command-line programma's. Naast de uitleg van de programma's zelf worden ook alle parameters, opties en switches beschreven. De commandline tools die in dit boek aan bod komen zijn de Delphi compiler (dcc32.exe), resource compiler (brcc32.exe), DFM converter (convert.exe), Object File Dumper (tdump.exe) en de IDE (delphi32.exe). Het laatste hoofdstuk bevat een complete documentatie van de SysUtils unit. Deze unit wordt in bijna ieder Delphi-project wel gebruikt en is een nuttige toevoeging aan de language reference. Het hoofdstuk is onder-verdeeld in de onderwerpen Exceptions & Errors, File Management, String Management, Numeric Conversion, Dates & Times, Localization, Modules en Windows. Voor ieder onderwerp worden alle procedures, functies en types opgesomd en uitgelegd.

Conclusie

De inhoudelijke kwaliteit van het boek is zonder meer prima, de kwaliteit van de buitenkant wat minder. Het boek is ingebonden met een lijmlaag. De lijmlaag is niet van de beste kwaliteit waardoor de kaft al snel loslaat van het boek. De bladen blijven wel vastzitten. Desondanks is dit boek een "must-have" voor programmeurs. Juist omdat dit boek gaat over onderwerpen waar je dagelijks mee te maken hebt en sommige onderwerpen onvoldoende zijn gedocumenteerd is dit een goed naslag-werk. De uitgebreide beschrijvingen zijn helder en de voorbeelden ondersteunen de begrips-vorming. Een echte aanrader dus! Voor de prijs hoef je het in ieder geval niet te laten: US$ 24,95 (minder dan 60 gulden).



Delphi in a Nutshell - Ray Lischner
Uitgever:: O'Reilly
ISBN: 1-56592-659-5
561 bladzijden, paperback, Engels, geen CD

Delphi 5 Web Techniques

By: **Bob Swart** (aka Dr.Bob)

This article gives an overview of Web Development Techniques found in Delphi 5 Professional and Enterprise.

Throughout this article, I'll use the same example to illustrate the end-result of a specific Delphi internet technique. The example is based on two tables: CUSTOMER.DB and ORDERS.DB in a master-detail relationship, found in the DBDEMOS alias which is installed with your copy of Delphi (or C++Builder), so anyone with a copy of Delphi 5 should be able to replicate the results shown in this article. All three examples use the same (remote) datamodule below.

Client-Side: ActiveForms

Delphi 5 contains support for ActiveX controls and ActiveForms. While ActiveX controls are very popular in some development environments, you need a certain amount of functionality before it makes sense in Delphi to wrap it in an ActiveX control instead of a native VCL component. Apart from that, there are a number of VCL classes that cannot be used as parent base class to derive an ActiveX control from. Not without a good reason, since data-aware controls, for example, obviously need to connect to a TDataSource component, and not to a TDataSourceActiveX component. As a consequence, the One-Step-ActiveX features of Delphi are a bit less useful when it comes to client-side internet development.

Instead, internet ActiveX clients should be developed as Delphi ActiveForms! An ActiveForm is just like a "normal" ActiveX control. The only difference is that it looks like a regular TForm, and that we can place just about any control on the ActiveForm, thereby building an entire application in one ActiveX control - an ActiveForm. There are a few restrictions: we have no main (pull-down) menu, and there's no caption (but you can fake a caption with a panel on top). Other than that, I always try to avoid using other ActiveX controls on an ActiveForm. As you'll see from the screenshot below, an ActiveForm - even at design-time - gives you most flexibility when it comes to designing the internet GUI of your application:

Advantages of ActiveForms: we can replicate an entire Windows GUI inside an ActiveForm, which looks just great and works excellent. Besides, older Client/Server systems can be ported to ActiveForm

Disadvantages of ActiveForms: ActiveForms only operate inside Internet Explorer (or Netscape with the right plug-in). Since they are in effect Win32 applications running on your machine, you may experience potential security problems (unlike Java applets, there's no sandbox for ActiveForms). Codesigning is one way to verify the author of an ActiveForm, but they are still best used in an intranet environment.

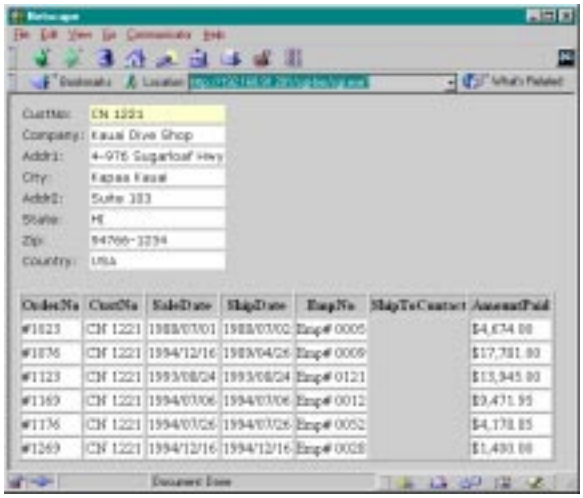
Another disadvantage is the fact that the client is pretty fat, needing a database engine as well (note: this can be solved using Delphi 5 Enterprise multi-tier techniques like MIDAS). We can keep the filesize down a bit by using run-time packages, but we still need to deploy these packages as well.

Recommendation: ActiveForms are fine in an intranet environment, where you can control the browser people use, and the users in their turn can trust the ActiveX control itself. Also, an intranet means that you can share the same database server on a LAN, which might be harder (or next to impossible) on the internet.

Server-Side: WebBroker

Delphi 5 supports a number of server side solutions. Traditionally, these consist of two main approaches: CGI or ISAPI. CGI stands for Common Gateway Interface, which stands for the way in which the client (browser) sends a request through the web server to the web server application, and receives the dynamic result back. A CGI script, or executable with Delphi, is loaded, executed and unloaded from memory again. While this is convenient for small CGI apps, it can be a burden for larger systems, especially if a connection to a database has to be made over and over again (and loaded and unloaded over and over again). For this reason, both Microsoft and Netscape "invented" a solution called resp. ISAPI and NSAPI. An xSAPI solution no longer consists of an executable, but of a DLL instead. And this DLL is only loaded once - in the memory space of the web server.

The benefits are clear: ISAPI DLLs only have to be loaded once, so there's no more speed penalty. The major downside is the difficulty developers may have in updating ISAPI DLLs (since they are running, you have to unload them manually - usually by bringing down the web server - before you can replace them with another version). Also, since ISAPI DLLs run inside the memory space of the web server (like IIS), a fatal crash can even bring down the web server (calling application) itself. Ideally, one would like to start with a CGI application - which can be replaced often, and while not fully debugged, will not be able to bring down the entire web server. At a certain point in time, one would then like (or need) to switch over to the ISAPI approach.
Unfortunately, the "API" part of ISAPI or NSAPI is not directly compatible with a simple CGI executable, and conversions from one to another can easily introduce new bugs and potential problems. That's where Delphi WebBroker Technology comes in handy: apart from being able to develop CGI and ISAPI applications "the old way", Delphi also offers the WebBroker Technology, which embeds the CGI, WinCGI and ISAPI/NSAPI solutions in a single "Web Module". It shields the developer completely from the differences, and focuses on the process of receiving requests and producing responses instead. For the latter, a number of "HTML-producing" components are available for template based record and entire table producing. And almost everything can be configured or modified by using properties and event handlers. In fact, for the example in the screenshot below, I didn't need any HTML knowledge:



Apart from WebBroker, Delphi 5 also includes support for ASP - Active Server Pages and Active Server Objects. This consists of a special ASP Object Wizard and generation of a template. The result as seen in a browser can be quite similar to what we can obtain using the WebBroker.

Advantages of WebBroker: they produce thin-client applications, no distribution problems. CGI vs. ISAPI discussion regarding robustness vs. speed. Templates can be saved as external files, so it's easy to maintain and configure the "WebBroker results" without having to recompile the application itself. Great for clients who know HTML but don't (want to) know Delphi.

Disadvantages of WebBroker: for GUI we're limited to the CGI Form controls, meaning edit, button, checkbox, radiobutton, listbox / combobox, memo field and general HTML formatting features.

Recommendation: for an internet solution, WebBroker is suited perfectly for presenting data in a fixed format. Additional HTML formatting can be applied to enhance the user interface, but it remains limited.
WebBroker components can also be applied to produce HTML in non-web applications.

N-Tier: MIDAS 3

We saw that ActiveForms tend to be really fat: filesize, but the required DLLs and drivers as well - specifically database drivers. And since we can't expect our internet clients to have the BDE drivers on their local machine, it's time to take a look at MIDAS; the middle-ware layer for Delphi and C++Builder (and JBuilder, with JMIDAS).
Using MIDAS, we can split an application in two parts: a client (GUI) part and a middle-ware "application server" part, which in its turn is often directly connected to the DBMS (like Oracle, InterBase). In this architecture, the client has no database related code at all, and connects to the application server using a Client DataSet and a Connection component. The later comes in a number of flavors: CORBA, DCOM, sockets and a new one with Delphi 5: HTTP, which enables you to put the application server behind a proxy or firewall.

Visually, nothing changes, as the DataSets on the fat client are replaced by ClientDataSets and connections to the application server. Data Modules are turned into Remote Data Modules to operate on the server side. Constraints and parameters are passed from the client to the server and back. And when disconnecting the server, we can locally save the data from the client(dataset), allowing the user to work "locally" in the so-called brief-case model (synchronising later when re-connecting to the application server).

Advantage of MIDAS: separate client from application server and database server. Produce thin-clients, communicate using CORBA, DCOM, sockets or http (through a proxy/firewall). The briefcase model is especially convenient for users "on the road".

Disadvantage of MIDAS: there is a run-time license fee. The good news: recently, the price was slashed to US$ 299.95 per MIDAS server

Recommendation: if you don't know on beforehand how many clients/hits your internet application might get, you can start by using regular datasets. The MIDAS technology can be "plugged-in" later, as long as you start using data modules from the start (so you can easily migrate these to remote data modules).

InternetExpress

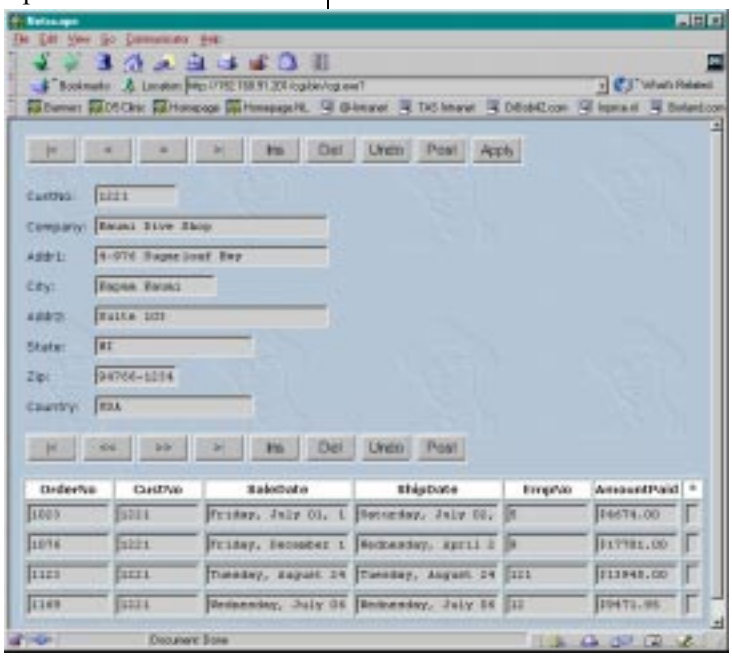
Right now, we've covered almost every internet feature of Delphi. Except for the latest and greatest, called InternetExpress.

InternetExpress combines the power of WebBroker with the scalability of MIDAS to produce truly amazing output.
In fact, WebBroker is extended with a special MidasPageProducer, which makes use of special web components that use JavaScript code to bind XML data to them. Apart from sounding impressive, it really looks impressive, especially when you see an entire dataset on one page, and only the data scrolls when you click on the navigator buttons (that's right, the page itself remains the same - no flicker - and only the data inside the edit controls changes value).

Advantage of InternetExpress: Extends WebBroker CGI and ISAPI solutions with multi-tier MIDAS support and XML/JavaScript for dynamic HTML webpages.

Disadvantage of InternetExpress: new technology, relative few publications/examples, only available in Delphi 5 Enterprise edition. Apart from that, it's currently not easy to send data from the application server to the client in "batches", so it's less useful - for now - for really large datasets. Dr.Bob has prepared a "quick fix" for this problem, which was published in The Delphi Magazine.

Recommendation: if you can afford it, go for InternetExpress as web server, using MIDAS as middle-tier layer for your database application server. This technique offers you the most flexible and scaleable approach for web server side development. Personally, I love it.



Delphi OplossingsCentrum

Sinds 1 januari 2000 is het TAS-AT Delphi OplossingsCentrum (DOC) aktief. Dit orgaan, onderdeel van TAS Advanced Technologies, is beschikbaar voor de gehele TAS Groep en haar klanten. De missie van het Delphi OplossingsCentrum is het aanbieden van kwalitatief hoogstaande oplossingen met Delphi. Daarnaast wil het DOC omgaan met nieuwe ontwikkelingen aangaande Delphi of daaraan gerelateerde zaken; we zijn front-runner op Delphi gebied binnen Nederland (en Europa).
De eerste ervaringen uit de markt - zowel bij bestaande als nieuwe klanten - zijn uitermate bemoedigend. Bij Delphi projecten, bouw of consultancy staan onze mensen niet langer "alleen"; een lid van het DOC kan altijd terugvallen op de kennis en ervaring van het DOC zelf. Onze klanten ervaren dit als zeer positief, omdat daarmee de kwaliteit van de service op het gebied van Delphi toeneemt. Daarnaast geeft het **Inprise Enterprise Solutions Partnership** van TAS Advanced Technologies het Delphi OplossingsCentrum ook naar buiten toe meer glans.
De doelstelling van het Delphi OplossingsCentrum wordt ondersteund met een aantal middelen:

Case Study/White Papers

De consultants van het DOC voeren regelmatig case studies uit en schrijven white papers, waarmee onze leidende positie op Delphi gebied bevestigd en gehandhaafd blijft. Artikelen worden zowel in tijdschriften als op websites en tijdens (nationale en internationale) conferenties gepresenteerd.

Consultancy

Bij het aangaan van nieuwe Delphi projecten of het begeleiden van klanten die Delphi trajecten overwegen zullen de consultants van het DOC meedenken over de architectuur en het ontwerp.

Projecten

Het DOC probeert aktief betrokken te zijn bij ieder Delphi project dat door TAS wordt uitgevoerd - al is het maar op informatieve basis. Door Delphi ervaring te delen met de hele TAS Groep kunnen we de hoge(re) kwaliteit waarborgen.

Trainingen

Maandelijks organiseert Bob Swart een gevorderde Delphi 5 Clinic bij TAS Advanced Technologies in Best. Daarnaast kunnen zgn. custom clinics samengesteld worden (voor tenminste vier personen van één klant, rondom een zelf te bepalen onderwerp).

Richtlijnen en Audits

Door het opstellen van Code Richtlijnen en het houden van Audits wordt de kwaliteit van de Delphi projecten gewaarborgd. Begeleiding en betrokkenheid bij Delphi projecten is hierbij noodzakelijk.

Herbruikbare Componenten

Het DOC zal de geproduceerde componenten in Delphi testen op hergebruik, archiveren, onderhouden, documenteren en beschikbaar maken voor nieuwe Delphi projecten.

Boeken, Tijdschriften, CD-ROMs

Bijna honderd boeken en vele tijdschriften en CD-ROMs over Delphi zijn beschikbaar in de TAS-AT DOC bibliotheek van Bob Swart in Best.

Maandelijkse Bijeenkomst bij TAS-AT in Best

Iedere maand is er een - interne - DOC bijeenkomst, waarbij één of meerdere technieken van of rondom Delphi diepgaand besproken worden. Voorzitter van de bijeenkomst is Bob Swart, secretaris is Micha Somers. Dit is dè manier om kennis uit te wisselen onder de aktieve DOC leden.

TAS Advanced Technologies en Delphi

Het moge duidelijk zijn: Delphi leeft, en het TAS-AT Delphi OplossingsCentrum timmert flink aan de weg. Bent u geïnteresseerd wat het TAS-AT Delphi OplossingsCentrum voor u kan betekenen, dan nodigen wij u bij deze uit om een afspraak te maken, zie http://www.tas-at.com/doc of stuur een mailtje naar **doc@tas-at.com** voor meer informatie.