Delphi OplossingsCourant

Vol 2. No. 4. Een publicatie van het TAS Advanced Technologies Delphi OplossingsCentrum- http://www.tas-at.com/doc

Best, 4 september 2000,

Op het moment dat ik dit welkomstwoord schrijf is de vakantieperiode afgelopen en zijn we allemaal weer met frisse moed begonnen aan het nieuwe seizoen. Komend halfjaar staat natuurlijk in het teken van Kylix (Delphi voor Linux) en daarna Delphi 6.

Als Inprise Enterprise Solutions Partner doet TAS Advanced Technologies mee met het Linux Development Seminar dat Inprise Northern Europe organiseert op 20 september in Brussel en op 21 september in Den Haag. TAS-AT DOC oprichters Micha Somers en Bob Swart hebben toestemming gekregen om op deze twee dagen Kylix te presenteren aan het publiek. Zodra Kylix officieel beschikbaar is zal dit opgevolgd worden met verschillende Kylix Clinics (zie http://www.tas-at.com/doc voor details).

Ook deze Delphi OplossingsCourant, geeft enigszins weer waar onze mensen zich (soms zelfs als hobby) mee bezighouden. De Delphi OplossingsCourant verschijnt eens in de twee maanden (behalve in de vakantieperiode), en dit is al weer het vierde nummer van het nieuwe jaar. 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".

Behalve op het Linux Seminar in Brussel en Den Haag, kunt u ons in de nabije toekomst ook vinden op de Inprise Conferenties in London en Frankfurt.

[illegible]

Inhoudsopgave

De **TAS-AT Delphi OplossingsCourant** is een productie van het TAS Advanced Technologies Delphi OplossingsCentrum, met medewerking van Arjan Jansen, Arnim Mulder en Micha Somers.

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

TAS Advanced Technologies

Delphi Oplossings Centrum

Een Schot in de Roos!

http://www.tas-at.com/doc doc@tas-at.com

Inprise Enterprise Solutions Partner

Delphi 5 Frames

Auteur: **Bob Swart**

One of the many new features of Delphi 5 is called Frames. At first sight, it bears resembles to both Form Inheritance and Component Templates. However, according to my own experience, it has some (not all) of the good features of these two pre-existing features, but fewer of the problems. As such, it's a good reason to take a closer look at Delphi 5 Frames and see what they can mean for you - and what to watch our for when you actually start to use them...

Form Inheritance

Delphi Form Inheritance was introduced a few versions of Delphi ago. It offers the feature of having one or more custom parent Form classes, from which instances can be derived. It's a great way of enforcing a common base look-and-feel (such as forms with a certain background, toolbar, menu or any other "common" part).

The problem that I had with early versions of Form Inheritance is that I found it a bit too fragile. If - for whatever reason - the parent Form definition had become corrupt (or missing), then you could no longer open any of the derived Forms. This means you could no longer edit them either. And this once happened right after I made some changes to a base parent Form, so I did not have a recent (useful) backup I could restore. While I'm sure I was just unlucky that day, I must admit that I've seldom used Form Inheritance again.

Component Templates

Delphi 4 introduced Component Templates. The principle is simple: select one or more components, and create a component template out of them. This includes new property values but event handlers as well. The only limitation are Form event handlers (a component template can contain everything except for the form itself). The end-result is another component (template) which appears on a page of the component palette as well.

A disadvantage of component templates is the fact that they are hard to share. Of course, once installed on my machine, I can use them in all my projects (on that machine), but when I move a project over to another machine, then I get an error message stating that the component (template) cannot be found. Component templates are stored in the Delphi.dct file (Borland Palette Template Library), and although you can copy this file from one machine to another, it's hard to copy a single component template from one machine to another. In that case, I have to recreate the component template on each machine where I want to work on a given project, and this isn't easy when working on a project with more than one developer.

Frames

A frame is a true piece of a jigsaw puzzle, and yet an entity in itself. Frames can be seen as component templates, and as such stored on the component palette. However, a frame is also the definition (like Form Inheritance) which is shared among all instances of the frame in any place we've used the frame. And you're not limited to one frame per form, but you can see the form as a big puzzle and the frames as pieces of the puzzle. The best news is that you can use the same frames to work on a regular form as well as an ActiveForm. This is a great help when writing regular applications and ActiveForm-based applications. Apart from that, frames can also be used on other frames, to construct mega-frames.

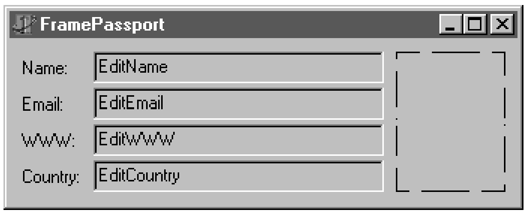
There are however a few rules and restrictions to creating and using frames, so let's fire up Delphi 5 and experiment with creating and using a single frame now.

Delphi 5

To create a new frame select the File | New Frame menu option. This will create a new frame which looks very much like new form. One of the visual differences is the fact that a form usually shows a grid of points (used as help when building your user interface), whereas a frame does not. My guess is that this difference is mainly to indeed point out that you're looking at a frame and not at a form. Other than that, I can think of no good reason why a frame wouldn't need a grid of points as well (it is quite helpful).

Creating Frames

For our example frame, close the project (if one is open), and select File | New Frame. Now, add four labels, four edits and one image component to this frame. The purpose of this example frame would be to create some sort of user passport frame (with name, e-mail address, website and country, together with a photo of the person itself). Due to the lack of grid points it's a bit harder compared to regular forms to design the frame, but not that much. My own version is designed as follows:



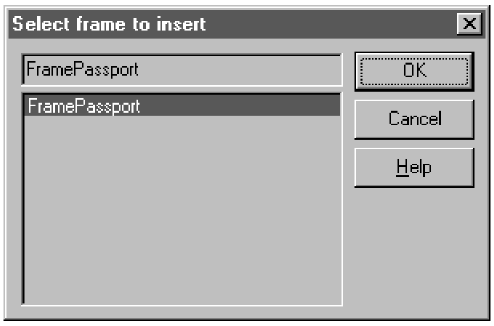
This frame is the bare-bones information that I'd like to use in several applications. At some places, I'd like to have only these four fields, at other places I need some additional fields as well. Note that I've given the frame a sensible name: FramePassport. Also note that the four edits have specific names as well: EditName, EditEmail, EditWWW and EditCountry. And the image component is named ImagePhoto. It is important to give each control that may be accessed from outside the frame a sensible name, as we'll see in a minute.

Using Frames

I've saved the frame we've just created in a file called FPassport.pas (with associated FPassport.dfm of course). In order to use a frame, we must add it to a project. First, start a new project, this will close the frame source files. In order to add a frame to a form, we must drop a "frames" component from the Standard tab of the Component Palette. However, as long as no frame source files are actually part of the project, we cannot drop the frames component (you'll get an error message telling you that you first need to create some new frames).

In order to add a frame to a project - so you can use it - just open the Project Manager, right-click on the project target, and select "Add" to add the frame source file to the project.

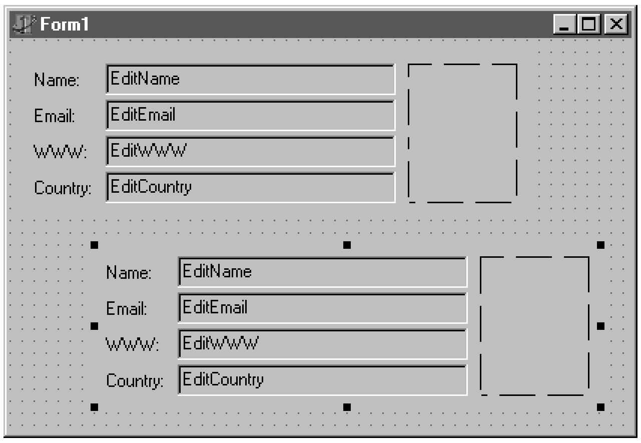
Once a frame source file is part of the project, you can drop a "frames" component on the form, which will result in the "frame selection" dialog. In our example, it only lists a single frame, but it's not hard to imagine a longer list of available frames, which means it's important to give each frame a unique name. This will be one of our most important guidelines.



The editbox on top of the listbox can be used as speed-search, which is helpful when you have a really long list of frames.

Once you've selected a frame from the list, it will be the actual frame "displayed" by the frames component you just dropped on the form. Or, to be precise, the frames component is actually "turned" into an instance of the FramePassport in this case. The normal Delphi naming convention is used, so the first one that we drop will be called FramePassport1, and is of type TFramePassport. A second frames component, also used to show a FramePassport will be named FramePassport2, and so on.

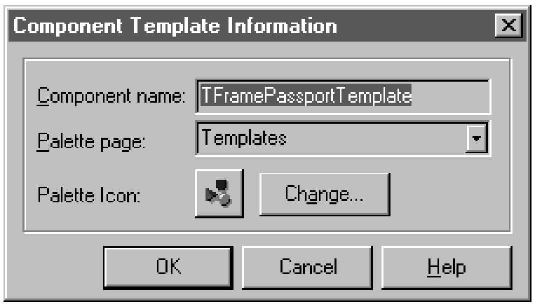
Once a frame is dropped on a form (or any other location for that matter) we can still see the frame component itself - unlike a component template which is just expanded into a bunch of components, but doesn't result in an actual "template" itself. As a direct consequence, we can select the frame and drag it around, taking everything from the frame with it. So frames are pretty easy to align or place next to each other - unlike component templates.



Another difference between a component template and frame is the fact that the former results in the actual components (that make up the template) to be dropped on the form. Renaming the components along the way (adding numbers to them), which might impact the event handers as well. Dropping a frame on a form means you get an instance of the frame itself, but all components that are placed on the frame will still be sub-components of the frame. Keeping their original name, so every EditName will be called EditName. The first one is FramePassport1.EditName, the second one FramePassport2.EditName. In that sense, the frame is a super-component, and can be seen as "portal" to the sub-components.
Of course, we can still individually manipulate the sub-components on each frame. For example, we can click on the EditName of the first frame and set its background color to clWhite (in the frame, all edits used a gray background color, as you may have noticed). In order to undo all "custom" changes, we only have to right-click on the sub-component and select the "revert to inherited" option - much like the same feature used in Form Inheritance.

Using Frames (2)

Apart from adding frames to a project by hand, it's often much easier to just open the frame file, right-click with the mouse on the frame, and select the "Add to Palette" pop-up menu option. This will show the new component template dialog, which already suggests the name TFramePassportTemplate for us. You may wish to change the palette page (from the default Templates page to DrBob42 for example) as well as the palette icon, before you click on OK and add the frame as component template to the component palette:


Component Template Information

Component name: TFramePassportTemplate
Palette page: Templates
Palette Icon:

Once the frame is installed as component template, we can use it in each and every project on that machine, without having to add the frame source file to the project by hand. We still need the frame source file, since the component palette refers to it (you can easily test this by renaming or removing the frame source files right after you've turned it into a component template - you won't be able to compile your project again). Once you turned a frame into a component template, you no longer need to drop a "frames" component to put one on the form, but you can just select the right component template directly.
Compared to a "regular" component template, the frame has as benefit that it includes the container, i.e. the frame itself is turned into a component template, including all sub-components. Which results in stronger coherence of the component template when based upon a frame.

Frames Benefits

One of the most useful features of frames is the fact that they can be used many times in your projects. And while they are used in many different places, you can still change the original frame and - all at the same time - change them at all places where they are used. For example, if we decide to change the background color of each editbox back to clWhite again, we could either select every editbox on every frame instance on each form of all our projects, or we could just open the original frame source code and change it right there. A single recompile is all that's needed to propagate the changes to the project(s) you're working on. Exactly like form inheritance, but somewhat less fragile (form and frame .DFM files can be stored as text files in Delphi 5).
Apart from the easy way to change the look and feel of frames in your applications, the main benefit is of course the fact that a frame is a piece of functionality that can be used over and over again. It's truly a GUI super-component. And it gets better: suppose you drop a large image on a frame and re-use it all over your application. Without frames, each image would be an actual copy, using actual resources. When using frames, however, each image would be pointing to the same (original) frame resource. It also offers us the option of allowing us to change the bitmap at all locations whenever we want by modifying it on the original frame.

Frame Restrictions

A few restrictions apply when using frames. The most important one to keep in mind is that **you cannot delete any sub-components from a frame (instance) once you've placed it on a form.** By that I don't mean that you cannot delete a sub-component from the original frame - of course you can - it will only break all your code that uses this frame! No, the real restriction is that you cannot delete a sub-component on the "used" frame. When you drop the FramePassport on a form and select the "Country" label and "EditCountry" editbox in order to press delete and remove them, you'll get an error message that says that "the selection contains a component introduced in an ancestor form which cannot be deleted". And quite right. In order to "delete" a sub-component, we have to make it invisible (the end-result will be the same). Resizing the frame so you won't be able to see the Country label and editbox again won't work, as this will simply produce a set of scrollbars.

Some Guidelines

Apart from the restriction that we cannot delete sub-components from frames that we use, there are a few more guidelines that are important to abide when you think you are ready to start using frames. Most are only recommendations, but I found out the hard way that it's best to make sure you follow them.

The most important rule is: **finish your frame before using it.** This seems obvious, but you must believe that just about any change you might make to your original frame will affect all places where the frame is used. Of course, this is one of the benefits of using frames (change the original frame and all places where you use it are automatically updated as well), but it is a feature which can be dangerous.

As a consequence: **give your frame and all sub-components that you plan to access from the "outside" a truly sensible name**. And this is important. If you come back later and change the EditName to EditUserName for example, you'll break all existing code that refers to EditName (and must now refer to EditUserName). It gets worse: if you change the name of the Frame, you'll implicitly change the type of the frame as well. This will not be reflected in the forms where you'll use that frame, and you'll get a syntax error (it seems Delphi 5 is sometimes able to identify that a certain frame type has changed to another type, but it will first show a dialog to confirm that you may want to change type FramePassport to FrameUserInfo for example.

Conclusions

The restrictions and guidelines seem pretty tough. But it all comes down to a well thought-out plan. You must think ahead, and design your frames well. And once you have a well-designed frame, finished for use and with useful names, then you can sit back and enjoy the benefits of repeated re-use, possible updates all-at-once and efficiency resource use. All in all, frames are very powerful, you just have to use them intelligently (which also means no frame inheritance)...

TAS Advanced Technologies

D Delphi O Oplossings C Centrum

Een Schot in de Roos!

http://www.tas-at.com/doc doc@tas-at.com

Dr.Bob's Delphi 5 Clinics

De volgende Delphi 5 Clinics staan in september en oktober 2000 nog op het programma (zie ook de website http://www.drbob42.com/training):

14 september: MIDAS 3

Trefwoorden: ClientDataSet, DataSetProvider, Connections, Reconcile Errors, XML, Pooling, Brokering

12 oktober: VisiBroker for Delphi 5

Trefwoorden: CORBA, ORB, Type Library, Dynamic Interface Invocation, IDL-2-PAS, CORBA Exceptions

ActiveForm Updates

auteur: **Bob Swart**

ActiveForms are a quick way to create a browser-based application and still be able to use all of Delphi's visual components and design-time support.

If you have an existing Win32 GUI application and want to move to the web with it, then ActiveForms are certainly an option to consider (although you must be aware of the downsides: it only runs natively inside Internet Explorer, and really requires Win32 as client operating system). And apart from the code signing (security!) issue, which can take some time to setup, ActiveForms are really a quick solution to put a first version of your app in the browser. Until you want to deploy an updated version, that is, at which time you may find that the client (running Internet Explorer) not always automatically uses the new version of the ActiveForm. In fact, the current way Delphi generates ActiveForm deployment files and Internet Explorer "interact", it would be a miracle if your real-world ActiveForm would be automatically updated even once at the client side. Fortunately, I've found an easy work-around, but let's take a closer look at the cause of this problem first...

Before we start looking at the problem, I assume you've all started an ActiveForm with the "include version information" enabled, as well as all other "auto-increment build number" and "auto-increment release number" options inside the Project Options and Deployment options dialogs. If you don't include version information with your ActiveForm, then you don't even need to read the remainder of this little article, as there will be no way ever that Internet Explorer on the client will be able to determine that a new version of the ActiveForm exists. Which means clients need to stick with the original version (unless you can instruct them to either manually delete the old one, or download and install the new one).

When deploying an ActiveForm, Delphi (and C++Builder) generates a number of files. If you do not use .CAB file compression and also do not deploy any packages or additional files, then Delphi generates a single .HTM file and a single .OCX files (both with filenames equal to your projectname). The .HTM file will contain a section that identifies the ActiveForm to be loaded, which also includes the version number, as follows:

```
<OBJECT
  classid="clsid:42424242-4242-4242-4242-424242424242"
  codebase="./DrBob42X.ocx#version=1,0,0,0"
  width=640
  height=480
  align=center
  hspace=0
  vspace=0
>
```

Now, if you either specify that you want to use .CAB file compression (always a good idea) or you need to deploy packages or additional files, then Delphi will generate an .HTM file that will not directly specify the ActiveForm with a version number, but rather an .INF file (without a version number - that's the problem). This .INF file will in its turn contain the specifications of the ActiveForm and all other deployed files, including detailed version information, for example as follows:

```
[Add.Code]
DrBob42X.ocx=DrBob42X.ocx
VCL50.bpl=VCL50.bpl
[DrBob42X.ocx]
file=./DrBob42X.cab
clsid={42424242-4242-4242-4242-424242424242}
RegisterServer=yes
FileVersion=1,0,4,2
[VCL50.bpl]
file=./VCL50.cab
FileVersion=5,0,6,18
DestDir=11
```

The problem when using an .INF file (which you get - again - even if only using .CAB file compression), is that Internet Explorer only looks at the version information inside the HTM file in order to determine whether or not to download a new set of files from the server. Even if we supply a new INF file with higher version numbers, Internet Explorer will not check these version numbers and download a new ActiveForm. This has been a problem for a long while now, and a potential showstopper for real-world applications using ActiveForms.

Fortunately, there's an easy work-around: just specify the version number inside the HTM file again. When using an INF file, the HTM file will originally contain the following snippet:

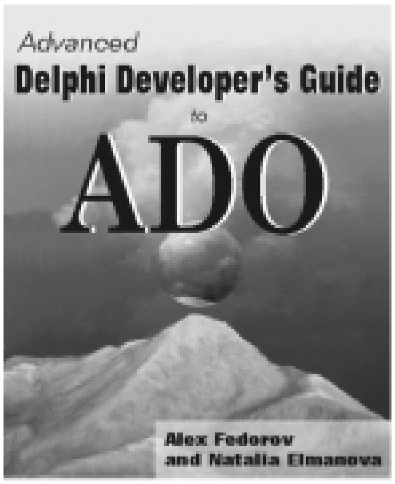
```
<OBJECT
  classid="clsid:42424242-4242-4242-4242-424242424242"
  codebase="./DrBob42X.inf"
  width=640
  height=480
  align=center
  hspace=0
  vspace=0
>
```

As soon as you enter a version number to this ./DrBob42.inf file, then Internet Explorer will be triggered to compare it to the local version, and download a new ActiveForm (and if needed packages and additionally deployed files too).
In short, the "working" OBJECT code snippet would look as follows (for the same 1.0.4.2. version number inside the DrBob42.inf file):

```
<OBJECT
  classid="clsid:42424242-4242-4242-4242-
              424242424242"
  codebase="./DrBob42X.inf#version=1,0,4,2"
  width=640
  height=480
  align=center
  hspace=0
  vspace=0
>
```

The only downside is that you need to manually adjust the HTM file everytime you deploy your ActiveForm. But that's not really a problem, since I doubt you'd want to use the HTM file generated by Delphi (or C++Builder) in a real-world situation anyway. It only means you need to look into the .INF file to find out the actual version number (which in my case increases after every build and every deployment) and modify the HTM file to reflect that new version.
And that will make sure all my clients will get the right new version of the ActiveForm next time they load Internet Explorer, as they should.

One final word of caution: if you've selected the "auto-increment build number" and "auto-increment release number" options, then you should know that the release number will be auto-incremented right *after* you've deployed the ActiveForm. In other words: right after you've deployed it for the first time, the ActiveForm version on disk (and on the server) will be 1,0,0,0. However, the file version information in your current project will then show 1,0,1,0 - i.e. the release number (third number of the series) in the IDE will always be one higher than the release number of the actually deployed ActiveForm. This is not so with build numbers (fourth number of the series) - these will always be in sync.
Remember this when your client calls and specifies the version number (and it might also be a good idea to have the ActiveForm display the version number somewhere in a lower-left corner, for example)



Book Review

Auteur: **Bob Swart**

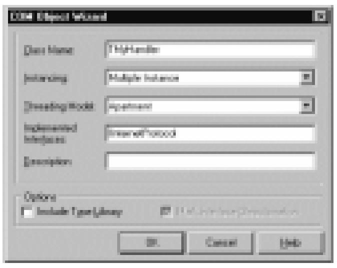
Based on the title of this Delphi and ADO book: **Advanced Delphi Developer's Guide to ADO** (from Alex Fedorox and Natalia Elmanova), I expected "only" coverage of ADOExpress in Delphi. However, that's only a part of this book. The book consists of 23 chapters, starting with Microsoft Data Access Components (MDAC), OLE DB Providers, ADO and its role in the Delphi Database Architecture. Delphi 5 Enterprise ADOExpress specific components like TADOConnection, TADOCommand, TADODataSet etc. are all covered in a lot of detail - as expected.
However, apart from these "basics", the book also contains chapters that explain how to actually build ADO Applications, and how to do Business Graphics and Reporting with ADO.
Even after those chapters, we're only still halfway the book. Further topics include OLAP and ADO, ADOX in Delphi, JRO Objects, and a very helpful chapter on deploying ADO applications. Chapter 19 and later introduce Distributed Computing; Windows DNA (Distributed interNet Applications), including RDS and MTS with ADO. There's even a chapter on MIDAS ADO Applications. There are also a number of appendices; and the most interesting one covers BDE to ADO migration issues!

All in all, this book sure is a very useful book if you want to use ActiveX Data Objects with Delphi 5 Enterprise (also with C++Builder 5) Enterprise, so you may want to check it out (a more detailed book review will be posted on the website http://www.drbob42.com soon).

Asychronous Pluggable Protocol Handlers

Auteur: **Arnim Mulder**

In Delphi kun je browser-functionaliteit toevoegen door gebruik te maken van de TWebBrowser component uit de Internet-tab (Delphi 5). De enige systeemeis om een applicatie met een webbrowser te kunnen draaien is dat Internet Explorer 4 (of hoger) geïnstalleerd is. De TWebBrowser component geeft toegang tot de webbrowser functionaliteit van Internet Explorer van Microsoft. Internet Explorer haalt normaal gesproken de gegevens of van het internet of uit een bestand op de harde schijf. Maar als je vanuit je Delphi applicatie zelf de data wilt aanleveren zul je hier op de een of andere manier toch met tijdelijke bestanden moeten werken….tenzij je je eigen protocol voor Internet Explorer implementeert! En dat is wat we nu gaan doen. In ons voorbeeld gaan we het DOC-protocol ontwikkelen zodat we in de browser van onze Delphi applicatie de url doc://urlnaam in kunnen typen en dan zelf de data aan kunnen leveren. Door de interface IInternetProtocol te implementeren in Delphi kunnen we de controle krijgen over de datatoevoer voor ons doc:// protocol. De interface definitie is te vinden in URLMON.PAS. IInternetProtocol is afgeleid van IInternetProtocolRoot, dus we dienen ook de methoden van de laatstgenoemde te implementeren. We maken een nieuw com-object door in het menu File, New, ActiveX, Com-object te kiezen.


COM Object Wizard

Class Name: TNSHandler

Instancing: Multiple Instance

Threading Model: Apartment

Implemented Interfaces: IInternetProtocol

Description:

Options

☐ Include Type Library

☑ Mark interface Oleautomation

OK Cancel Help

We moeten dan Include Type Library uitvinken omdat de interface definitie al bekend is en wij deze voor ons object niet hoeven te genereren. Daarnaast wordt de handler afgeleid van IInternetProtocol (uit UrlMon.pas) en dient deze naam te worden ingevuld in het veld Implemented Interfaces.

De code ziet er dan als volgt uit:

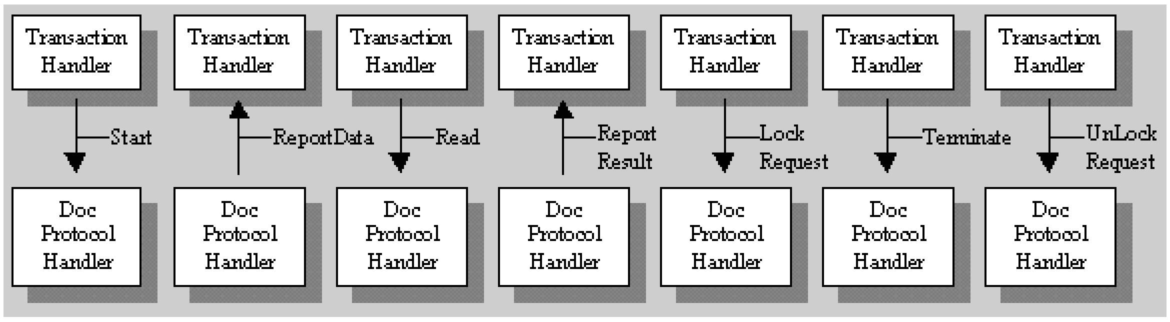
```
// onze Namespace handler voor het
// TAS-AT Delphi OplossingsCentrum

TNSHandlerDOC = class(TComObject, IInternetProtocol)
protected
// IInternetProtocol Methods
  function Abort(hrReason: HResult;
    dwOptions: DWORD): HResult; stdcall;
  function Continue(const ProtocolData:
    TProtocolData): HResult; stdcall;
  function Resume: HResult; stdcall;
  function Start(szUrl: PWideChar; OIProtSink:
    IInternetProtocolSink;
    OIBindInfo: IInternetBindInfo;
    grfPI, dwReserved: DWORD): HResult; stdcall;
  function Suspend: HResult; stdcall;
  function Terminate(dwOptions: DWORD):
    HResult; stdcall;

// IInternetProtocolRoot Methods
  function LockRequest(dwOptions: DWORD):
    HResult; stdcall;
  function Read(pv: Pointer; cb: ULONG;
    out cbRead: ULONG): HResult; stdcall;
  function Seek(dlibMove: LARGE_INTEGER;
    dwOrigin: DWORD;
    out libNewPosition: ULARGE_INTEGER):
    HResult; stdcall;
  function UnlockRequest: HResult; stdcall;
end;
```

Wanneer welke methode wordt aangeroepen is weergegeven in onderstaande figuur. De pijl geeft aan wie actie onderneemt. Meer informatie over dit onderwerp is te vinden op de MSDN website (http://msdn.microsoft.com/workshop/networking/pluggable/overview/overview.asp).

De methode Resume en Suspend zijn al wel opgenomen in de interface, maar worden nog niet gebruikt door de browser. De implementatie van Abort is niet verplicht en deze methode implementeren we dan ook niet.



De zojuist genoemde methoden zijn niet opgenomen in het schema (zie vorige pagina) omdat ze niet zullen worden aangeroepen. De implementatie van niet noodzakelijke methoden is eenvoudig:

```
function TNSHandlerDOC.functienaam(…) : HResult; stdcall;
begin
  Result := E_NOTIMPL; // functie niet geimplementeerd
end;
```

De methode Start heeft als doel de URL te verwerken en de juiste data op te halen. Je kunt kale tekst, jpegs en HTML code aanleveren. De browser zal zelf ontdekken welk bestands-formaat wordt aangeleverd. Om direct aan de vraag naar data te kunnen voldoen halen we de gegevens in één keer op en slaan deze op in een stream. Hiervoor voegen we aan de class een private member toe van type TMemoryStream. Nadat de gegevens zijn opgehaald moet de Transaction Handler met ReportData worden ingelicht dat er data klaarstaat om opgehaald te worden. Vanuit de Transaction Handler wordt actie ondernomen door methode Read aan te roepen. Wat dan gebeurt is dat de Transaction Handler onze stream beetje bij beetje leeghaalt. De Read methode kan dus vaker worden aan-geroepen, ook al is er geen data meer aanwezig! Nadat alle data uit de stream is gelezen moet de Transaction Handler worden geïnformeerd dat de transactie succesvol was. De code van Start en Read is als volgt:

```
function TNSHandlerDOC.Start(szUrl: PWideChar;
     OIProtSink: IInternetProtocolSink;
     OIBindInfo: IInternetBindInfo;
     grfPI, dwReserved: DWORD): HResult;
const MyData = ‘<HTML>De ontvangen URL is ‘+
               ‘<B><I>%s</B></I></HTML>’;
var MyText : string;
begin
  MyText := Format(MyData, [szUrl]);
  FmyStream := TMemoryStream.Create;
  FmyStream.Write(MyText, Length(MyText));
  FmyStream.Seek(0,0); // plaats cursor aan begin stream
{ er is data aanwezig, de stream kan leeggehaald worden }
  OIProtSink.ReportData(BSCF_FIRSTDATANOTIFICATION or
    BSCF_LASTDATANOTIFICATION or BSCF_DATAFULLYAVAILABLE,
    FmyStream.Size,FmyStream.Size);
{ alle data is succesvol doorgegeven }
  OIProtSink.ReportResult(S_OK,S_OK,nil);
  Result := S_OK;
end;

function TNSHandlerDOC.Read(pv: Pointer; cb: ULONG;
                   out cbRead: ULONG): HResult; stdcall;
begin
  cbread := FmyStream.Read(pv^, cb);
  if FmyStream.Position >= FmyStream.Size then
    Result := S_FALSE
  else Result := E_PENDING;
end;
```

Na ReportResult zal de Transaction Handler het initiatief nemen om eventueel nog aan-wezige data op te halen. Omdat er in dit voorbeeld in geen geval nog data aanwezig is mag het resultaat S_OK worden teruggegeven aan LockRequest en UnlockRequest.

```
Result := S_OK; // functie succesvol uitgevoerd
```

Direct daarna wordt de methode Terminate aangeroepen. Hierin moet gealloceerd geheugen en resources weer worden vrijgegeven die voor de transactie zijn gebruikt. In het geval van onze memorystream moeten we de geheugenruimte hier weer vrijgeven.

```
function TNSHandlerDOC.Terminate(dwOptions: DWORD):
  HResult; stdcall;
begin
  FmyStream.Free; // geef resources weer vrij
  Result := S_OK;
end;
```

Het laatste wat nu nog moet gebeuren is het registreren van ons protocol. We zullen het protocol alleen registeren voor onze applicatie. Vanuit een extern browser window kan ons protocol niet worden aangesproken. We maken nu een nieuwe applicatie en voegen onze zojuist gemaakte namespace-unit toe aan ons project. Daarna plaatsen we een label, edit-box, button en webbrowser op ons form. In de FormCreate event plaatsen we de code om ons protocol te registreren en in de FormDestroy plaatsen we de code om ons weer te verwijderen:

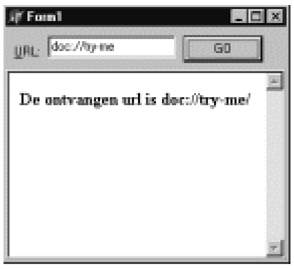
```
uses ActiveX, UrlMon, unit_docnamespacehandler;

const MyProtocol = ‘doc’;
var
  Factory: IClassFactory;
  InternetSession: IInternetSession;

procedure TForm1.FormCreate(Sender: TObject);
begin
  CoGetClassObject(Class_NSHandlerDOC, CLSCTX_LOCAL_SERVER, nil,
                   IClassFactory, Factory);
{ Creeer een sessie waarbinnen ons protocol
  kan worden geregistreerd }
  CoInternetGetSession(0, InternetSession, 0);
{ Registreer ons doc:// protocol voor onze eigen applicatie }
  InternetSession.RegisterNameSpace(Factory, Class_NSHandlerDOC,
                                    MyProtocol, 0, nil, 0);
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  InternetSession.UnregisterNameSpace(Factory, MyProtocol);
end;

procedure Tform1.Button1Click(Sender : Tobject);
begin
  WebBrowser1.Navigate(edit1.text);
end;
```



De ontvangen url is doc://try-me/

Hiernaast staat het resultaat van de zojuist gemaakte applicatie. De sourcecode van dit voorbeeld is terug te vinden op de website van het TA-AT DOC te http://www.tas-at.com/doc.

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.

Delphi 5 Clinics

De doelstelling van het Delphi OplossingsCentrum wordt ondersteund met een aantal middelen, waaronder de Delphi Clinics. 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).
Voor de komende maanden staan in ieder geval nog twee normale Delphi 5 Clinics op het programma. Allereerst op 14 september een Clinic over MIDAS 3, en op 12 oktober over CORBA (VisiBroker for Delphi).

14 september: MIDAS 3

Op deze dag gaan we multi-tier toepassingen ontwikkelen met behulp van Borland's MIDAS (Multi-tIer Distributed Application Services Suite) versie 3 - een grote verbetering t.o.v. voorgaande versies van MIDAS. Zaken als DataSetProvider, ClientDataSet, XMLBroker en verschillende Connection componenten (DCOM, TCP/IP sockets, CORBA, HTTP) komen aan de orde, maar ook interne zaken als het briefcase model, reconcile errors, en de nieuwe stateless remote datamodules.
We zullen een MIDAS 3 server bouwen die zowel met een MIDAS 3 Windows client als met een MIDAS 3 internet (web) client zal communiceren.

12 oktober: VisiBroker For Delphi

Op deze dag zullen we leren "communiceren" met CORBA (VisiBroker). Zowel in Delphi 5 als JBuilder zullen we CORBA objecten, clients en servers maken die alle met elkaar communiceren. We beginnen met een Delphi Server voor JBuilder Clients (makkelijk) en vervolgen met Delphi Clients voor een JBuilder Server (iets moeilijker). We zullen drie manieren zien om een Delphi Client aan een CORBA Server te koppelen: via de Type Library import unit, via Dynamic Interface Invocation (DII) en via de IDL-2-PAS. Deze laatste is onderdeel van de nieuwe VisiBroker for Delphi (IDL-2-PAS), die daarnaast nog zaken ondersteunt zoals structured parameters en CORBA exceptions - zowel Client-side als Server-side!

Voor beide trainingsdagen zijn nog enkele plaatsen beschikbaar. De prijs is 895 gulden (ex. btw) per persoon (ex.btw), maar u krijgt 100 gulden korting op deze prijs als twee of meer personen van hetzelfde bedrijf meedoen. Voor meer informatie, zie de websites te http://www.tas-at.com/doc of http://www.drbob42.nl/training of stuur een mailtje naar **doc@tas-at.com** voor meer informatie.