

# The Developers Magazine

*published for members by The Developers Group*

*incorporating the DotNET Developers Group and the Delphi Developers Group*

**Autumn 2011**

---

## Contents

### FastMM - Preparing Your Apps To Report Memory Leaks..... 2

Alan Fletcher shares a quick and easy way to deal with memory leaks

### Give me some Latitude ..... 5

Joe Griffin shows you how not to get lost

### Delphi XE2 Cross-Platform using FireMonkey ..... 6

Bob Swart introduces Delphi for Mac and other animals

#### DG meetings diary 2012

January	Monday 9 <sup>th</sup>	at Sybase, Maidenhead with Brian Long
February	Tuesday 21 <sup>st</sup>	at Theodore Bullfrog, Charing Cross with Pete Sykes
March	Wednesday 14 <sup>th</sup>	at Theodore Bullfrog, Charing Cross with Jason Chapman
April	Monday 16 <sup>th</sup>	at Sybase, Maidenhead with Brian Long
May	Tuesday 15 <sup>th</sup>	at Theodore Bullfrog, Charing Cross with Pete Sykes
June	Wednesday 27 <sup>th</sup>	at Theodore Bullfrog, Charing Cross with Jason Chapman
July	Monday 16 <sup>th</sup>	at Sybase, Maidenhead with Brian Long
August	no meeting	
September	Tuesday 18 <sup>th</sup>	at Theodore Bullfrog, Charing Cross with Pete Sykes and Bob Swart
October	Wednesday 24 <sup>th</sup>	at Theodore Bullfrog, Charing Cross with Jason Chapman
November	Monday 19 <sup>th</sup>	at Sybase, Maidenhead with Brian Long
December	no meeting	

**The Developers Group, run by Joanna Goulson and friends,  
is a division of Richplum Ltd, 7 Devizes Road, Upavon, Wiltshire SN9 6ED, U.K.  
telephone 01980 630032 fax 0800 066 4336 email [Joanna@richplum.co.uk](mailto:Joanna@richplum.co.uk)  
web sites [www.dotnet.uk.com](http://www.dotnet.uk.com) and [www.richplum.co.uk](http://www.richplum.co.uk)**

# FastMM – Preparing Your Apps To Report Memory Leaks

by Alan Fletcher

One of the most challenging parts of inheriting a legacy project is to fix the memory leaks that most often are hiding in the code. A while ago, while dealing with an application that managed to eat all the available memory within a few hours I found FastMM. And it sure was a great find.

## What is FastMM?

FastMM is a memory manager replacement designed to be used with Delphi and C++ Builder. It is an Open Source project developed by Pierre Le Riche in South Africa.

Starting with Delphi 2006 FastMM replaced the Borland memory manager. Unfortunately, Delphi only ships with a subset of FastMM. Most of the useful debugging reporting that can be done with FastMM has been stripped from the shipping version of Delphi, RAD Studio and BDS.

But fear not! It is very simple to replace the stripped down version of FastMM with the full version. All you have to do is follow the directions outlined below. I have also included a step to install the *FastMM4 Options Interface* program. That program is a very friendly way of configuring the options contained in the file *FastMM4Options.inc*.

## How to install FastMM

1. Download the latest *FastMM* source code from <http://sourceforge.net/projects/fastmm/>
2. Copy the content of the downloaded zip file to a folder on your computer.
3. In Delphi add a path in *Tools>Options>Library – Win32 Library Path* to the FastMM folder that contains the unit *FastMM4.pas*.
4. Copy the file *FastMM\_FullDebugMode.dll* from the folder *FastMM\FullDebugMode DLL\Precompiled* to the Delphi install folder. For example in Delphi 2007 - *C:\Program Files\CodeGear\RAD Studio\5.0\bin* or in the Delphi XE install folder - *C:\Program Files\Embarcadero\RAD Studio\8.0\bin*
5. Optionally one can download and install the *FastMM4 Options Interface* program from JED software's web site at <http://jedqc.blogspot.com/2007/07/new-fastmm4-options-interface.html>.

## Configuring FastMM

Once you have properly installed FastMM4 you will be able to detect any memory leaks and attempts to use freed memory. But before we do that let's take a look at what we have just installed.

In the FastMM folder you will find the file *FastMM4Options.inc*. This is the file that controls how FastMM behaves. Each option is very well documented and it is how you set the default behavior of FastMM. One can manually edit this file or, optionally, use the *FastMM4 Options Interface*.

Let's take a quick look to this file. There are eight different sections in this file.

### 1. Miscellaneous Options

This section contain general settings to control memory alignment, use of *fastMove* library, multithreaded behavior and debug only when running the IDE

### 2. Debugging Options

This section contains defines that control the debugging behavior of FastMM such as logging errors to a log file, dumping of memory along with an error, stack traces and more.

### 3. Memory Leak Reporting

This section controls the reporting of memory leaks, how to deal with expected memory leaks and the presence of the IDE or debug info to report errors.

### 4. Instruction Set Options

This sections deals with using MMX instructions and this option currently only affects the variable size move routines.

### 5. Memory Manager Sharing Options

This section allows sharing of the memory manager between a main applications and DLLs.

### 6. Option Grouping

Allows you to group a set of options for the release version and the debug version of your applications. So you can have the memory manager report issues when your application has been compiled for debugging and quietly ignore memory errors when compiled for release.

### 7. Compilation Options For borlndmm.dll

If you're compiling the replacement borlndmm.dll, set the defines in this section for the kind of dll you require.

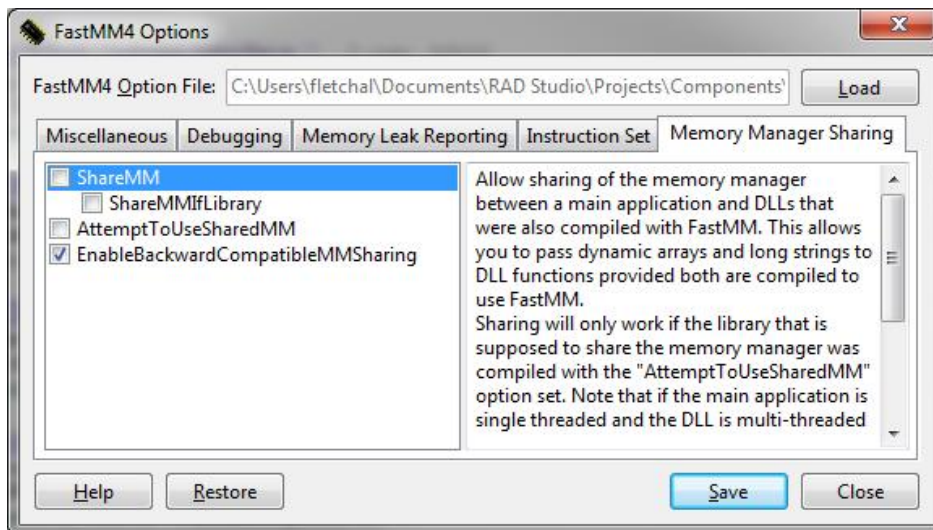
### 8. Patch BCB Terminate

To enable the patching for BCB to make uninstallation and leak reporting.

## What if I don't want to mess with FastMM4Options.inc file?

Of course, you don't have to edit the file manually. You can use *FastMM4 Options Interface*. However, you will not have access to the compilations options for debug and release version of your programs. Nevertheless, it is a very user friendly straightforward way of making changes to the default behaviour of your application.

Switches are grouped in different tabs and for each option a detailed explanation (from FastMM4Options.inc) is displayed on the right pane.



Once you start the program you need to load the appropriate Option file, make the desired changes. Save the changes and rebuild your application.

## Preparing and compiling projects in Delphi

There are a few simple steps to prepare your existing projects to use the full version of FastMM in a useful way.

1. You can control how FastMM behaves in two ways:
  - a. By making changes to Option Grouping in the FastMM4Options.inc so you can define different FastMM behaviors for your release version and debug version.
  - b. Or by using a conditional IFDEF statement to use the complete library on the debug version and ship the Delphi supplied library in the release version. To accomplish this, in Delphi open a project and add the unit FastMM4 as the first unit in the uses clause of the .dpr file.

```
...
uses
  {$IFDEF DEBUG}
    FastMM4,
  {$ENDIF}
  DB,
...
```

2. Optionally, set FastMM4 debug options using the *FastMM4 Options Interface* program. Make sure to build your program every time you make changes to any FastMM4 options.
3. In order for certain debug features of FastMM4 to work you must make sure that certain debug switches are turned on. The following is a list of recommended switches:
  - a. In the Compiler options set the following options
    - Debug Information
    - Reference Info
    - Use Debug DCUs
  - b. In the Linker options make sure that **one** of the following options is set
    - TD32 Debug info
    - Map file
4. After you run the application a log of the memory manager can be found in the same folder where the application ran. The log file is named *Leaks\_MemoryManager\_EventLog.txt*.
5. If your project includes EXEs and DLLs you also need to define ShareMM, ShareMMIfLibrary and AttemptToUseSharedMM using the *FastMM4 Options Interface* program and add FastMM4.pas to the top of the uses section of the .dpr for both the main application and the DLL and follow the directions outlined in item 3. This will allow FastMM to report memory leaks across EXEs and DLLs.

## What's next?

That's it! Now you are ready to detect memory errors like never before. Fire up your IDE and start plugging those leaks!

## About Alan Fletcher (<http://www.alanfletcher.org>)

I am a senior Software Developer with over 28 years of experience in the area. 16 of those years were mostly spent in Delphi/ Pascal. I am a Nokia Qt Champion. I have been involved (and fascinated) with computers since 48K of RAM was a lot and the source code for the OS was provided as part of the computer documentation. I have worked on software development in Brazil, Canada, Austria, France and the United States. I run the Pacific Northwest Delphi User Group (<http://www.pnwdelphi.org/>) the Qt Seattle chapter (<http://www.qtsea.org>)

# Give me some Latitude

by D.P. (Joe) Griffin

Some time ago I was working on a project where I created an application to provide the “glue” between a County Council’s Road Maintenance Management System (RMMS) and the handheld computers used by the gangs working on the roads. Using their high definition maps, the Council would pinpoint the work to be carried out by supplying Northings and Eastings – effectively the National Grid Reference (NGR) expressed in metres. Part of my task was to convert the Northings/Eastings into Latitude/Longitude values as used by the GPS built in to the handheld.

I already had the algorithm for the conversion from a Psion application, so it was easy to set up the conversion.

However, when we started to test the system, we found that the locations given by the handheld were out by a few tens of metres.

By coincidence, at about the same time, someone on the Cix Delphi conference asked for a routine to convert NGRs to Latitude/Longitude and was offered a routine which was “accurate to about 1.5m”. That sounded pretty good to me so I scrounged a copy, incorporated it into my application and received a big “thumbs up” from the field. At that point, I dumped the earlier routine as erroneous and forgot about it.

Just recently I was asked for a two-way latitude/longitude to Northing/Easting calculator and had to do a bit of research to go from Northing/Easting to the OS letter/number grid reference. That was when I discovered that latitude/longitude isn’t necessarily fixed!

When the national grid was re-triangulated in the period 1936 – 1962, the map datum was redefined based on the Airy 1830 ellipsoid. This gives a best fit to the UK and is known as the OSGB36 datum. Fast-forward half a century to the availability of GPS systems and a newer standard was introduced. The World Geodetic System 1984 standard (WGS84) gives a worldwide best fit and as such varies slightly from the OSGB36 standard. It is the WGS84 standard which has been adopted by the GPS systems.

The discrepancies between the two systems are not massive, but are significant when trying to work to the nearest metre. For example at the Greenwich Observatory, the WGS84 meridian lies 102.5 m to the east of the OSGB36 prime meridian. In Cornwall the WGS 84 longitude lines are about 70 metres east of their OSGB36 equivalents. (They match out in the North Atlantic!) The WGS84 latitude lines are about 70 m south of the OSGB36 lines in South Cornwall, the difference diminishing to zero in the Scottish Borders, and then increases to about 50 m north on the north coast of Scotland.

My original routine for converting to Latitude/Longitude wasn’t “wrong”; it was just converting using the OSGB36 datum, rather than the WGS84 datum used by the GPS system.

## About Joe Griffin

Having graduated in Mechanical Engineering, I spent the first half of my career working in Heat Transfer. In the late 1980s my employers ran away to the USA and I started a second career as a software developer. My company GerbilSoft Associates Ltd was formed in 1993, initially to work at what was then the London Air Traffic Control Centre (LATCC) at West Drayton. A number of systems were developed using Paradox for Windows, one of which was still in use until the LATCC site closed in 2008. Having had to resort to the use of Visual Basic for some tasks, I adopted Delphi with enthusiasm and have used it ever since.

# Delphi XE2 Cross-Platform using FireMonkey

by Bob Swart

Apart from 32- and 64-bit Windows support, Delphi XE2 also offers native Mac OS X as target platform. The IDE remains a 32-bit application, however, so running (launching) a Mac OS X application from the IDE will require some tinkering.

## Project Targets

Where previous Delphi projects had a special node in the Project Manager for the Build Configuration, Delphi XE2 introduces the Target Platform node. This can get a value of 32-bit Windows, 64-bit Windows, and OS X. Since the Delphi XE2 IDE itself is still a 32-bit Windows application, by default a new project will be created with the 32-bit Windows target. Depending on the project type, we can add one or more Target Platforms. A VCL project, for example, can be targeted for both 32-bit and 64-bit Windows, but not Mac OS X. The VCL is really tied to the Windows API, and is just about impossible to migrate to another platform. If you want to create an application for Mac OS X, you have to use either a console application (which can be handy at times, but may not be exactly what you have in mind when you think about a Mac application), or you have to create an application using a special cross-platform application framework called FireMonkey.

It's easy to add a 64-bit VCL target to a project, and we've seen how to create, run, debug, and deploy 64-bit applications with Delphi XE2 already. However, for Mac OS X applications, we have to use a different framework, called FireMonkey.

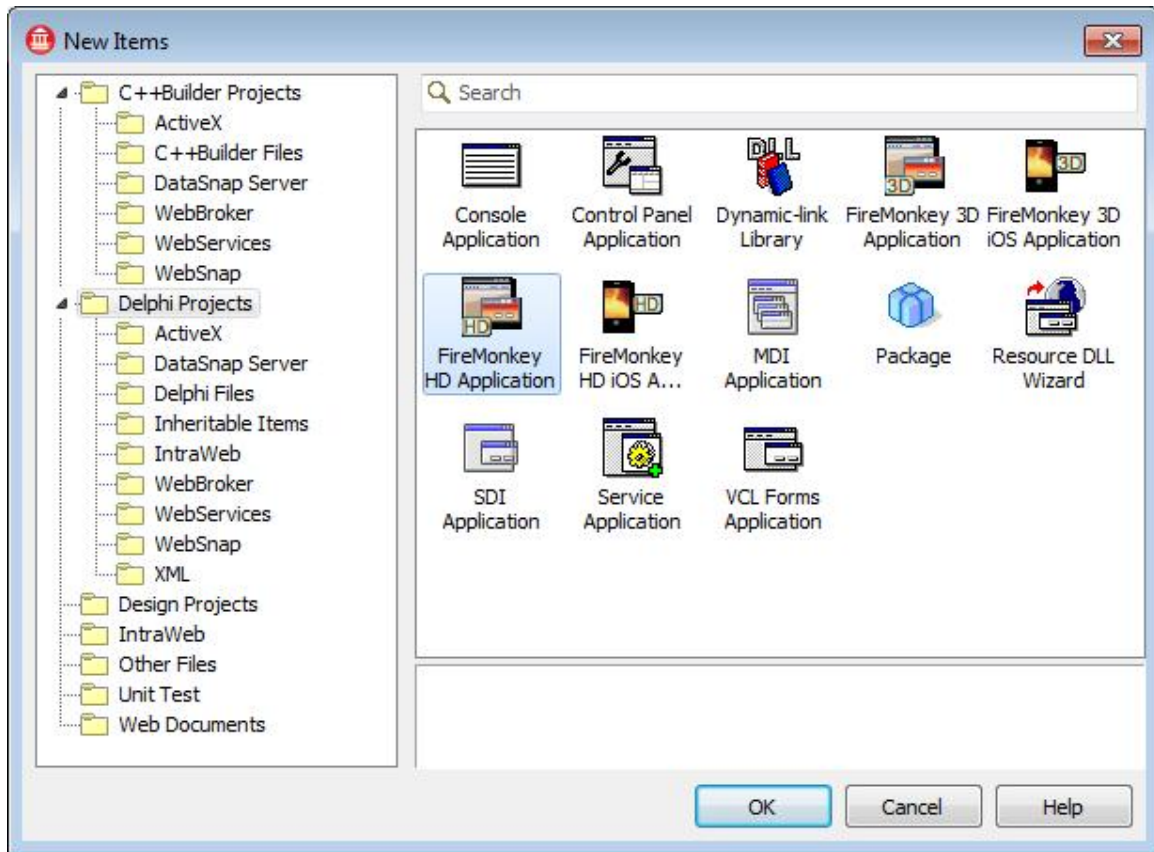
## FireMonkey

In order for an application to look like a Windows application on Windows, and a Mac application on Mac OS X, Embarcadero had to bring in (*read: buy*) a whole new set of visual components aka GUI framework. They actually call it the FireMonkey Application Platform, and it is specifically designed for cross-platform application development. Note that the VCL is not dead or being replaced. The VCL is still the best solution for native Windows applications (32-bit and/or 64-bit). But for cross-platform applications that need to move to other platforms like Mac OS X (and in the future probably Linux), we should use the FireMonkey Application Platform.

The FireMonkey Application Platform presents cross-platform controls and concepts like forms, dialogs, buttons, menus, etc. It supports 2D and 3D graphics and uses the GPU for the graphics, freeing the CPU itself for doing the "real work" (like calculations or database operations). The FireMonkey designer may feel a bit awkward at first, since it's not entirely the same as the VCL designer. However, remember that FireMonkey is at version 1.0, and there will be many enhancements (and fixes) in the time to come.

## FireMonkey Example

To demonstrate the FireMonkey Application Platform, let's create a new application using Delphi XE2. Using *File | New - Other*, we get into the Object Repository and can see a number of different FireMonkey project targets for Delphi:

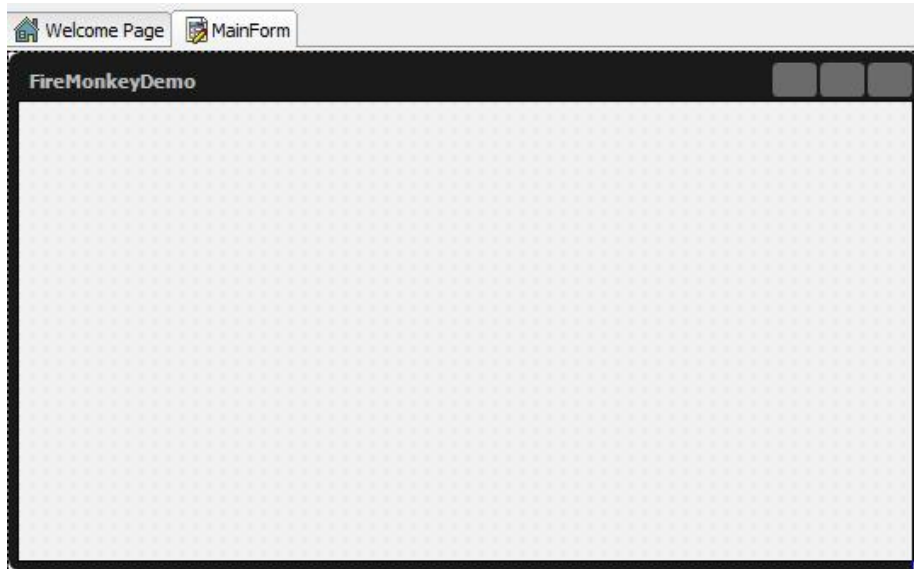


A FireMonkey 3D Application is capable of doing more graphic “stuff” than a FireMonkey HD Application. However, as a consequence, a FireMonkey 3D Application requires more power from the GPU. The FireMonkey HD Application is the one that most closely resembles a VCL Forms Application, with a 2D Form Designer that can host FireMonkey controls that almost look and feel like good-old VCL controls.

## HD Example

As an example, let’s create a FireMonkey HD Application for Delphi, which creates a new project starting with a new empty form with a black caption and border. Save the project in FireMonkeyDemo.dpr and the form in MainForm.pas (in case you want to play along, for example using the trial-edition of Delphi XE2).

One thing that you may immediately notice: if you click on the FireMonkey designer area for the form, you cannot do Alt+F to go to the File menu. Somehow, the Alt menu strokes are disabled here. Also, the nice IDE short-cut of *Edit | Copy* to copy the contents of the VCL Forms Designer is absent for the FireMonkey designer, so I had to make a screenshot the hard way, as can be seen in the following picture.



As you can see, this looks like a platform-independent window, with placeholders for the border icons (on the upper right, probably because the IDE is still running on Windows), and a black border and caption.

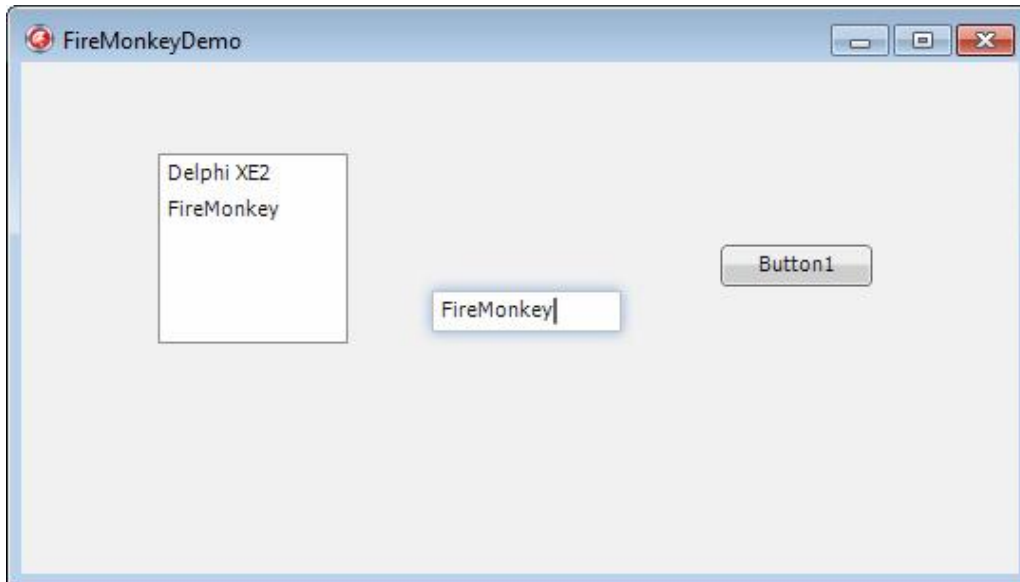
We can now look at the Tool Palette again for a list of FireMonkey controls that can be used. As I've mentioned before, the Standard category contains a number of familiar controls like TEdit, TListBox and TButton, which can be placed on the form. Warning: do not place them "on top of each other" (the effect that you get when you enter "Edit" in the Tool Palette search box and type <enter>, followed by "ListBox" and another <enter>. This will place the TListBox as child of the TEdit (unlike the way the VCL works), which is probably not what you want, and can lead to funny effects (if you move the TEdit, the TListBox will move along!).

Speaking of positioning the FireMonkey controls; this is another area of differences with the VCL. FireMonkey doesn't expose a Top or Left property, but uses a Position property with X and Y subproperties. There are no Anchors, but a Margin and a Padding property that appear to conflict with the way margins and paddings work in the VCL or CSS.

Anyway, after you've placed a TListBox, TEdit and a TButton on the TForm (at least the control names are the same, although they originate from different units), we can write the well-known event handler for the OnClick of the TButton:

```
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    ListBox1.Items.Add(Edit1.Text);  
end;
```

And then we can run the application, which by default will show up as a 32-bit Windows application (the default target platform of any project in Delphi XE2).



This sure looks and feels like a regular Windows application. In fact, it already contains some more-than-regular effects, like the “glow” around the TEdit that has the focus (visible in the screenshot). This effect can take up some processing power (GPU or CPU), and if the application becomes too slow, you can disable the effect as follows:

```
GlobalDisableFocusEffect := true;
```

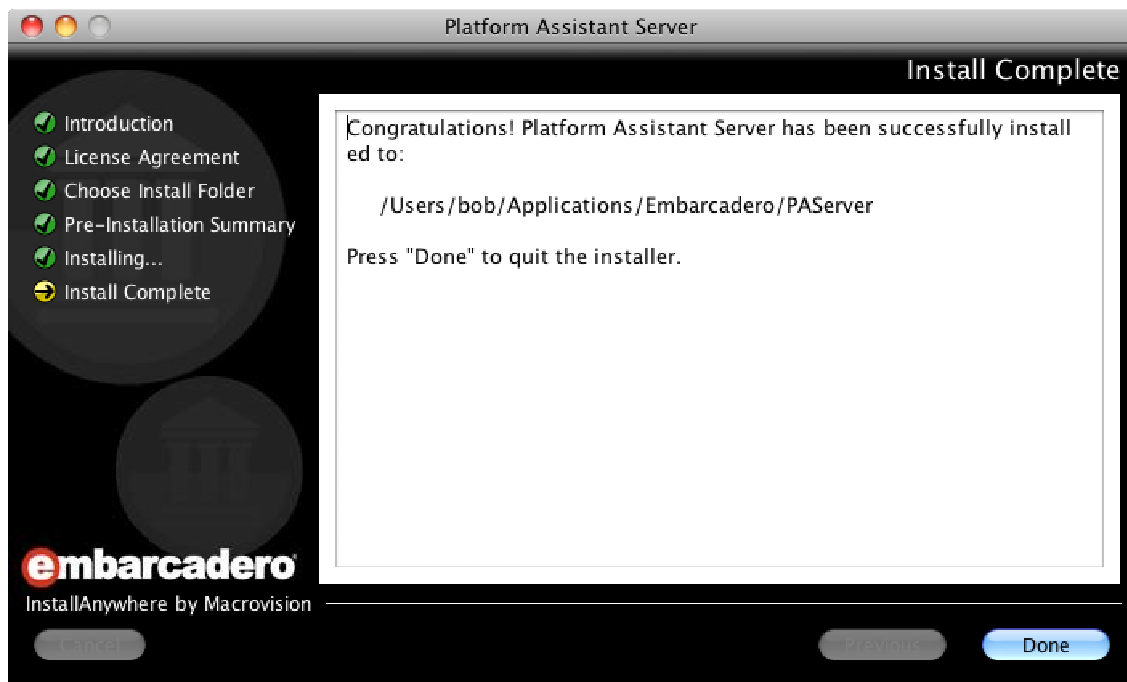
In order to run the same project, without any changes to the source code, on Mac OS X, we need to perform some additional steps.

## Mac OS X

First of all, you need a Mac running Mac OS X version 10.6 (Snow Leopard) or later. In order to deploy from your Windows development machine to the Mac target machine, you also need to install a separate utility called the Platform Assistant (paserver) on the Mac. The paserver can be found on your development machine in the C:\Program Files\Embarcadero\RAD Studio\9.0\PAServer directory. There is a setup\_paserver.zip to be used on Mac OS X, and a setup\_paserver.exe to be used on a 64-bit Windows machine. You need the setup\_paserver.zip and open it on the Mac where you need to unzip it and run the setup\_paserver application to launch the installer of the Platform Assistant Server application.



A number of screens need to be passed (leaving the default settings intact), after which the Platform Assistant Server is installed on the machine, in my case in the /Users/bob/Applications/Embarcadero/PAServer directory:



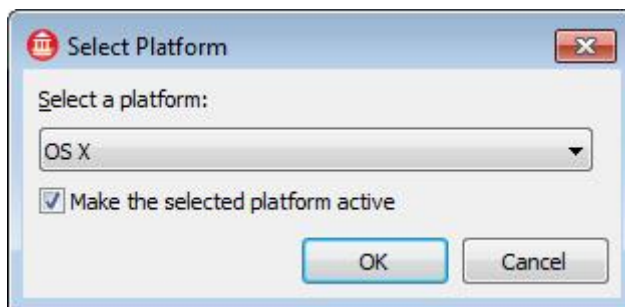
Now, as soon as you start the paserver application itself, it will start a terminal window and ask for a password. Don't panic, this is not a password that you should have remembered or written down somewhere. Rather, it's the password that you can define here (at the deployment machine) and that you have to specify at the development machine to allow the development machine access to the deployment machine (and avoid any other "visitors" to access the Mac at port 64211 as well).



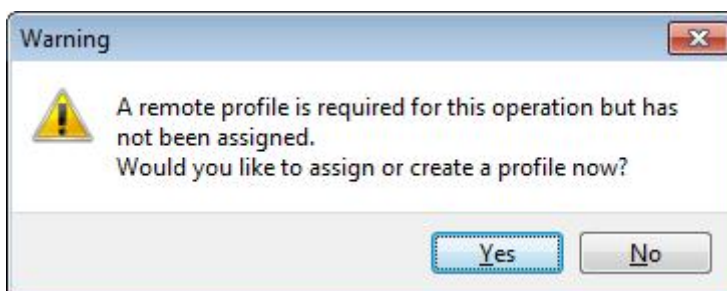
Once the password is entered, we can leave the paserver running at the Mac, and return to the Windows development machine to compile and run (with or without debugging) and/or deploy the project as a Mac OS X application.

## Mac OS X Development

The next step back at Delphi XE2 involves the addition of a new target platform. In the Project Manager, open the Target Platforms node to verify that it only contains the default 32-bit Windows target. Then, right-click on the Target Platforms node and select “Add Platform...” to add a new target platform. In the dialog that follows, select OS X as new target:

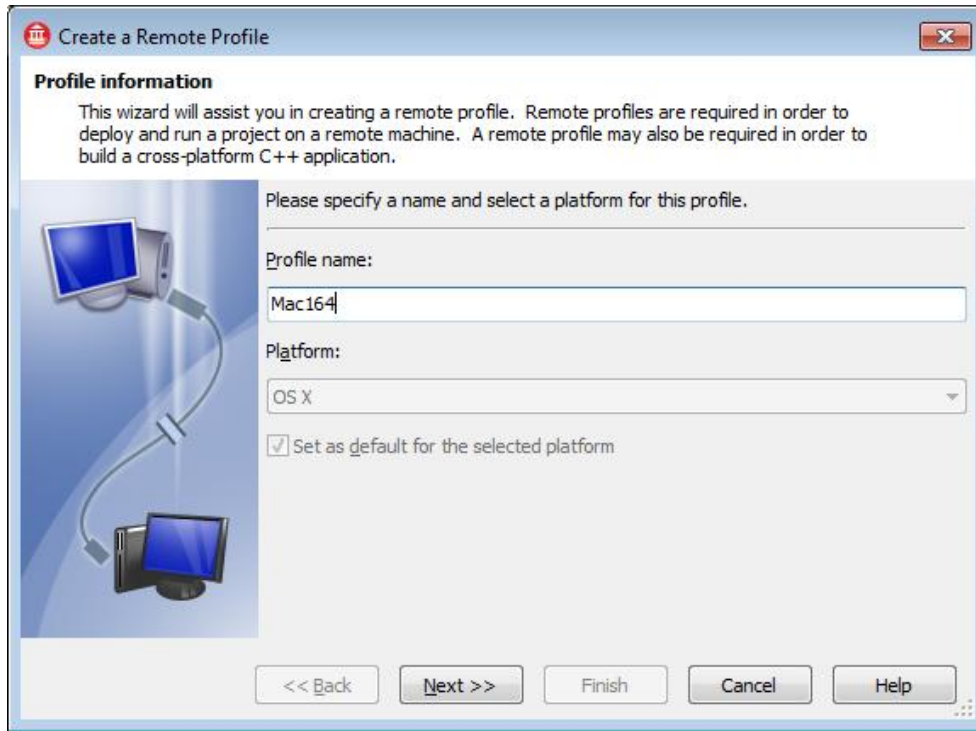


A Target Platform also needs a Remote Profile with the information to connect to the remote target machine. If a remote profile for the selected platform already exists, then Delphi XE2 will assign it as remote profile. Otherwise, you will be prompted to create a new remote profile as soon as you try to run the project for OS X.



If you click on next, a dialog will show up with all available Remote Profiles for the OS X Platform. Initially, this list will be empty, so you need to click on the “Add...” button to create a new Remote Profile, or you need to click on the “Import...” button to import a Remote Profile (in case you saved and exported one from another development machine for example).

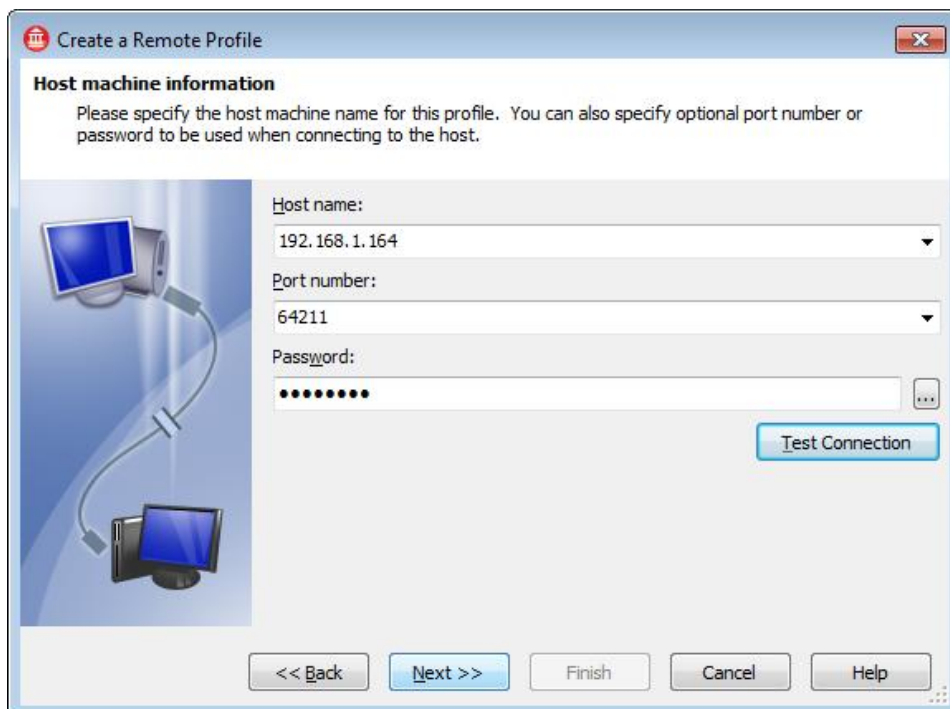
The Add button will display the “Create a Remote Profile” wizard, where we can specify the name of the remote profile. I typically call the Mac profiles “Mac” followed by the last part of the IP-address, so I know which Mac I’m talking about. Mac164 is the Mac mini running OS X Snow Leopard that we just installed the paserver on:



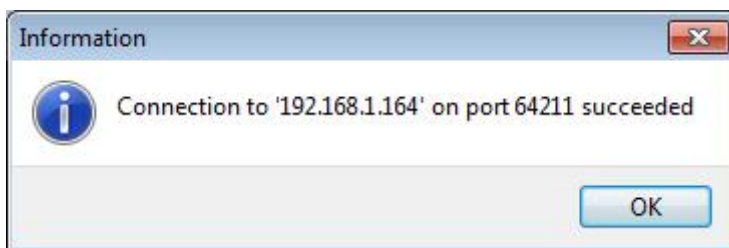
Note the checkbox to use this Remote Profile as default remote profile for the OS X platform, so next time you add a Target Platform for OS X, this Remote Profile will be assigned to it automatically.

On the next page, we can specify the Host Name of the Mac OS X machine (or the IP-address). The Port number is already specified using the default port number 64211. If you debug and deploy in your own local network, that port number is fine. However, If you plan to debug or deploy over the internet, I would change that port number to a slightly less obvious one, since only the password is keeping other “visitors” from connecting to your Mac and “deploying” applications to it.

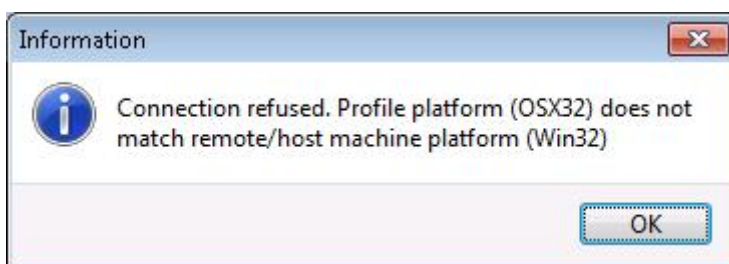
Finally, do not forget to specify the same password here as the one you specified in the paserver back on the Mac.



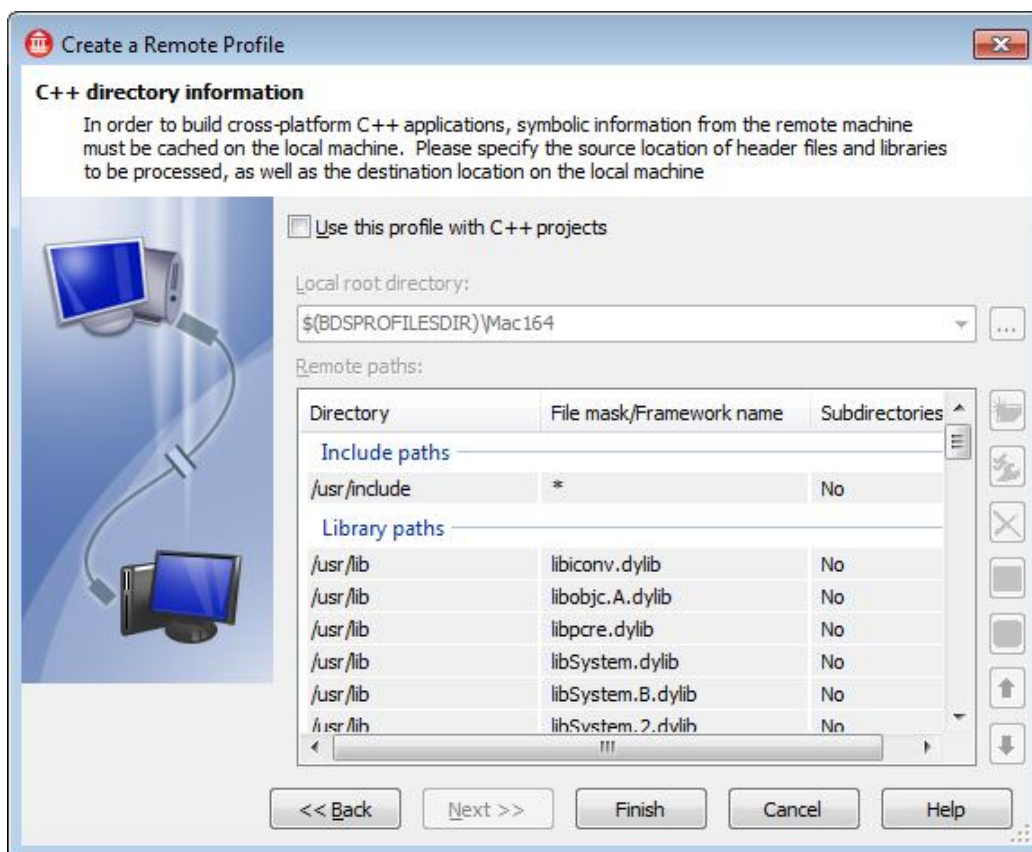
Click on the Test Connection button to ensure you can talk to the Mac. If the connection is refused, then you may have to configure the firewall on the Mac to allow the incoming connection.



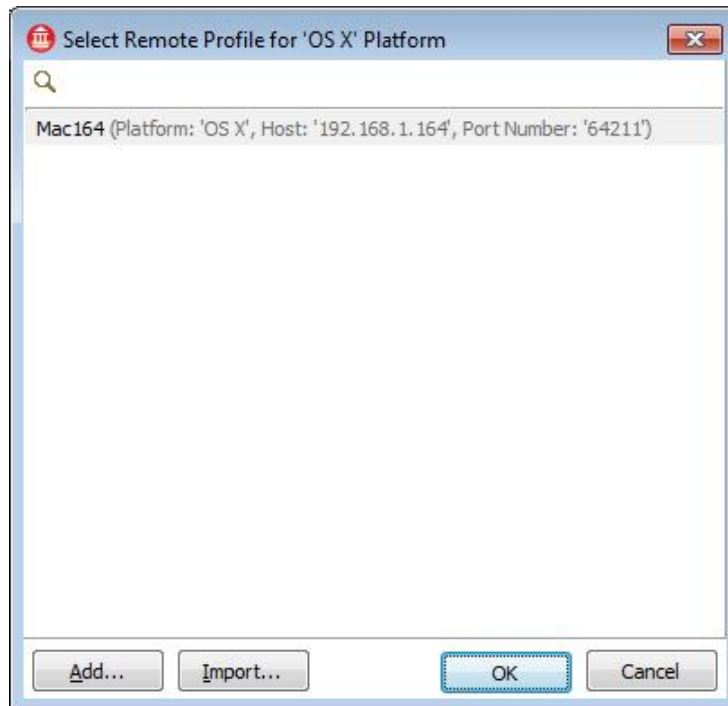
If you try to connect to a remote profile for a Win32 or Win64 machine, you'll get an error message:



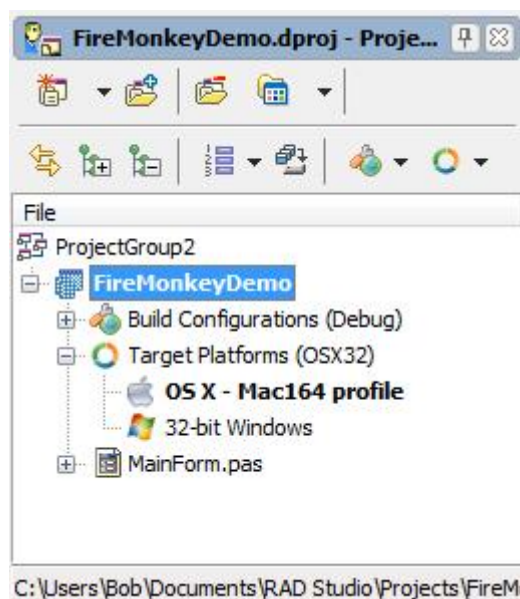
If you close this confirmation dialog and go to the next page in the Remote Profile Wizard, a page is shown which is only relevant for C++Builder developers (in which case you need to cache symbolic information from the remote machine on the local machine). Delphi developers can just skip that page and click on the Finish button to create the remote profile.



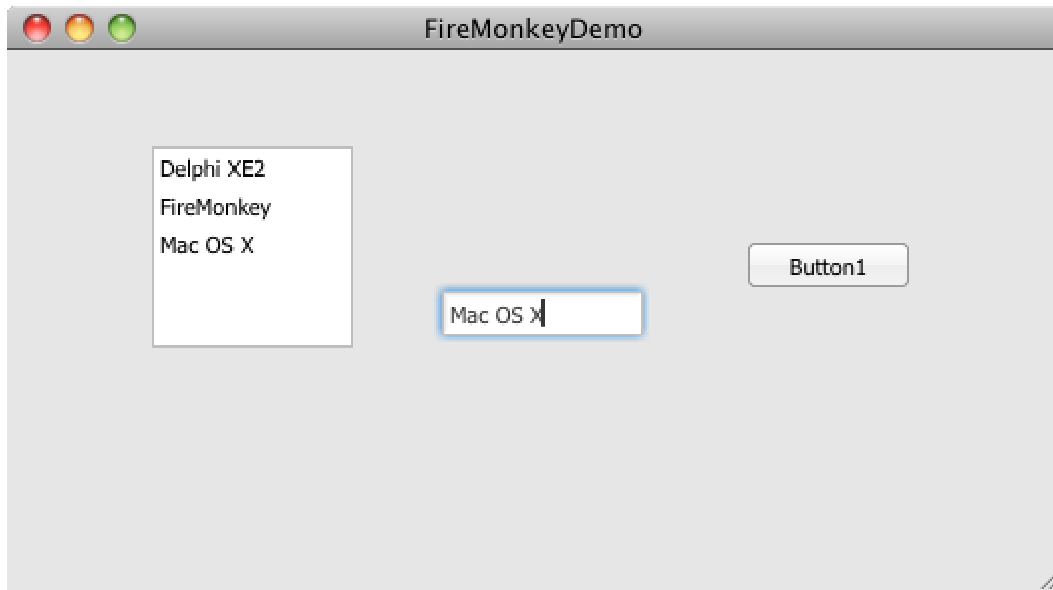
This will bring you back to the Select Remote Profile dialog, where you can now finally select the newly created Remote Profile.



As a result, the Project Manager will now show the Remote Profile next to the Target Platform in the Project Manager:



Also, if the Remote Profile selection was shown as a result of the initiative to “Run” the application, the actual application will now be running on your Mac OS X machine! This may not be clear at first, especially if you did Run | Run without Debugging, but if you switch back to the Mac, you’ll see the FireMonkey demo application running as a native Mac OS X application!



And this certainly looks like a native Mac OS X application to me. If you compare this screenshot to the Windows edition of the FireMonkey Demo, you will see the same ListBox, Edit and Button, but the actual look-and-feel is now totally different and Mac-like, while the Windows edition truly feels like a Windows application. Of course, these are just simple demo applications, but you should get the idea...

## Mac OS X Deployment

When it comes to deployment of your application, especially to the Mac running OS X, you need to perform a number of steps again. Using Project | Deployment you can get a Deployment tab in the IDE that shows the files that need to be deployed to the Mac OS X machine. There is one gotcha: if you compile a Release Build, then the project .rsm file will not end up in the OSX32\Release directory, so this file cannot be deployed. The Debug Build will produce the .rsm file, which causes it to be displayed in the list of deployment files:

Local Path	Local Name	Type	Platforms	Remote Path	Remote Name	Remote Status
<input checked="" type="checkbox"/> OSX32\Release\	FireMonkeyDemo.info.plist	ProjectOSXInfoPList	[OSX32]	Contents\	Info.plist	Same
<input checked="" type="checkbox"/> OSX32\Release\	FireMonkeyDemo.icns	ProjectOSXResource	[OSX32]	Contents\Resources\	FireMonkeyDemo.icns	Same
<input checked="" type="checkbox"/> \$(BDS)\Redist\osx32\	libcgunwind.1.0.dylib	DependencyModule	[OSX32]	Contents\MacOS\	libcgunwind.1.0.dylib	Same
<input checked="" type="checkbox"/> OSX32\Release\	FireMonkeyDemo	ProjectOutput	[OSX32]	Contents\MacOS\	FireMonkeyDemo	Same
<input checked="" type="checkbox"/> OSX32\Release\	FireMonkeyDemo.rsm	DebugSymbols	[OSX32]	Contents\MacOS\	FireMonkeyDemo.rsm	

If you connect to the deployment machine, you can see the Remote Status. The green arrow will actually deploy the project files. On my Mac, they end up in the /Users/bob/Application/Embarcadero/PAServer/scratch-dir/Bob-Mac164 where Bob is the name of the remote user (since the local user is called "bob") and Mac164 is the name of my Remote Profile. The FireMonkeyDemo is a 17.6 MB archive that contains all selected deployment files, and can be run as stand-alone application on the Mac. We can also copy it to another Mac (running at least Mac OS X Snow Leopard) and run it from there. If you copy it to a USB stick, you'll see a FireMonkeyDemo.app directory with a Contents subdirectory and a MacOS as well as a Resources directory inside plus the Info.plist file (see list of files above). The MacOS directory contains the FireMonkeyDemo, the FireMonkeyDemo.rsm and the libcgunwind.1.0.dylib library (only 20 KB), while the Resources directory contains the FireMonkeyDemo.icns file. I'm not sure if the .rsm file is still required after we've done a Release build, but that's something to figure out later. At least it's easy to deploy, and we can now even run it from the USB stick!

## FireMonkey Connectivity

Apart from the data-access components found in the dbExpress, dbGo and InterBase categories of the Tool Palette, we can also create FireMonkey applications that act as DataSnap Client or Web Service clients. Note that this only applies to FireMonkey applications that are compiled for Win32, Win64 or Mac OS X, and not the iOS targets.

Next time, I'll cover FireMonkey in combination with DataSnap, but now I will limit myself here to the use of SOAP for FireMonkey applications, leaving the data-access (and DataSnap Clients) to be covered later.

## FireMonkey and SOAP

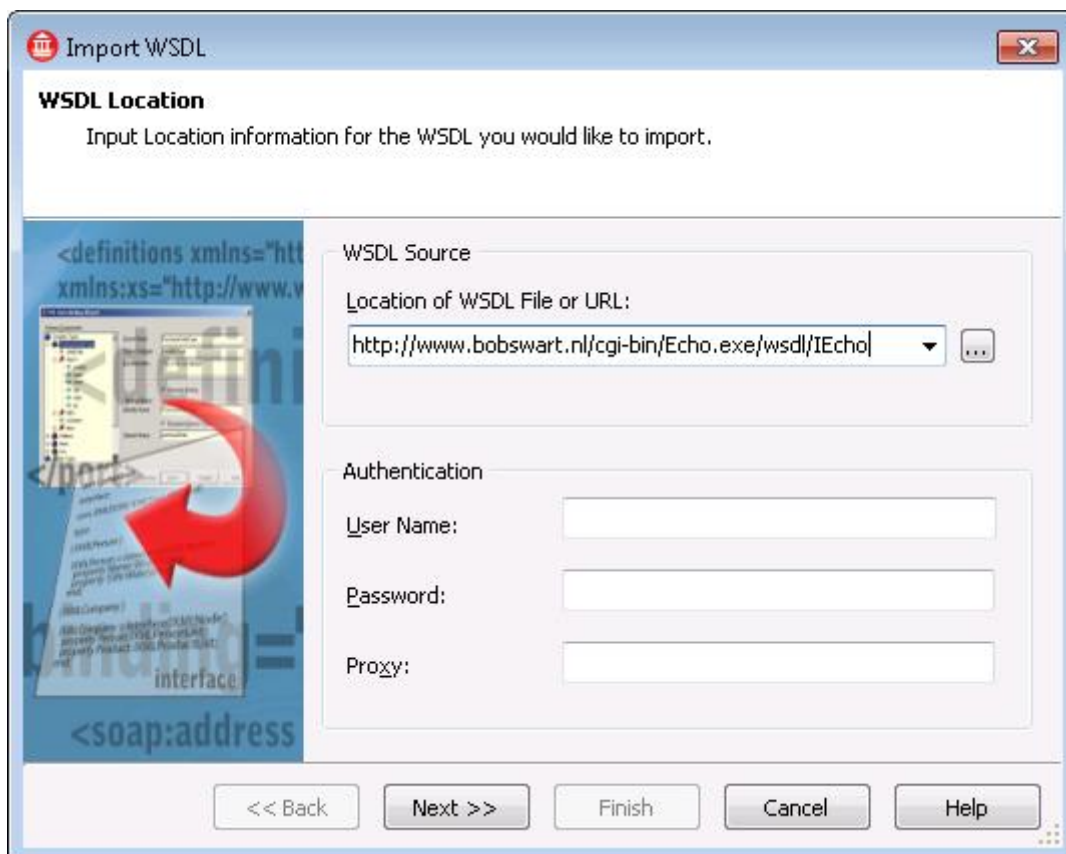
One way to allow FireMonkey client applications to connect to servers, is to use SOAP as the connectivity protocol. For this demonstration, we need a SOAP Web Service that can be accessed from both Windows and Mac OS X. I've used Delphi XE2 to produce a simple SOAP Web services as a CGI executable, and hosted it on my [www.bobswart.nl](http://www.bobswart.nl) domain as <http://www.bobswart.nl/cgi-bin/Echo.exe>

Feel free to use this SOAP Web Service for your own tests with FireMonkey (or VCL) SOAP clients. The WSDL URL that defines the formal SOAP service can be found at <http://www.bobswart.nl/cgi-bin/Echo.exe/wsdl/IEcho> and will remain available just as the SOAP Web Service itself.

## SOAP Client

For this demo, I'm using a FireMonkey HD Application, saved in project SoapClient with FireMonkey HD Form ClientForm. In order to consume the web service, we must import the WSDL using the WSDL Importer from the WebServices category in the Object Repository.

In this dialog, we must specify the WSDL URL, as follows:



Click in Next, make sure the Automatic SOAP versioning is still selected, then again on Next, and finally on Finish. If you want to learn more about the different WSDL Import Options, check the online help or my SOAP & Web Services courseware manual for more details.

After a click on the Finish button, you'll get an import unit called IEcho1, with the definition of the IEcho web service interface:

```
IEcho = interface(IInvokable)
[ '{6CE4129F-B1EE-1621-3D51-13A71B1998D8}' ]
function echoEnum(const Value: TEnumTest): TEnumTest; stdcall;
function echoDoubleArray(const Value: TDoubleArray): TDoubleArray;
stdcall;
function echoMyEmployee(const Value: TMyEmployee): TMyEmployee;
stdcall;
function echoDouble(const Value: Double): Double; stdcall;
end;
```

The TEnumTest, TDoubleArray and TMyEmployee types are defined earlier in the same import unit, as follows:

```
type
TEnumTest = (etNone, etAFew, etSome, etALot);

{ $SCOPEENUMS OFF }
TDoubleArray = array of Double;

// ***** //
// XML      : TMyEmployee, global, <complexType>
// Namespace : urn:EchoIntf
// ***** //
TMyEmployee = class(TRemotable)
private
  FLastName: string;
  FFirstName: string;
  FSalary: Double;
published
  property LastName: string read FLastName write FLastName;
  property FirstName: string read FFirstName write FFirstName;
  property Salary: Double read FSalary write FSalary;
end;
```

In order to “get our hands” on the IEcho Web Service interface, there is a special function declared and implemented in the IEcho1 import unit, called GetIEcho. We can pass three optional parameters: UseWSDL (by default False), the Address (of either the WSDL or the SOAP action), and a HTTPRIO component. See my SOAP and Web Services courseware manuals for more details, but rest assured that we can just leave them empty to call the IEcho web service directly.

On the FireMonkey form, place a TButton and implement the OnClick event handler as follows to grab the IEcho interface, call a web method, and display the results:

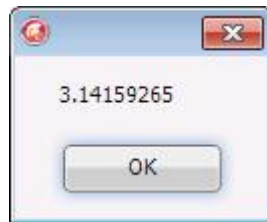
```
procedure TFormFMX.Button1Click(Sender: TObject);
var
  WS: IEcho;
begin
  WS := GetIEcho;
  ShowMessage(FloatToStr(WS.echoDouble(3.14159265)));
end;
```

The result is the echo of the double value, which will unfortunately be unreadable in the FireMonkey ShowMessage:



In order to fix that particular issue, we need to add a space to the string, as follows:

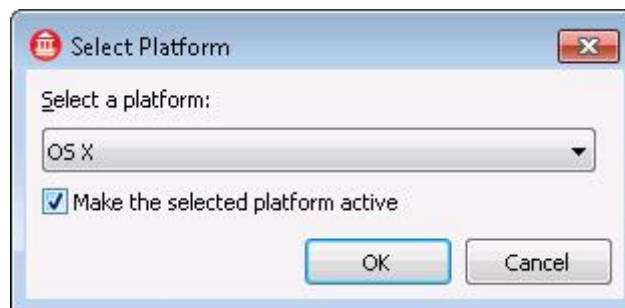
```
ShowMessage(FloatToStr(WS.echoDouble(3.14159265)) + #32);
```



## Deployment to Mac OS X

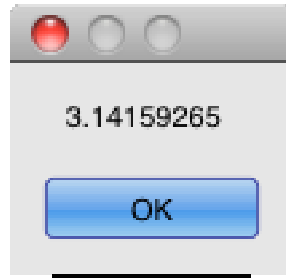
Once the FireMonkey SOAP Client has been tested on Windows (either or both Win32 and Win64), we can add the Mac OS X target platform, and test it on a Mac. To realize this, we need to right-click on the Target Platform node in the Project Manager, and select "Add Platform".

In the dialog that follows, select the OS X target:



Then click on OK. If you haven't associated a default Remote Profile to the OS X target platform, you must now either select or create a new remote profile (see half a dozen pages back).

Deployment - using Project | Deployment - shows that no additional libraries are needed (apart from the libcgwind.1.0.dylib). And if we run the application, we'll see the same output on a Mac OS X client:



All other Web Service methods and types can be used on the Mac running OS X as well. Note that Indy is used for the communication at the Mac side. And while Indy had been made compatible with Mac OS X, it does not work on iOS. So the same solution will not work in a FireMonkey iOS application, unfortunately.

## Summary

In this article, I've demonstrated how we can use Delphi XE2 and FireMonkey to design, run and even debug an application to Mac OS X. I've also discussed deployment details and the connectivity of FireMonkey to SOAP Web Services. Next time, I'll continue with the combination of FireMonkey and DataSnap.

This article is an excerpt from the first edition of my Delphi XE2 Development Essentials courseware manual, available in PDF format with free updates and e-mail support from <http://www.ebob42.com/courseware> (and free for everyone who buys Delphi XE2 or RAD Studio XE2 from me).

