FoxRockX

Your Free Issue

December 2009

Number 3

Dear FoxPro Developer,

Dear FoxRockX Subscriber,

here comes another free copy of FoxRockX for you! This is not a regular issue but a special issue dedicated to a topic that's very interesting for Visual FoxPro Developers! As you know the FoxPro community is still alive and keeps going. And the large number of active users of Visual FoxPro with their amazingly large number of FoxPro applications in production are interesting to other companies. The lack of efforts from Microsoft for Visual FoxPro does not mean that other companies are not allowed to extend their products to work together with Visual FoxPro and come up with interesting offers to the FoxPro community. One of these offers is the Visual FoxPro Netcompiler from eTecnologia.

This free special issue of FoxRockX contains all the details you need to know.

Boudewijn Lutgerink is the author of this special issue. He is a kind of an evangelist for the VFP.NET compiler and delivered presentations for this tool at local events and conferences many times. In a merry go round he shows you a lot of features you might never have heard about before which are available in this exciting product. But please bear in mind that it is still a beta version, and we all have to wait some more time until the final version becomes available.

Once in a while we will report about the progress of these projects in the regular issues of FoxRockX magazine. Hopefully you will continue your subscription or in case you are not a subscriber yet will become a subscriber. Here the details again about our small magazine dedicated to Visual FoxPro:

FoxRockX is published bimonthly with 24 pages DIN A4 and your subscription includes access to the online archive of FoxRockX as well as of all issues of former FoxTalk and FoxTalk 2.0 magazines. Sometimes we add a free issue (with sponsored articles) like this one - at least once a year and possibly more often. The Annual On-Line Subscription rate is US$ 99.00 / 75.00 EUR.

For more details visit our homepage at http://www.foxrockx.com. To subscribe now, visit either http://shop.dfpug.com (Europe/Asia) or http://www.hentzenwerke.com (USA/Canada). On-line articles, archives and companion materials are accessible for subscribers through the "FoxRockX" tab at http://portal.dfpug.de. The access information will be sent with the confirmation of your subscription.

The VFP Netcompiler from eTecnologia

The hope and future for VFP Developers

Boudewijn Lutgerink

There's a new kid on the block, the word is that it is mean, lean and blazing fast. The kid is the karate kid of the developer tools. It cuts "time to ship software" in half like a true karateka cuts stones and pieces of wood, it has no mercy with data and strings and it has some "weapons" developers now only can dream of. Even worse, it is in the hands of a gang that consist of software weapon smiths, and they have some heavy artillery for "the Kid" in mind.

The biggest mistake MS made in .Net is that they make accessing the second most expensive asset of companies extremely hard. This asset is the data in their databases. Accessing and manipulating data is, was and will always be the strongest side of "the Kid" and its fantastic parent Visual FoxPro. The current tools emphasize too much on doing OOP for the sake of OOP that simple access to data is near to impossible. This is the area where the "The Kid" kicks doors like they are made of paper.

The name of "The Kid" yet unknown, but that is gonna change, for now we call it the VFP.Net. In its current state it is a product under development. The manufacturer, eTecnologia from Guatemala and Miami, is working on it for a bit more than two years. Although in beta phase, it already gives a tremendous insight of the possibilities for developers who care for fast data access.

This article shows you how you can use this new tool and mix and mingle the well known VFP language with the new and exciting possibilities the .Net framework offers. IMO the classes in this framework are extremely well worked out, and if you are serious about your work, knowledge of the .Net framework is part of your future work. So, .Net developers of the world, shake, shiver and say your prayers, because "The Kid" is on its way to teach you a lesson!

Introduction

In March 2007 Microsoft came with the dreaded message that there would never be a version 10 of Visual FoxPro.

From their point of view, rewriting VFP to the .Net platform was not feasable and doing so would take a huge effort, so they declared the fox dead. That this was also due to the fact that MS did not do any serious effort to market VFP as THE strongest database tool they had seems to be a blind spot in their eyesight.

A group of rebellious developers from Central America and the USA thought differently. They saw the power of VFP in the context of datahandling, they saw that rewriting this "capo di capi" of datahandling tools was THE way to go for .Net tools. They saw the true strength of the Fox where it comes to data and string manipulation, an area where all other .Net languages leave much to ask for.

So they took up the heroic task of rewriting the language as we know it for the .Net platform. In this article I will touch the possibilities of this new language, the rising star in the .Net world. I have no doubt that you will agree that this will be the incarnation of a language that always was the star in datahandling. As said, I can only touch the possibilities since I only have about 22 pages to fill, where my complete whitepaper already is going way over 100 pages. A book is on its way. Hopefully it will be published somewhere next year, at the same time VFP.Net will see its version 1.

Introducing VFP.Net

The Netcompiler for VFP, or VFP.Net as it is already called by its fans, offers you the language you already know.

For a better understanding, VFP in this context has the meaning of Very Fast Productivity.

The commands and functions of VFP are covered now for more than 90%. Some older functions, like DDE functions will never be included as these became obsolete due to the new technologies.

At the same time VFP.Net gives you access to all the classes that .Net assemblies offer you, either developed by MS or by 3rd party manufacturers.

eTecnologia also took a good look at the possibilities other languages had to offer.

This lead to language enhancements that you will most definetely like. I tell more about that later on.

As a result of having that well known language it offers you also the possibilities to access data the way you are used to, simple, fast and furious. . .

Here's an overview of the features the VFP.Net offer you right now:

- Code is compiled for the CLR.
  This means that any code you write and compile into an assembly can now also be used by other languages in the .Net world, whether that language is C# or VB.Net, cobol.net, vulcan.net or whatever is available.
- The language is embedded in the SharpDevelop IDE.
  SharpDevelop is an open source project, it also offers you a managed code environment. This is in contrast with Visual Studio, that still relies heavily on the COM model.
  The SharpDevelop IDE looks exactly like the IDE of Visual Studio, with all the tabs and tools you know, plus, again, some enhancements.
- The SharpDevelop environment works with project and file templates. These templates are actually XML files that you can easily modify for your use.
- Whether you want to create an application for the Desktop, the Web or, in the future, PDA's, you can use the same language you are so used to and love.
- You can include your VFP forms in your application.
  And the other way around, you can include also C# of VB.Net forms in your applications.
  Either way your development time and efforts are protected, you don't have to start from scratch when you need a form that you relied on for so long.
- For new forms you have the choice of creating Winforms, WPF forms and the new VFP forms.
  The latter comes with visual inheritance, later on in this article I will show you a simple form.
- Function overload, it is here.
  Function overloading is the possibility to create a function with a given name multiple times but with different parameters.
  This leads to cleaner code that is easier to maintain.
  After creating these functions one call to such an overloaded function IntelliSense kicks in and shows you all the possibilities for this one function.
- Talking about functions and parameters we can also now strong type those parameters.
  The LPARAMETER is still valid. Additionally you can now use the TPARAMETER keyword.
  An example will follow in this article.
- Just like the parameters we can now also declare strong typed variables, using the TLOCAL keyword.
  At the same time we can now also, on the same line assign a value to those variables.
  Using strong typed parameters also "somewhat" speeds up code execution.
- Talking about parameters the parameter array comes to mind, you can now pass an unknown number of parameters to a procedure and "catch" them in an array.
- Thinking of arrays. How often did you wish for a ZERO BASED array? I know there have been numerous times I wish I had one. Well, here is the good news, the compiler offers you zero based arrays by default.
- Blowing the lid of the table restrictions, gone is the 255 field limit, gone is the 2Gb limit.
  You can now create tables of 2000 fields and a max of 16 Exabyte on data.
  FYI, 16 Exabyte is 16,000,000,000 Gb, is that enough for you?
  2000 fields may be somewhat overdone, but I have been in situations where I could use more fields than the 255 we have now available.
- The compiler is a 64 bit compiler, ready for the future of 64 bit platforms.
- Best of all, on your conference CD you will find a beta version of the compiler so you can play with it yourself.
- For those of you who do not come to the German DevCon, SHAME ON YOU!
  But we have mercy. The beta will be available for download pretty soon as well, keep an eye on www.eTecnologia.net for more information.

Installing the Compiler

If you know drag and drop you are halfway the installation already.

Open the zipfile, in the zip you will find a folder SharpDevelop.

Drag that on your disk. Create a shortcut on your desktop from the bin/sharpdevelop.exe and you are about to start.

When you bought the Netextender, another product from etecnologia that brings the .Net framework to VFP, you will probably install the SharpDevelop environment under the eTecnologiaNetExtender folder.

As this is still a product under development you might find that some things are not entirely developed yet, other stuff may even be stronger and better already than we know from other .Net languages.

Interfaces

In VFP we are familiar with the idea of abstract classes. These classes are never instantiated in our applications, they merely provide a blue-print for classes we derive from them.

Interfaces are one abstraction layer extra between the classes we create and the design of those

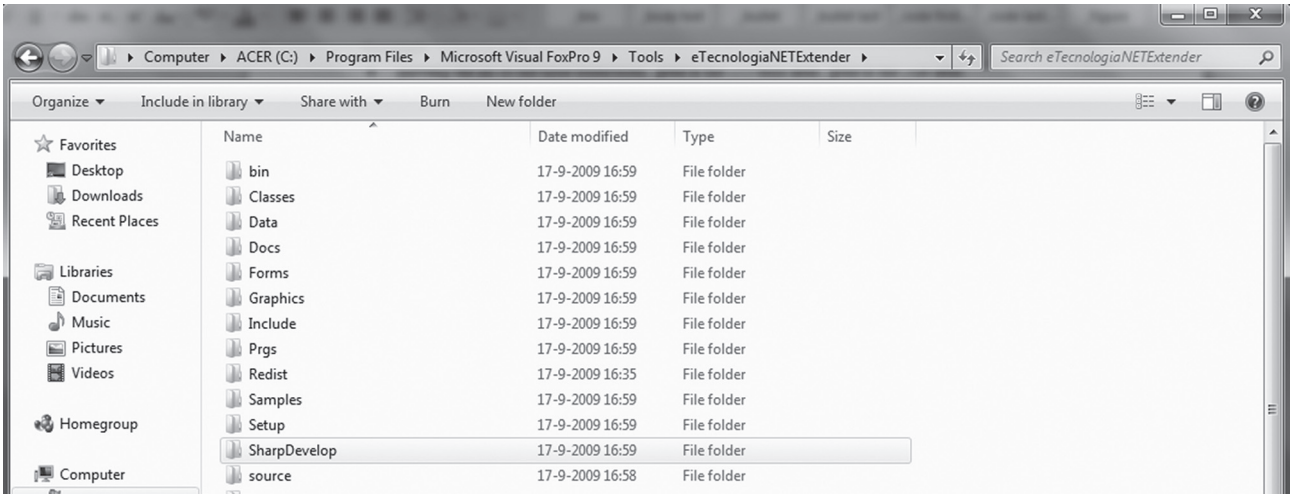

Figure 1. the SharpDevelop folder under the eTecnologia folder

Using Vista or Windows 7

If you drag it to the program files folder make sure that you start the application as admin, if you do not do that you might have trouble starting it and modifying and saving files.

Personally I have installed the samples in a different folder than the standard samples folder under SharpDevelop. Being at one session of Doug Hennig in 2007 I learned that virtualization could lead to problems if you install custom data in the program files folder (or one of its sub folders), hence my choice.

Things we always wanted (but were too shy to ask for)

As with any tool, there were "some" things we would like to have in VFP. Some of them we asked for, others were never asked for but we did envy those developers who had them.

In this section of this FoxRockX issue I will show you things that were not part of our beloved VFP language before, but, lo and behold, we do have them now.

classes. They do provide the lay-out of our classes BEFORE we create those classes.

For those who are familiar with CRC (Class responsibility Collaboration) cards, you can think of interfaces as such cards in that the interface describes the Responsibilities for the class that will implement the interface.

A class can implement more than one interface.

The syntax for defining an interface is as follows:

```
DEFINE INTERFACE BossInterface
    DelegateWork as Logical
    DoWork as Logical
    ComeLateAtWork as Logical

    Procedure raiseSalary
       Tparameter tnPercentage as Integer
ENDDEFINE
```

We can now define an abstract class, based on this interface.

```
DEFINE CLASS TheBoss IMPLEMENTS BossInterface
    DoWork = .F.
    DelegateWork = .T.
```

```
ComeLateAtWork = .T.

Procedure raiseSalary
   TPARAMETER tnpercentage as number
ENDDEFINE
```

Now that we have the base class designed we can subclass it to a class like RainerIsTheBossInFrankFurt based on the TheBoss class.

In future releases of VFP.Net, any additions you make to the Interface will be mirrored in the classes, implementing that interface.

Removing functions or properties from the interface definition will not be mirrored in your classes as this MIGHT break your code.

ENUMS, numeric data logically grouped

Enums are a special breed of grouping data. You could use days of the week, months or whatever you can come up with.

Keep in mind that enums only work with numbers, using the ToString() method of a value in the Enum will return the actual string as a value. The syntax for enums is (with the days of the week as example):

```
DEFINE ENUM DaysOfWeek
       Monday = 1
       Tuesday = 2
       Wednesday = 3
       Thursday = 4
       Friday = 5
       Saturday = 6
       Sunday = 7
Enddefine
```

You can now assign a variable like:

```
TLOCAL tnDay = DaysOfWeek Monday

?m.tnDay  && returns 1
?tnday.tostring && returns Monday
```

Macro substitution

Yeah, I hear you, we have that already in VFP. I know, but what .Net language has macro substitution?

RIGHT! That is not possible unless you use a whole lot of code with, as I understand it, reflection all over the place. (Frankly, I could not care less, macrosubstitution is easier than that always!)

Remember however that there is a slight difference between the pure VFP macro as we know it and the way it is implemented in VFP.Net.

In VFP we would do something like:

```
lcCmd = "Goto "+transform(lnRecord)
```

&lcCmd. && See the closing dot, not commonly used in VFP, but allowed nonetheless.

Mind you that closing a macro with a dot (.) with nothing following it will result in an error.

You will definetely get an error, using &lcCMD without closing period is the correct way.

Now you can create commands at your leisure and execute them with the macro substitution, something that is unthinkable for other .Net tools. But hey, was it not so that VFP's superiority was proven over and over again? It just takes time for others to understand that.

Structs, powerful "classes" to use in your code

Structures can be seen as classes with the exception that, like enums, they cannot have an explicit base class. You can instantiate them like any other object.

Structs can be defined within or outside classes. The basic idea is that you can define properties and methods/functions within them. Actually, all the data types you use are structs in .Net

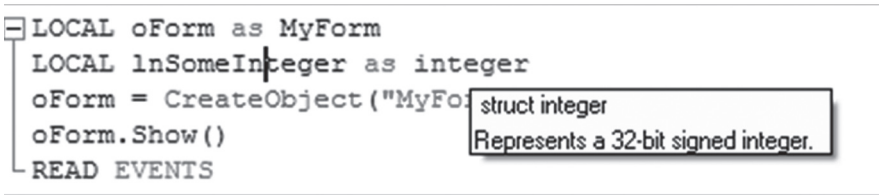

Figure 2. Representing a data value as an integer

The above picture shows a small piece of code.

The lnSomeInteger tooltip shows it as a structure, when hovering your mouse over the variable name.

In your class you create an instance of that struct and then you work with that.

A simple example:

```
DEFINE struct MyCoOrdinates
    STATIC lnX as integer
    STATIC lnY as integer
    PUBLIC Function coOrds( )
        TPARAMETERS tnX As integer, ;
                    tnY As integer
        this.lnX = tnX
        this.lnY = tnY
    ENDFUNC
ENDDEFINE
```

This structure has two properties, lnX and lnY. We use those properties to store the co-ordinates of the mouse position in our form. The function is a public function, making this available to the calling object.

In a winform you can add an object like

```
TLOCAL oCoord as MyCoOrdinates = ;
    MyCoOrdinates()
```

This one line of code creates a variable oCoord AND gives it a value, being an instance of MyCoOrdinates.

We now create a method in our form to track the position of the mouse:

```
HIDDEN PROCEDURE EmployeesMouseMove AS void
    TPARAMETERS sender as object, ;
    e as System::Windows::Forms::MouseEventArgs
```

First there is a call to the Coords function in the struct, as we saw above this functions fills the two properties lnX and lnY :

```
this.oCoord.Coords( e.X, e.Y)
TLOCAL lnXCoord = this.oCoord.lnX, ;
      lnYCoord = this.oCoord.lnY
```

Then we set the text of the labels to display the position of the mouse:

```
        this.label1.Text = ;
               „X: „+lnXCoord.ToString()
        this.label2.Text = ;
               „Y: „+lnYCoord.ToString()
ENDPROC
```

In the definition of the form the structure is instantiated, this can be done in two ways, either by the most familiar way with the CreateObject() function:

```
partial DEFINE CLASS employees2 ;
               AS WinForms::Form
oCoord = CreateObject(„MyCoOrdinates“)
```

Or in a shorthand version:

```
partial DEFINE CLASS employees2 ;
               AS WinForms::Form
oCoord = MyCoOrdinates()
```

Either way works equally well. Using the first way is more in line with the VFP way. The sequence of the events firing at startup do differ slightly from what we know from VFP (LISA G).

Note that if you want to instantiate any C# or VB.NET object you MUST use the second way. CreateObject is a function that is just a bit too strong for these MS tools.

In a true life application you would, of course, use the e.X and the e.Y values of the MouseMove procedure directly. I used this as an example of how to implement a struct. It is all up to your imagination how to use structs in your apps.

Attributes, tools you need to do your work

Every now and then we use external DLL's.

After all, the Windows API's are a true treasure case with a wealth of functionality.

Maybe you just want to use the function you are used to and meanwhile you keep looking for the function and the right class in the .Net framework. Another reason could be is that you use a 3rd party component and need that in your Net application. There can be many good and compelling reasons to use an external DLL.

Also, and I assume that this is not common knowledge, the classes in .Net that offer the WIN API possibilities actually are a wrapper around the Win API. Using these classes thus means that first the class allowing access to a given API has to be called, it then calls the API, receives a value, and then returns that value. Direct calls to API functions give you more immediate responses. Maybe not in measurable timeslots, but still it is a bit faster.

This is one place where Attributes come to rescue.

Think of an attribute as an additional tool you add to a class, one more attribute to use. The function you need is the DLLImport() function. This function is available from the system.runtime.InteropServices class.

So the first thing to do, before you define a class where you add an attribute, is make clear that you are using this class (and the methods in it) in your code.

The structure looks like:

```
USING NAMESPACE System::Runtime::;
        InteropServices
USING NAMESPACE System::Windows::Forms ;
        AS Winforms
DEFINE CLASS YourclassName AS WinForms::Form
hMenuHandle = 0
```

Once that class is defined you can now add attributes (external functions, stored in DLL's) to that class

```
[DllImport(„User32.DLL“,EntryPoint = ;
        „CreateMenu“)] ;
static HIDDEN PROCEDURE CreateVFPMenu ;
        as integer
ENDPROC
```

Of course, in VFP we do it like:

```
DECLARE integer CreateMenu IN user32.dll ;
        as CreateVFPMenu
```

This construction is still possible. The first way is more in line with the experience of those who are used to work with .Net (eg VB or C#) languages. And not only that, you can now refer to the function by the VFP name you gave it in any method or event in the form. In my example I would load the menu in the form Load event:. The difference with the VFP way of declaring API functions is that in VFP you call the function as declared, which is still possible, and in the way using attributes you call the function as a method of the class itself. So the VFP.Net way looks like:

```
WITH this
    .hMenuHandle = .createVFPMenu()
        messagebox( .hMenuHandle )
ENDWITH
```

This short piece of code gives me a messagebox showing the handle of the menu.

Also there is a remarkable speed increase using attributes.

Using an external DLL of course requires that you know about the whereabouts of the functions in the DLL, but that is where documentation is important. So in any case RTFM (read the fine manual).

There are DLL functions that need one or more parameters. The way to do that is quite similar as in the previous example. Since I used the CreateMenu function I will now show the DestroyMenu() function.

The CreateMenu() function returns a handle to a menu, the DestroyMenu() function needs this handle to destroy the right menu.

First we add an attribute to the method that makes it possible to use the correct function.

```
[DllImport(„User32.DLL",EntryPoint = ;
        „DestroyMenu")] ;
static HIDDEN PROCEDURE DestroyVFPMenu ;
        as integer
        TPARAMETERS tnMenuHandle
ENDPROC
```

In the FormClosed event you now can use:

```
this.DestroyVFPMenu(this.hMenuHandle)
```

Attributes are very powerful. As you can see in the last example I made a call to a function, using the this keyword. It looks as if the DestroyVFPMenu is a method handling everything. It actually makes a call to the DestroyMenu API.

Codeblocks

The structure of a codeblock looks like this:

```
oCodeBlock = {;
|exVal as double| RETURN exVAl^3
 }
```

Mind you, after the line: |exVal as double| RETURN exVAl^3 is NO semi-colon for line continuation! Putting a line continuation there gives you errors at compile time.

The way you can use it is like:

```
FUNCTION test_codeblock
    LOCAL oCodeBlock, bInfo[1]
    DIMENSION aInfo[3]
    aInfo[1] = 1
    aInfo[2] = 2
    aInfo[3] = 3
    oCodeBlock = {;
       |exVal as double| RETURN exVAl^3
                 }
    bInfo = AApply(aInfo, oCodeBlock)
    ? bInfo[1] && 1
    ? Binfo[2] && 8
    ? bInfo[3] && 27
ENDFUNC
```

In the following function the codeblock is passed as a delegate. Which basically means that the codeblock is transformed by the compiler into a fully working function

```
FUNCTION AApply
    LPARAMETERS aInfo, oDelegate
    FOR i = 1 TO ALEN(aInfo)
        aInfo(m.i) = oDelegate(aInfo(m.i))
    ENDFOR
RETURN aInfo
```

Strong vs weak typed variables and parameters

Visual FoxPro is a language that uses weak typed variables. This means that even if we define a variable like:

```
LOCAL lnVariable as Integer
```

We can still do this:

```
lnVariable = „This is NO integer"
```

The above code gives us no error.

This way of working with variables is still possible in the Net compiler, giving you time to get used to strong typing your variables.

Using strong typed variables has several advantages. First of all we cannot (by mistake) change the type of the variable. If we do so the compiler will complain about it. Which is a good thing, that way we are warned even before we run the compiled code, one advantage of strong typing other development environments already had, it is now available in the development language of your choice. More so we increase the speed of our applications. Compare the following two blocks of code:

```
HIDDEN PROCEDURE Button1Click AS void
   LPARAMETERS sender as object, e as ;
       System::EventArgs
    * LOCALS variation
    *
    LOCAL nValue as long, i as integer, ;
       nSeconds as number
    nValue = 0
    i = 1
    nSeconds = Int(Seconds())
    FOR i = 1 TO 5000000
       nValue = nValue +1
    ENDFOR
    messagebox( Transform( Seconds() ;
                - nSeconds ))
ENDPROC

HIDDEN PROCEDURE Button1Click AS void
    TPARAMETERS sender as object, ;
                e as System::EventArgs
    * TLOCALS variation
    *
    TLOCAL nValue as long = 1, i as integer,;
           nSeconds as number = Seconds()

    FOR i = 1 TO 500000000
       nValue = nValue +1
    ENDFOR
```

```
    messagebox( Transform( Seconds() ;
                - nSeconds ))
ENDPROC
```

The first piece of code is easy recognized as VFP code, with a few minor exceptions, being the parameters:

- The first parameter is a reference to the object calling this code (the sender)
- The second parameter (e as System::EventArgs) uses directly a Net class.

The second piece of code has the TLOCALS keyword. With that, you tell the compiler that these variables following that word are strong typed variables.

In contrast with the first block of code I do not assign a value to the "i" variable.

Since I have strong typed variables the compiler itself assigns values. In this case a value of zero.

In the for … endfor loop I increase nValue from 1 to 5,000,000 (five million, 1st block) or 500,000,000 (five hundred million, 2nd block)

The first block of code runs for nearly 4 seconds, meaning 1,250,000 (one million two hundred and fifty thousand) increments per second.

The second block of code runs in about 2 seconds. Meaning 250,000,000 (two hundred and fifty million) increments per second. Using strong typed variables makes your code quite a bit faster. (in this case nearly 200 times faster).

The machine I tested this code on is a laptop, running windows 7 ultimate with a premium Dual Core CPU running at 2.1 Ghz with 4Gb Memory.

Inline assigning values

One wish that was regularly heard when the wish list could be filled for the next version of VFP, was to have the possibility to assign values to variables when they are declared. That is possible now as well.

The LOCAL line in the first sample can also be written as:

```
LOCAL nValue as long = 0, i as integer= 1, ;
    nSeconds as number= Int(Seconds())
```

There is no difference in speed. This construction saves you some lines of code.

Also it is possible to declare whether a variable is signed or unsigned.

This can be done by simply adding the word signed/ unsigned before the name of the variable.

Variables, having the signed or unsigned keywords before them, are of course, of a numeric type.

Function overload

One concept that many programming languages have, but that was not available in VFP as we know it, is function overload.

Function overload is the ability of a compiler to allow for two (or more) functions with the same name but with different types of parameters and return types.

Looking at VFP as we know it we can find overloaded functions as well. Take a look at the VFP Fsize() function as an example:

- You can pass it a fieldname;
- You can pass it a fieldname and a work-area number;
- You can pass it a fieldname and a table alias;
- You can pass it a filename.

Four (4) possibilities for one function. Each one of the returns a numeric value.

The advantage may be obvious. In order to create a function in pure VFP that could handle all situations as described for the Fsize() function, you had to check for the number of parameters, using Pcount(), then you have to check for the type and validity of the parameters. (What happens if you pass the alias of a table that is NOT open?). Only after all this checking you can come to the very core of the function.

Using function overload you can simplify the checks on the passed parameters because you can now split the function (taking the Fsize again) in 3 functions.

One function receiving only a string, being either a fieldname or the name of a file;

One function receiving two strings, being the name of a field and the alias of a table;

One function receiving a string, being the name of a field, and a numeric value, representing the workarea of a table.

IntelliSense, as it is available in VFP.Net is immediately recognizes the functions.

Also, as soon as you add a new version of that function in your code IntelliSense immediately sees and shows it.

The advantage of function overload is obvious. You need less code per function since your checks are less complicated. Less lines of code means easier maintenance. Keep in mind that there are 10 types of programming code, one with no obvious mistakes and the other with obviously no mistakes. The latter has most likely less lines of code and is less complicated. Also, speaking about maintenance, creating easy to read code is also having

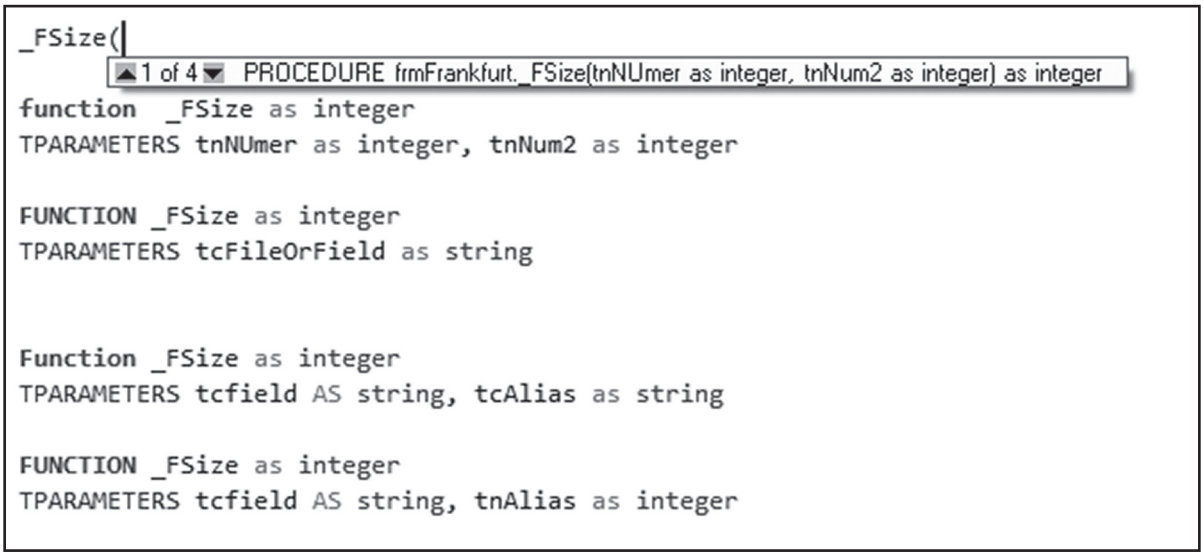

Figure 3. Function overload

mercy on the poor lad or lass that has to maintain your code in a few years from now, and that poor developer could very well be YOU.

XXX()…, a special kind of parameter

One special kind of parameter is the xxx(). . . parameter, where xxx stands for any name you wish to give. Sometimes you need to write functions that could receive an unknown number of parameters.

In VFP you could handle that with checking the vartype and/or type of the variable and the number of parameters passed. In VFP.Net there is the xxx() . . . parameter, where xxx stand for any name you want to use.

This creates a zero based array in the receiving function or method. You can skip through that array using the for next loop combined with the alen() function.

Like in:

```
PROCEDURE SomeTesting as void
   TPARAMETERS toSomeUnknowAmountOf;
               Parameters()...
        && do NOT forget the 3 dots here.
   TLOCAL lni as integer, ;
          lcSomeString as string
   FOR lni = 1 to ALEN( ;
        toSOmeUnknowAmountOfParameters,1)-1
    lcSomeString += ;
        toSOmeUnknowAmountOfParameters( lni)
   ENDFOR
ENDPROC
```

You could call this function like:

```
SomeTesting(1,2,3,4,5)
```

OR like:

```
SomeTesting("Rainerfest","in", "Frankfurt",;
        "is", "always", "Deutschgruendlich", ;
        "organized")
```

An example

In the OptionalParameters_Demo you'll find one program. I suggest that you study that sample to get an idea of the possibilities. Here is a small program that does the same, but with less code:

```
TLOCAL oInfo = TestingOptional()
oInfo.TestOptionalParam(„Rainerfest","in",;
"Frankfurt","Is","always","Fantastic!")
RETURN
```

This function is part of the TestingOptional Class that is instantiated in the first line of the prg.

The TestAdditional Function looks like this:

```
Function TestOptionalParam
    TPARAMETERS toArguments()...
    IF ALEN( toArguments,1) > 0
        FOR lni = 1 to ALEN(toArguments,1)
                MessageBox( toArguments[ ;
                        lni-1].ToString() )
        ENDFOR
    ENDIF
```

And that is all there is! More than enough however to test the results of several calls to that function.

An unknown number of parameters is passed to the function.

Next the toArguments array is created and one by one the values in the array are read. The array is an object.

As you can see in the Messagebox() function the loop value lni is decremented with 1 with every call.

Arrays in VFP.Net are now zero based, as opposed to arrays in VFP that are 1 based.

So for the code you have to take care of that, not doing so will lead to errors.

We put a breakpoint in the function to see what happens to the toArguments() values, using the debugger. To place a breakpoint select the line and press F7 OR click on the gray column before the line number. This works ONLY if there is no icon in that gray band, if there is you will get a context menu that allows you to:

- Rename the member;
- Find overrides of that member;
- Find references to that member, or;
- Open the .Net Reflector.

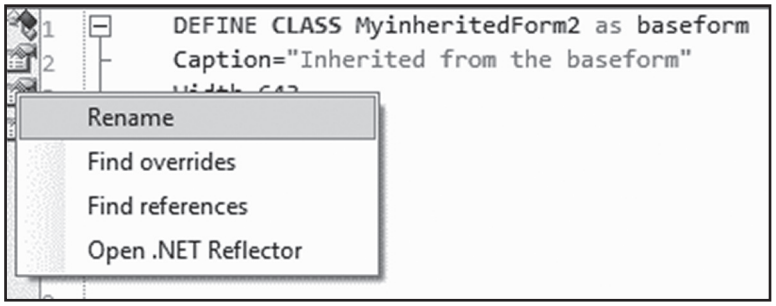

Figure 4. The contextmenu of a property

When we reach the breakpoint we can look at the local variables (Figure 5) by pressing view → Debug → local variables. Step through the code by pressing F11

In the above picture you can see the values in the array toArguments, also you can see that this is a zero based array, something we always wanted. Let's explore that a bit more.

In our next call to the function we first create an Array and fill that with values, then we make a call to the function:

```
TLOCAL TestArray(6)
TestArray(1) = „Rainer"
TestArray(2) = „Is"
TestArray(3) = „The"
TestArray(4) = „Boss"
TestArray(5) = „in"
TestArray(6) = „Frankfurt"

oInfo.TestAdditional(TestArray)
```

The result is "not quite" what we expect (Figure 6).

We get an error saying, Index was out of the Bounds of the array.

Looking at what happens in the debugger we see that the code stopped at the definition of the last element of the array (Figure 7).

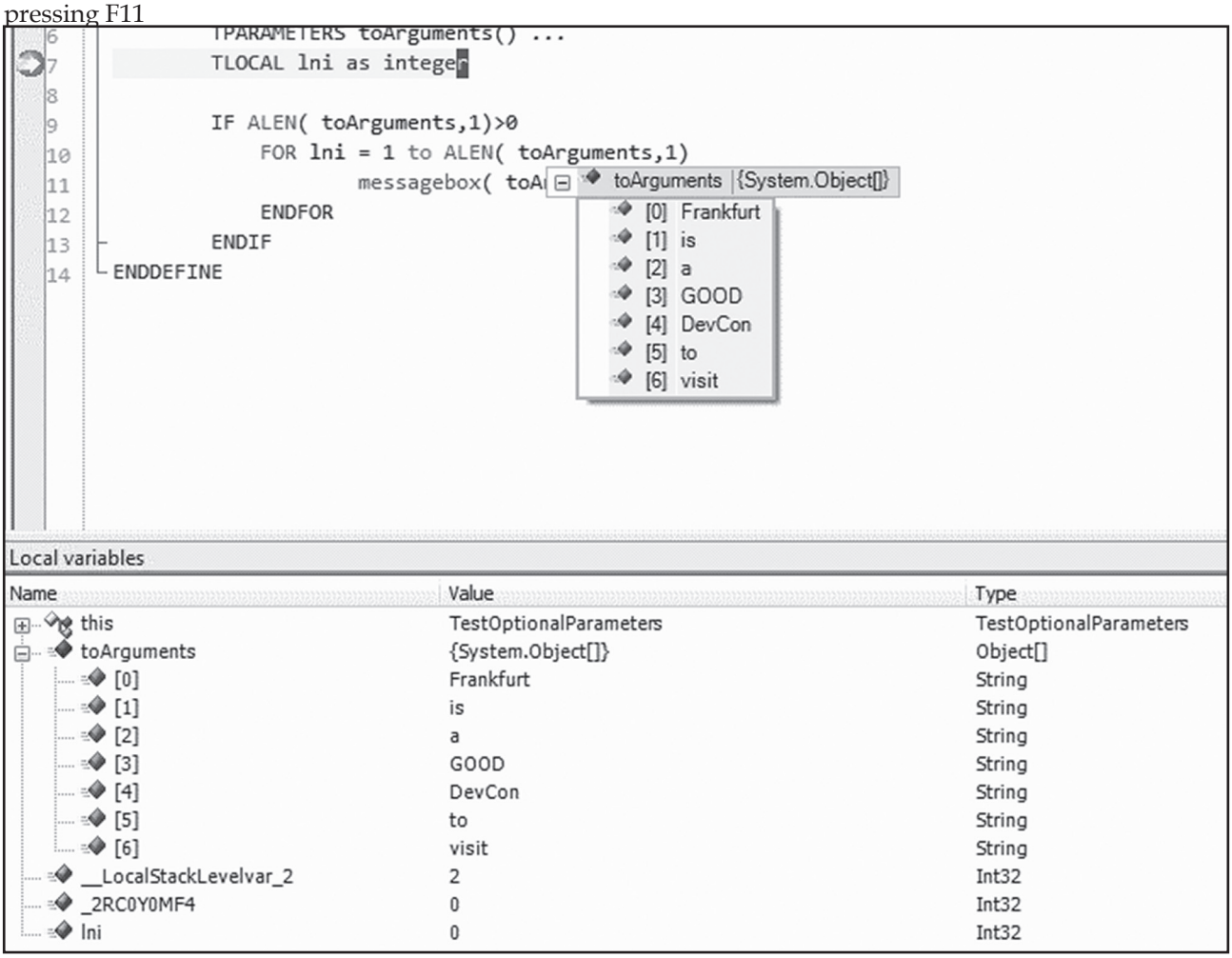

Figure 5. Debugging in action

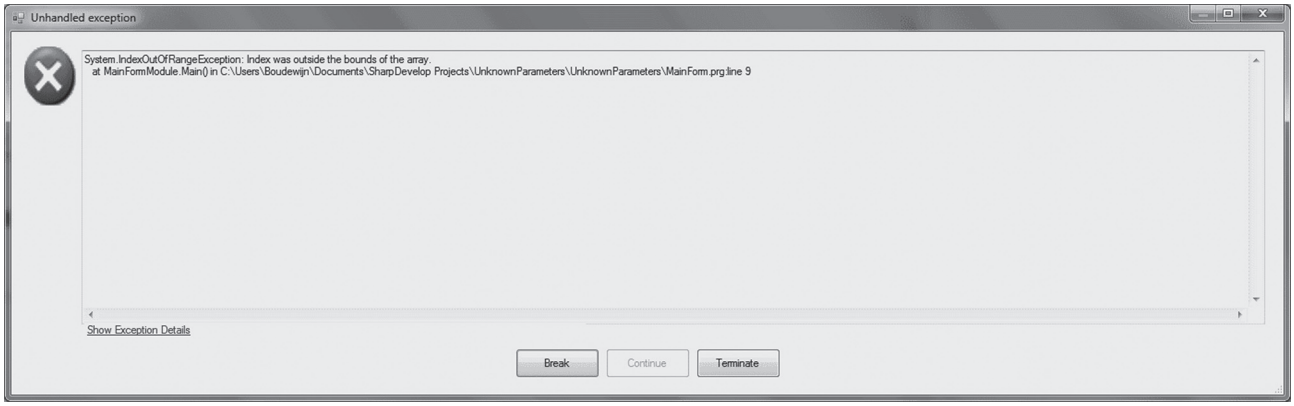

Figure 6. Duuuhhh not quite what we expect

```
TLOCAL TestArray[6]

TestArray[1] = "Rainer"
TestArray[2] = "is"
TestArray[3] = "THE"
TestArray[4] = "Boss"
TestArray[5] = "in"
TestArray[6] = "Frankfurt"
```

Figure 7. One way of defining an array

What this code actually shows is the fact that Arrays in VFP.Net are zero based.

The message, "Index out of Bounds" indicates that one element could not be found by the compiler.

So we can change the code in two ways.

We either make the dimension of the array bigger as in:

```
TLOCAL TestArray(7)
TestArray(1) = „Rainer"
TestArray(2) = „Is"
TestArray(3) = „The"
TestArray(4) = „Boss"
TestArray(5) = „in"
TestArray(6) = „Frankfurt"
```

Or we start filling the array at position 0 (zero) as in:

```
TLOCAL TestArray(6)
TestArray(0) = „Rainer"
TestArray(1) = „Is"
TestArray(2) = „The"
TestArray(3) = „Boss"
TestArray(4) = „in"
TestArray(5) = „Frankfurt"
```

Either way works, with the seconds method you are more in line with the way the compiler works (zero based arrays)

Handling NULL values

Let's take a closer look at the first way to handle array elements. We define an array as having 7 elements (first sample), and then make a call to the function in our class, of course we set the debugger so we can take a look at the elements (Figure 8).

As we can see in the debugger, the first position in our array is not filled, thus NULL

We can handle NULL values in our code as follows:

```
IF ALEN( toArgs,1) > 0
    FOR lni = 1 to ALEN(toArgs,1)
        IF toArgs[lni-1] = NULL
               MessageBox( ;
               „Yoh man, that is a NULL value")
        ELSE
```

Local variables

Name	Value	Type
this	TestOptionalParameters	TestOptionalParameters
toArguments	{System.Object[]}	Object[]
[0]	null	Object
[1]	Rainer	String
[2]	is	String
[3]	THE	String
[4]	Boss	String
[5]	in	String
[6]	Frankfurt	String
__LocalStackLevelvar_2	2	Int32
_2RC0ZPUW8	0	Int32
lni	0	Int32

Figure 8. Null values in an array

```
                MessageBox( ;
                   toArgs[ lni-1].ToString() )
        ENDIF
    ENDFOR
ENDIF
```

In the 3rd Line we check whether the value in the toArguments array is NULL

As soon as the code runs into a NULL value it displays:

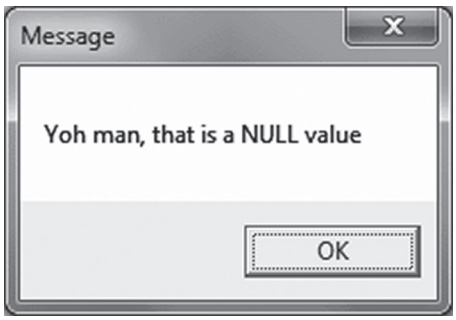

Figure 9. Handling NULL values

So, now we have two options for checking for NULL values, we can do that with an operator as in the above code and with the ISNULL() function. You now have a choice.

In the subject on weak vs strong typed parameters I told already that you can assign values in the same line where you declare the variable. There is no exception for arrays. You can assign values to an array like this:

```
TLOCAL laSomeArray[9] = {"The","German",;
"DevCon","is","the","event","of","the","year"}
```

This fills position 0 to 8, being an array with NINE positions.

Including your VFP forms in your Net-Compiler apps

It would be an absolute waste of time and effort if you could not include your VFP forms in a .Net application. All the wisdom you did put in it would go down the drain if you were not able to reuse it in .Net. No company board or manager thinking wisely would do that. (OK, I know, being wise is much asked for "managers").

There are however several issues at the moment you have to take care for.

Single threaded application

On an application level you have to make sure that your application is single threaded. This is necessary at the moment when you use OLE controls.

Not really obvious but nonetheless very true is the fact that functions like GetFile(), GetPict(), GetFont() also use OLE controls. To stay on the safe side of things you are strongly advised to make your application single threaded for the moment. Maybe in future versions this issue will be solved, but that is not certain at the moment of this writing (beginning October 2009). Making your application Singlethreaded is as simple as can be and requires only two lines of code.

All you need for that is a few lines of code at your startup application:

```
USING NAMESPACE System

[STAThread];
PROCEDURE Main as void
```

These lines make your application behave as a STA (Single Threaded Application).

That is all that is needed to prevent any problem with any type of ActiveX control.

_Screen is not available

This is a problem and an opportunity, first of all _screen is not available. You have to make your own _screen object.

You can use any form, be it a base form or a class you design for this purpose. And that is your opportunity to FINALLY create your own _screen object. FYI, _screen has a form base class. Take a look at the property sheet. The basis for you code is pretty simple.

```
LOCAL oForm = CreateObject(„form") && Screen
oForm.WindowState = 2
oForm.Show()
```

Toplevel forms

Every now and then applications do exist of one form only. For those applications a top level form is a good solution which is possible in VFP.Net as well. Keep in mind that using the "DO FORM XYZ" is the only syntax that will work for those forms. Creating those forms as objects and then showing them will give errors. So don't do that. Also, setting the ShowWindow property to 2 (as top level form) is not strictly necessary. The form will show anyway. The SetBack is that no matter what BackColor you give the form, it will take the backcolor you define for your applications in the system.

The mean beast called DataEnvironment

Many applications written in VFP and using the native database container, do open the tables through the DataEnvironment. From my experience the DataEnvironment is a mean beast that never was tamed really, it will bite you when you least expect it.

Using these tables and automatically opening them in the DataEnvironment will lead to errors. Use the load event for opening the tables.

SQL Select not yet implemented

So be aware that using CursorAdapters will not work yet. Use other means for this. Also, CursorToXML and XMLToCursor are not yet implemented so you need to work out other ways of doing things.

I am pleased though to tell you that the SQLXXX() (SQLConnect(), SQLDisConnect(), SQLExec() etc) functions are already implemented, so accessing remote data is possible.

Obsolete commands

Every function you define in files in your project become available for use whereever you see fit. There is no need to "set path" or to "set procedure", as you used to do in VFP projects.

The VFP menu

Every now and then you need to add a procedure to the VFP Menu. This results in a line of code like:

```
ON SELECTION PAD _2re1cj2u4 OF _MSYSMENU ;
    DO _2re1cj2u5 ;
    IN LOCFILE("MICROSOFT VISUAL FOXPRO 9\" + ;
        "TESTING" ,"MPX;MPR|FXP;PRG" ,;
        "WHERE is TESTING?")
```

This definetely will lead to errors in your application. As with everything, there is a way around this.

Andrew MacNeill created, years ago, the GenMenuX tool. That is your salvation for this situation.

All you need to do is add one simple line to your config.fpw

```
_noloc=ON
```

When you install the GenMenuX in your home() folder set the Menu Builder to this file.

Open the tools - > options and select the File Locations tab (Figure 10).

Select the Menu Builder and set it to the GenMenuX.prg

On the conference CD you will find the (adjusted) version of this tool.

Since it was written with VFP3 in mind it needed some tweaking. I did that for you, BTW with many thanks to Steven Black who found this small, albeit headaching, bug.

The afore mentioned config.fpw adjustment completely removes the LOCFILE() function.

Mix and mingle WinForms and VFP Forms

If you choose not to use the VFP menu but the .Net Menu Strip you are better of using a WinForm as your top level form. This gives you the possibility to visually design your menu. The setback is that WinForms have at least two files to deal with, the actual form.prg and the form.designer.prg.

However, you can use the CreateObject() function to start your forms. All your forms are objects in the .Net world, meaning that you can instantiate your forms as objects.

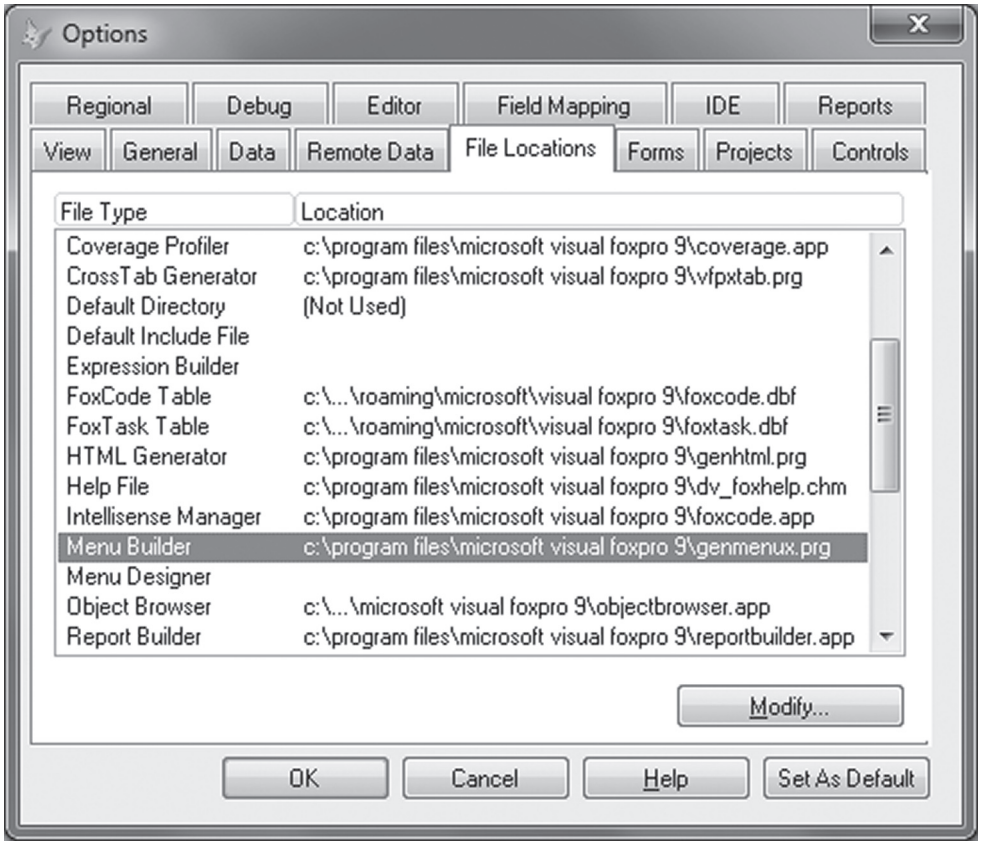

Figure 10. Adjusting the menubuilder

Visual inheritance

One thing I never could get a grip on in the .Net languages was the fact that creating classes took so much effort. You need so much code and so little visual stuff.

VFP.Net is changing that in a way we, as darned spoiled VFP addicts, are so used to.

In this chapter I will show you some basic ways of doing things. And yes, for the business logic there is of course such a thing as typing your code. Believe me, I tried sitting in lotus in front of my computer, singing AUM for hours, I had a painful butt and a sore throat, but no code saw the light.

So, here is what I did for this white paper.

First of all I created a form with several properties. This is (part of) that code.

```
DEFINE CLASS baseform  AS Form
    cTable AS string=""
    cDBC AS string=""
    cDataPath AS string=""
    lShared as boolean = .T.
    Height=186
```

This code, and that should not be a surprise for you, adds some properties to the form. Switching to the property sheet and setting that to the category brings you the following view:

Figure 11. User defined properties in the property sheet

We leave this form for now and create a new form, based on this basic form.

Select the projects tab, right click on the project (NOT the solution) and add new Item. Select the VFPForms class.

Double click that one and the code you get is:

```
DEFINE CLASS MyInheritedForm  AS Form
```

Change that to:

```
DEFINE CLASS MyInheritedForm AS BaseForm
```

Once we have this done we can take a look at the properties. It should be no surprise that the properties are indeed available (Figure 12). We always had that with VFP, so why not with VFP.Net?

By simply adjusting one line of code you inherit all the PEMs of a base form.

Filling the base form with methods and events shows these as well in the sub classed forms.

Adding custom controls to a form.

Just as you can create a base form and subclass that, likewise you can create your own custom controls. In the following example I will create a set of navigation tools. And add them to the form in the next step.

Creating a custom control (navigation buttons.)

Right click on the project and select add → New Item.

In the dialog that appears select the class template (Figure 13).

We add 4 buttons to it by dragging them from the windows tools to the form. Next we adjust the captions to "|<", "<", ">" and ">|"

The names are cmdTop, cmdPrev, cmdNext and cmdLast (Figure 14).

In this picture, you see the designer surface with the properties. The methods are simply the navigation stuff you created probably several times. I will not bore you with that.

Adding the custom control to your form.

Save that class and go back to the inherited form. There you now type one line of code immediately after the property settings.

```
Add Object cntNavigation as clsNavButtons
```

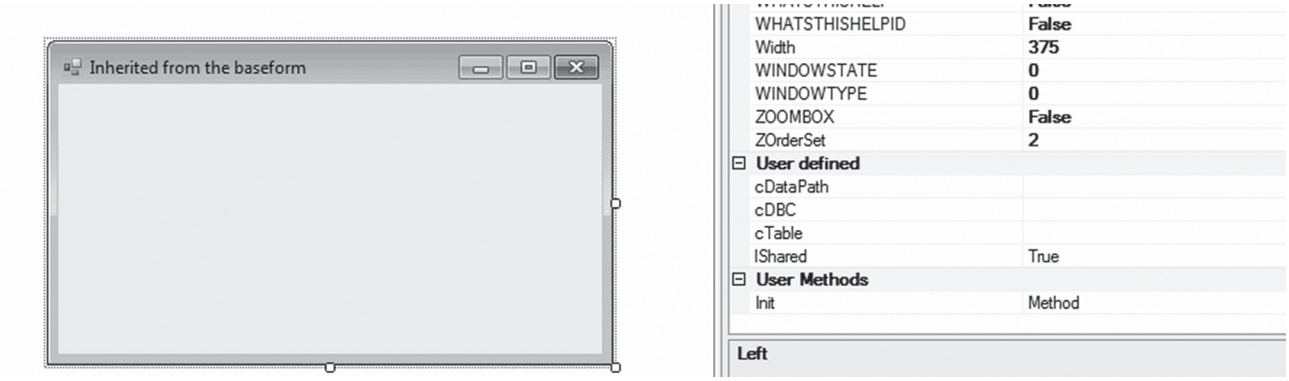

Figure 12. Inherited properties

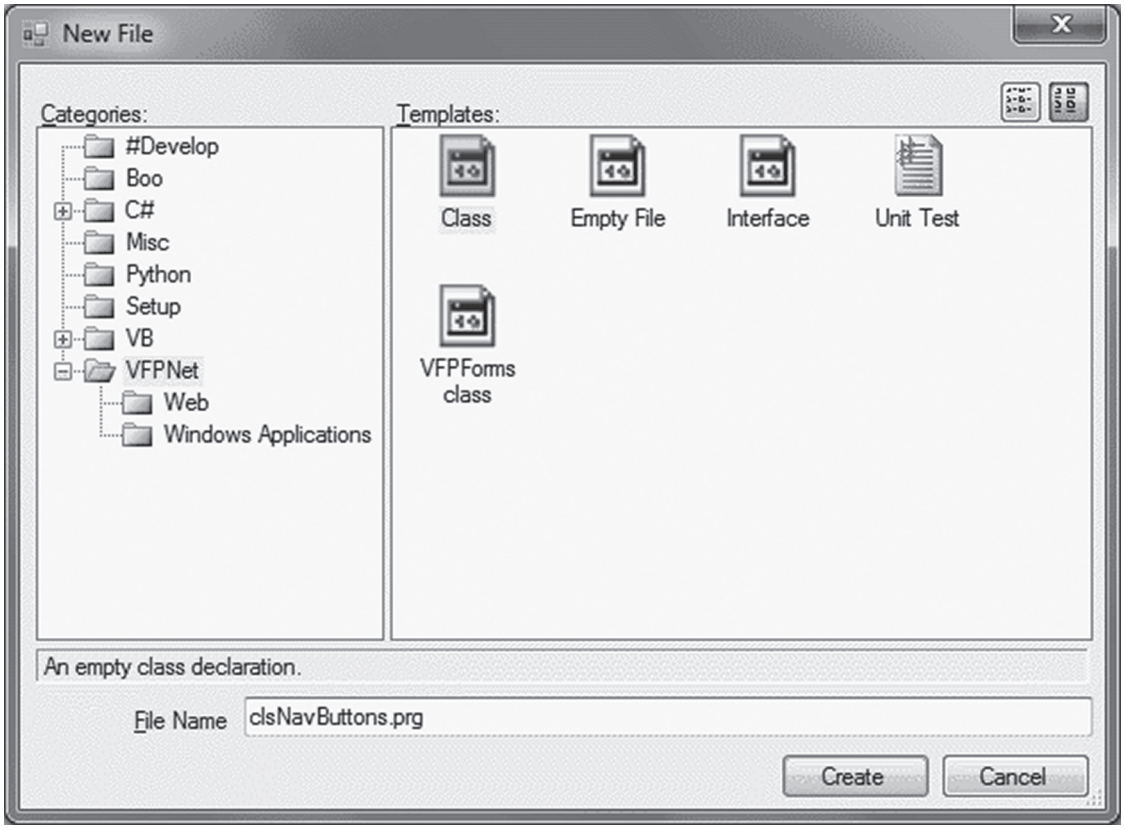

Figure 13. Creating a new class

Figure 14. The properties of a custom control

As you type this statement you will notice that IntelliSense kicks in:

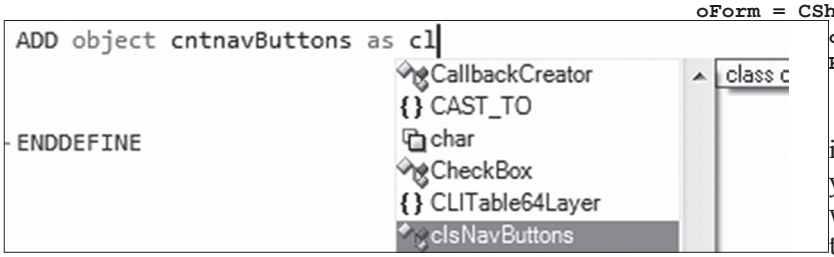

Figure 15. IntelliSense kicking in

Including .Net forms in your VFP.Net application

Just as you don't want to destroy any capital using your VFP forms in your new VFP.Net application, neither do you want to destroy any invested money from the hard gained knowledge you now may have about .Net forms and all that goes with it.

So, just as I explained how to include your VFP forms in your new application, I will now explain how to include your .Net forms in your application.

Since you developed your forms in a separate project you can include this project in your solution OR, in case you created a Net Assembly you can include that assembly in your application through the references node of the solution treeview.

In this specific case I will show a way through including several projects in one solution.

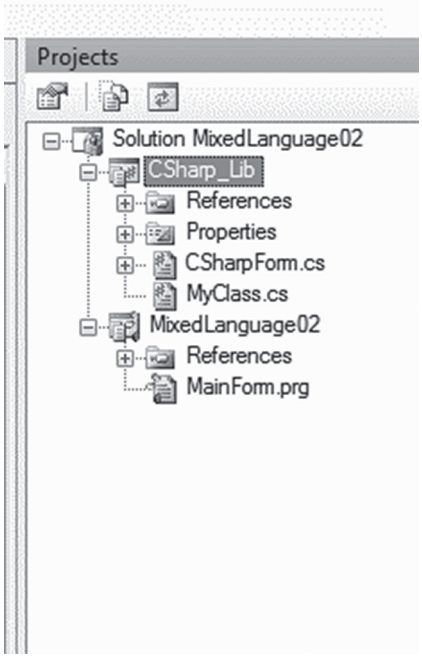

Figure 16. CSharp and VFP.Net combined

Since this project is all about including a .Net form in your VFP.Net appliucation the main code is this:

```
USING NAMESPACE System

TLOCAL oForm as CSharp_Lib::CSharpForm
oForm = CSharp_Lib::CSharpForm()
                    oForm.Show()
                    READ EVENTS
```

The CSharpForm in the Csharp_lib is instantiated and shown in the code you can find in the mainform.prg. This will show the CS form in your application.

If the form is included in an assembly the way to achieve this is exactly the same.

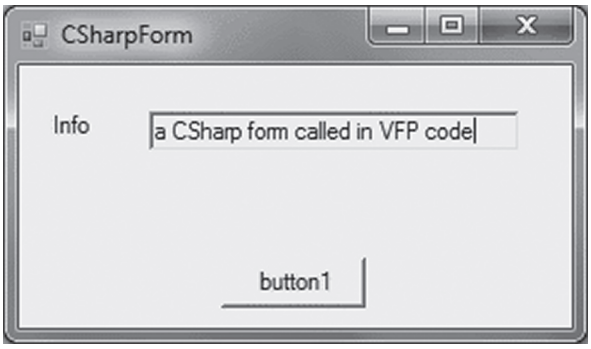

Figure 17. A CSharp form in a VFP.Net app

Templates

The Developer's Visual Studio, which is the chique name for SharpDevelop, relies heavily on templates. In this chapter I will touch the surface of what is possible with that. You can easily modify existing templates to fullfil your own needs.

Templates for SharpDevelop come in two flavors, as project templates and file templates. I will take a look at file templates here.

These templates are actually XML files with the exception that they just have a fancy extension. So they are cunningly disguished XML files. The VFP templates are stored under: Home()+"\Tools\eTecnologiaNETExtender\SharpDevelop\AddIns\AddIns\BackendBindings\VFPBinding\Templates"

A simple project template is the Windows application template. You can find that under the template folder for VFP templates, Home()+"Tools\eTecnologiaNetExtender\SharpDevelop\Addins\addins\backendbindings\VFPBinding\Templates". The filename is FormsProject.xpt, here is what it looks like:

```
<?xml version="1.0"?>
<Template >
    <TemplateConfiguration>
        <Name>VFP Windows Application</Name>
        <Category>VFPNet</Category>
       <Subcategory>Windows Applications
       </Subcategory>
        <Description>A VFP Windows Application
        with VFP Forms and WinForms
        </Description>
    </TemplateConfiguration>
    <Actions>
```

```
        <Open filename = „Main.prg"/>
    </Actions>
    <Project language = „VFPNET">
        <PropertyGroup>
                <OutputType>WinExe</OutputType>
        </PropertyGroup>
        <ProjectItems>
        </ProjectItems>
        <Files>
           <File name="Main.prg">
                <![CDATA[
USING NAMESPACE System
LOCAL oForm as MyForm
oForm = CreateObject(„MyForm")
oForm.Show()
READ EVENTS
DEFINE CLASS MyForm AS Form
Caption = „Hello from Fox"
ENDDEFINE
                ]]>
                </File>
          </Files>
    </Project>
</Template>
```

In the above template code I underlined the code that you will actually see once you create a project based on this template.

In the TemplateConfiguration section you see a name tag, a category and subcategory and a description.

These are also visible when you create a new project, as in this picture (Figure 18):

Changing the category and/or the subcategory has a direct effect on the above shown dialog (Figure 19).

Editing a Template

Open one of the files with a straightforward editor like Notepad or any ascii editor to your liking. Make changes in the template and save it under a different name.

The second way of editing is this.

Select the template file you want to work on to create a new template, copy that file and rename it. Then start SharpDevelop.

Next you drag the copy of the template from the windows explorer to the TITLEBAR of the SharpDevelop IDE.

That opens the template and you can edit it.

This way you can also make changes to existing templates.

The advantage is that you get syntax coloring and the possibility to indent/unindent blocks of code with the Tab / Shift+Tab key.

Keep in mind that changes to templates are only visible after restarting SharpDevelop.

Looking under the SharpDevelop\Data\Templates\File or SharpDevelop\Data\Templates\Project folders you will find in the Subfolders Csharp and VBNet several templates that you can change for VFP.Net.

On the conference CD you will find an extensive whitepaper on working with and editing templates, some more examples are available in that paper.

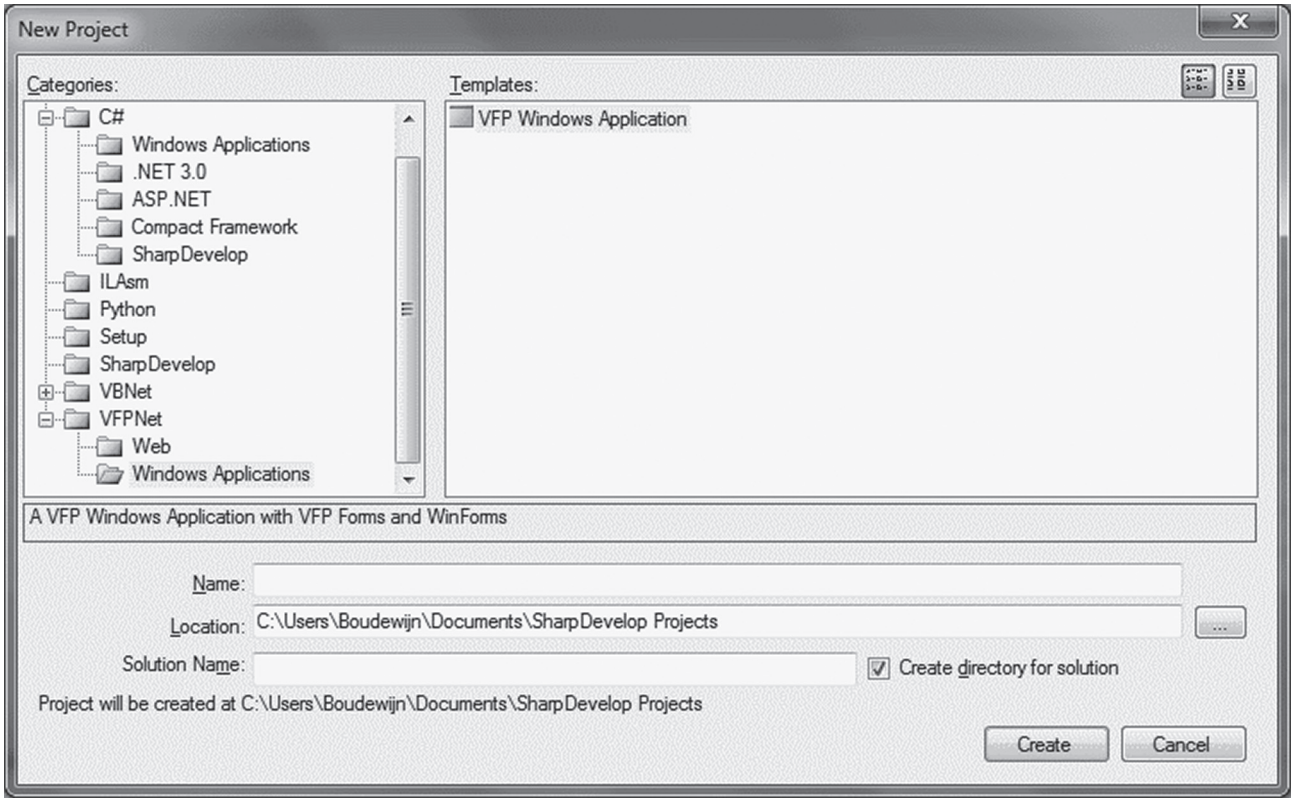

Figure 18. Fiddling around with templates

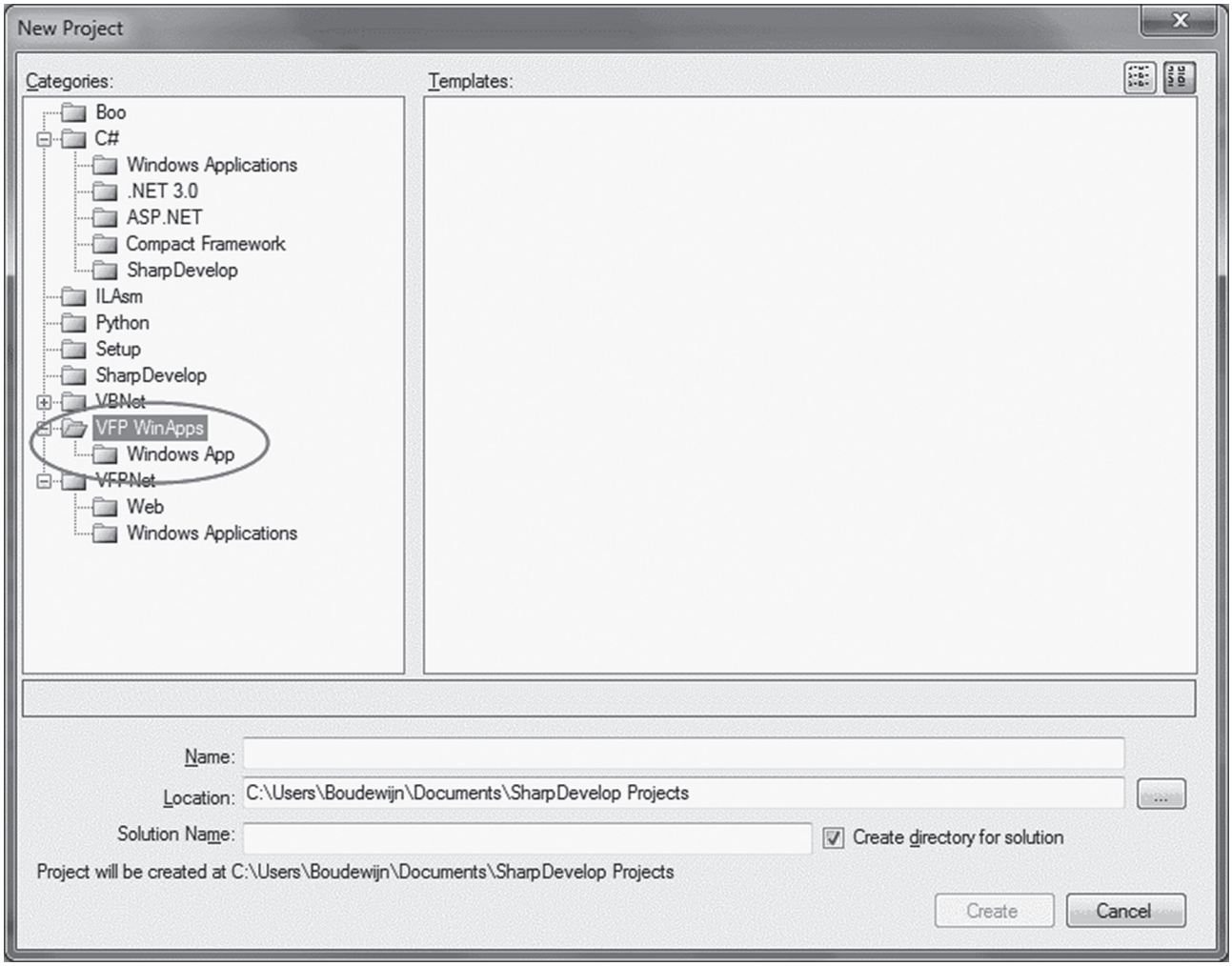

Figure 19. Category and subcategory changed in the template

What can we expect in the nearby future

Well, I can tell you that your friends at eTecnologia (hmmm, where did I hear THAT expression before!?) have a book full of features they want to implement.

So let's take a short tour on soon to be expected features, but only a few of them, otherwise you would still be reading until next week and beyond.

IntelliSense for tables

When working in VFP you can open a table in the command window and type the name followed by a dot (TableName.). This gives you a list with the fieldnames. This only happens in the command window but not in the codesnippets where it would be really useful.

VFP.Net in a future release will have this intellisense available in all code windows.

A table can also have assigned an alias to it in the DataEnvironment or in the Project Manager, it defaults to the table's name. After writing MyTableAlias and dot, you will get the list of fields available. And with the optional strong typing functionality you can get even better results like:

```
? LEFT(myTable.MyNumericField,4)
```

you will get an error at compile time because LEFT() (or any other string manipulation function for that matter) expects a string and you are using a numeric field. And of course you can catch wrong assignments.

You can also do:

```
MyTable.MyNumericField = someValue
```

This works faster than a REPLACE command.

Mind you, when you apply buffering you still should do a tableupdate(). Yeah, SOME work is left for you to do!

Visual inheritance extended to tables

Think about a table like a Designer Surface, you get a list of all the properties of the table. The events, methods and so forth. You customize the behavior of your table by changing or overriding events / methods in your table. In your table surface you can layout fields, whether those fields are created for your table or inherit from a fields library. A field is just a struct (See figure 2) and therefore in your table surface you can customize or extend them according to your new tables. This is subclassing taken to a level that is unseen yet in any development tool. (OK, promise here, the eTecnologia team will not break their hands while tapping on their shoulders.)

Some user scenarios:

You create a Zip Field or postal code field that stores the Zipnumber/Postal code, so it will always have some layout predefined, Input Mask and a validation method, and some utility methods.

Think of this as having the ability to plot a zip code in a map using Google Maps. With that you just put your field in your table and it inherits all the functionality that you wrote for its class. Quite the same as when you drag and drop a mapped field on a form in VFP. You can extend and override its functionality.

This way you can add a button to a form and in the click event have some code like: MyTable.MyZipField.ShowGoogleMap() provided you coded that at the field level.

You can create a composite type field integrating two fields: Cost and SalesPrice. You can create a rule like SalesPrice must always be greater than or equal to Cost. You can display and error or a warning when the rule is broken.

By Dragging the composed type to your table you get the two fields and the relation(s) between them that you can reuse or extend in other tables.

So the visual designer for fields is like a huge repository of fields, both as we know them now as well as custom types. And they have Events, properties and Methods. This is Data Access taken to a higher level, leaving all the other .Net languages in the cold and dark world of problematic data access.

Oh, you think that is cool hey!? Well buckle up, fasten your seat belts and get ready for the introduction of the:

Generalized Databases

Given a handle or connectionstring representing a connection to a remote database, you can create your own Database layer, and then:

You issue a "CREATE TABLE" command in VFP and that table is automatically created in the remote database.

You DROP a Table in the current database and the same table is DROPPED in the remote database.

USE "myTable" and you are using your remote table like if you did a SQLExec(nHandle, "select * from myTable")

If the back-end table has a lot of data it is unlikely that you want all of that data over the line, that would be dragging down the speed of both your application and the network you are working on. So the USE command will be extendable with a SQL select command, like USE myTable Select top 100 from my Table, thus narrowing down the amount of data you pull over the line.

If your table has a Primary key you just change your data and issue TableUpdate() and your remote data is updated. No need to setup anything else in your client side code.

You ADD TABLE and that table becomes a Remote Table automatically uploading the data as needed.

Offline / Online functionality and synchronize automatically with the remote data.

A new designer for your SCX forms

eTecnologia is working on implementing a new form designer in the VFP Developer Studio. Once this is tested and implemented you do not need Visual FoxPro to design VFP forms.

Your VFP forms as webcontrols

Would it not be nice AND useful if you simply could drag your forms on a ASPX page and run that page? With that you would have to write the business logic only once and have it running on the desktop and on webpages.

This would be developers' heaven, wouldn't it. Well, heaven is in sight. eTecnologia is working on it. So stay tuned, we'll be back after the commercials.

Object Oriented Reporting

One thing not yet implemented in VFP.Net is reporting. eTecnologia is planning on creating a fully object oriented reporting engine. This will be available in the near future.

Pointers and Native Interop

This will be a terrific feature extending the power of VFP.NET to the Native world, making it like a child's play to use classes in native libraries as if these are VFP Objects.

Let start with a sample C++ class:

```
class MyClass
{
public:
    int TestParameter(int x, int y);
    virtual bool IsValid();
    virtual MyClass* TestPointer();
    int* TestInteger();
};
```

You will be able to import that into an assembly and then use the class in that assembly from your VFP Code as:

```
USING NAMESPACE MyCPAssembly
TLOCAL oInfo = MyClass()
? oInfo.TestParameter(4,7)
? oInfo.IsValid()
oResult = oInfo.TestPointer()
? oResult.IsValid()
TLOCAL pnValue as integer ^ ;
        &&This is the syntax for pointers
*
* pointers point to an address in memo
*
pnValue = = oInfo.TestInteger()
^pnValue = 17 && changes the that
```

The line:

```
TLOCAL pnValue as integer ^
```

Ends with the caret (^) indicating a pointer.

As you can see in the last two lines first the TestInteger function of the C++ class is called first, then a value is assigned to the variable.

This feature will let us integrate external libraries as part of VFP.NET without any trouble.

Things that will come as part of that are integration of:

- Spell Checkers;
- Scanner Automation;
- OCR;
- Synonymous Dictionary;
- Advanced Image Manipulation like image filters;
- Video / Audio manipulation like Conversion and Playback;
- Computer Vision;
- Face recognition;
- Motion Detection capabilities;
- Any other kind of biometric libraries like fingerprints, iris scans, MRI software and much more.

These are just a few of the hot and cool things that you will be able to do using the NetCompiler. And yes, it will even be easier to use that from VFP.NET than from C++ or C# itself. This may sound like a boasting, but it is true. And hey, pardon me but VFP always was far superior to those other "tools". We always knew that, so why not boast a bit and tell them off. . . FINALLY!!

It is not for nothing that I subtitled this FoxRockX special "The hope and future for VFP Developers".

Another side benefit is that you can do direct pointer operations in VFP.NET that would be very, very hard to do in plain .NET, and it will run very fast in VFP.NET. This is part of a broader project that is to be able to compile VFP code to native code (object files) that can be linked and intermixed with C++ / C code and optimized to run several times faster than optimized .NET Code as I showed in the LOCALS/TLOCALS sample.

Conclusion

Geez' I wrote all this text just about VFP.Net, and I DID plan to write a short paragraph on the NetExtender as well. >sigh<, I guess you will have to visit that session then so I can bore you with that information, Sorry folks, there is simply so much to tell, and I am so thrilled by this new tool that I could go on for days on end. But then again, I guess you would be bored soon hearing about all this new and exciting stuff. Just know that the NetExtender brings the classes that are trapped in a .Net into the world of the sacred Fox, where they will live happily ever after. Just know that VFP.Net is developing rapidly and will be a tool for our future.

On-line sources

There is a google user-group for the Net compiler. The moderator is Hank Fay, one of the first Compiler addicts (like I am). You can find the group at: http://groups.google.nl/group/vfpnet-compiler-community-support-group/

If you are really addicted then know that you can find your peer addicts there.

Biography

Boudewijn Lutgerink is an addicted VFP developer who lives in the Netherlands. His first contact with automation was in the early '80's of the last century. Through a project at the company he worked for at that time he came in contact with automation and was hooked since then.

"Everybody speaks about computer viruses these days, but everybody seems to forget that computers themselves are like a virus, a dangerous addictive virus. You either get sick of them and NEVER want to deal with them for the rest of your life OR you get addicted and need more every day. The latter happened to me…"I am afarid I am addicted for the rest of my life."

He is a close friend with John Zijlstra who brought him in contact with FoxPro. Version 2.6 at that time. Boudewijn was thrilled by the speed of data– and stringhandling of the fox.

Pretty soon VFP version 3 came along, albeit in Beta and "not really stable" Boudewijn was immediately hooked on the idea of Object Oriented Programming. Through the years he was loyal to VFP to long extend.

His company, Lutgerink Economical Software Architects, did projects for the Royal Netherland Airforce, big companies like Philips and Teijin, factories, banks and daytraders and for

companies big and small in all kinds of businesses. Each and every time VFP proved its worth by the speed of development, speed in datahandling and more so, the ease of connection to a wide variety of data sources. He was a VFP-MVP in 2007 and 2008. t

He now lives in Huissen, near Arnhem, with his wife Elina, their six cats, Marmouzet, Gaya, Crystal, Barbara, Ginger and Quirine and his two border Collies,General and Private, Bordersand Sheep are another addiction of Boudewijn.

If it ain't a Border Collie, it's just a dog is his favorite expression.

If not developing asskicking software for customers he can be found in the fields training his dogs, in the kitchen cooking a meal, singing in his quartet (Base, the lowest lifeform in the singing world), playing the flute, dancing with his wife in the dance-school or doing Reiki treatments. (not necessarily in that order). If he is doing none of the above he is most likely studying on how to develop even better software.

Boudewijn can be reached at B.Lutgerink@Betuwe.net

FoxRockX™(ISSN-1866-4563)	FoxRockX is published bimonthly by ISYS GmbH
dFPUG c/o ISYS GmbH Frankfurter Strasse 21 B 61476 Kronberg, Germany Phone +49-6173-950903 Fax +49-6173-950904 Email: foxrockx@dfpug.de Editor: Rainer Becker	Copyright © 2009 ISYS GmbH. This work is an independently produced publication of ISYS GmbH, Kronberg, the content of which is the property of ISYS GmbH or its affiliates or third-party licensors and which is protected by copyright law in the U.S. and elsewhere. The right to copy and publish the content is reserved, even for content made available for free such as sample articles, tips, and graphics, none of which may be copied in whole or in part or further distributed in any form or medium without the express written permission of ISYS GmbH. Requests for permission to copy or republish any content may be directed to Rainer Becker.
FoxRockX, FoxTalk 2.0 and Visual Extend are trademarks of ISYS GmbH. All product names or services identified throughout this journal are trademarks or registered trademarks of their respective companies.	