Cross-platform Development the FireMonkey way

By Alan Fletcher

What is FireMonkey?

FireMonkey is cross-platform Framework developed by Embarcadero. FireMonkey was originally designed by Eugene Kryukov in the company "KSDev" as VGScene.

In 2011 Embarcadero acquired the rights to the software and renamed it to FireMonkey.

FireMonkey is included, along with the traditional Visual Component Library (VCL) in Delphi and C++ Builder.

FireMonkey was introduced in XE2. It's goal is to allow developers to design cross-platform applications and interfaces that take advantage of the acceleration features available in Direct2D on Windows Vista and Windows 7, OpenGL on Mac OS X, OpenGL ES on iOS, and GDI+ on Windows platforms where Direct2D is not available

Applications and interfaces developed with FireMonkey are separated into two categories HD and 3D. HD and 3D elements can be mixed by utilizing built-in components that are included in the IDE.

HD applications are 2D applications with flat interfaces similar to software that is developed using VCL. 3D applications are 3D applications and feature an three dimensional XYZ interface.

Firemonkey is a full software development framework, and retains many features available with VCL. The major differences are:

- Cross-platform compatibility
- Vector drawn interface elements
- Any visual component can be a child of any other visual component allowing for creation of hybrid components
- Built-in styling support
- Support for visual effects (such as Glow, Inner Glow, Blur for example) and animation of visual components

Due to the framework being cross-platform compatible, the same source code can be used to deploy to the various platforms it supports. Originally, FireMonkey natively supported 32-bit and 64-bit executables on Windows and 32-bit executables on Mac OS X and iOS.

As of the release of XE3, iOS support has been dropped, but it is still possible to develop iOS applications using XE2 editions of the same products. FireMonkey 2/FM² is the name of the framework in XE3, and though it provides similar features to what was shipped with XE2, there have been numerous improvements in many areas of the framework.

As of this writing (December 2012) the Embarcadero R&D team is working on iOS and Android support.

Windows 8 ARM and Linux server are targeted for the second half of 2013.

Are there alternatives to FireMonkey?

Yes! Certainly. There are even Pascal based alternatives. FPC and Lazarus are cross platform options as well as Qt (C++) wxWidgets(C++).

- **FPC/Lazarus**
  - **"**Free Pascal (aka FPK Pascal) is a 32 and 64 bit professional Pascal compiler. It can target multiple processor architectures: Intel x86, AMD64/x86-64, PowerPC, PowerPC64, SPARC, and ARM. Supported operating systems include Linux, FreeBSD, Haiku, Mac OS X/iOS/Darwin, DOS, Win32, Win64, WinCE, OS/2, MorphOS, Nintendo GBA, Nintendo DS, and Nintendo Wii. Additionally, JVM, MIPS (big and little endian variants) and Motorola 68k architecture targets are available in the development versions.**"**
- **Qt**
  - "Qt is a cross-platform application and UI framework for developers using C++ or QML, a CSS & JavaScript like language. Qt Creator is the supporting Qt IDE."
- **Mono**
  - "Mono is a software platform designed to allow developers to easily create cross platform applications. Sponsored by Xamarin, Mono is an open source implementation of Microsoft's .NET Framework based on the ECMA standards for C# and the Common Language Runtime."
- **wxWidgets**
  - "wxWidgets is a C++ library that lets developers create applications for Windows, OS X, Linux and UNIX on 32-bit and 64-bit architectures as well as several mobile platforms including Windows Mobile, iPhone SDK and embedded GTK+. It has popular language bindings for Python, Perl, Ruby and many other languages. Unlike other cross-platform toolkits, wxWidgets gives its

applications a truly native look and feel because it uses the platform's native API rather than emulating the GUI. It's also extensive, free, open-source and mature."

Why FireMonkey?

From a Delphi developer perspective FireMonkey is an interesting alternative to developing cross-platform solutions because it allows for leveraging on existing knowledge and concepts from the VCL and the language.

Another less often mentioned reason is the fact that Embarcadero is a software development company and as such is hardware and platform neutral.

To better understand the previous statement let's take a look at what happened to Qt. Qt development was started in 1994 by Trolltech, a Norwegian software company, as a way to developing cross-platform applications, including mobile platforms. In mid 2008 Nokia acquired Trolltech and imposed a new development strategy focusing mostly on it's own hardware and Symbian mobile OS. Lots of progress were done in the mobile environment. This was done in detriment of all other platforms. Then in early 2011 Nokia announced it was dropping the Symbian OS and as a side causality the Qt Framework that was sold off to Digia. As a result of that Qt is lagging on several areas of the mobile segment.

FireMonkey versions

FireMonkey was introduced in Delphi XE2. In 2012 a new version, FireMonkey 2 or FM2, was shipped with Delphi XE3. From this point onwards we are going to concentrate on what is new in FM2.

VCL vs. FMX - *a quick introduction to FMX from a VCL connoisseur*

Visual Component Library or VCL is a Windows only framework and can not be used in FireMonkey. FireMonkey has introduced it's on visual library named FMX.

FMX is compatible with Windows, Mac and soon IOS, Android, Linux and Windows ARM.

FMX and VCL share some common ancestry. Both object models start with a TObject that descends to a TPersistent and then to a TComponent. After TComponent the libraries diverge. FMX goes to TFMXObject,TControl and then to TStyleControl or TShape.

TStyledControl is used as a basis for all the visual components and is the base class for customizable and user-interaction controls.

TShape is the base class for 2D primitives. TShape defines the common behavior--methods and properties--for 2D graphic primitives and it cannot be used as stand alone component.

The event model remains the same between VCL and FMX. So, when it comes to events anything that applies to the VCL will apply to FMX. The same can be said about object persistence. FMX and VCL share TPersistent. As you already may know TPersistent is the ancestor for all objects that have assignment and streaming capabilities.

In a VCL application your component can broadcast messages to all the controls in a form, send messages to a particular control (or to the application itself), or even send messages to itself. FMX does not support component messages in the same way the VCL does.

The coordinate system is different in FM2. While the VCL uses left and top FMX uses X, Y and Z. Left and Top are integers and X,Y and Z are floating point. This change was brought into FMX out of the need to address 3D positioning on the form.

And speaking of properties, we need to be aware some properties have changed. For example: Caption is now Text, Left and Top are now Position.X, Position.Y and Position.Z

Object Ownership mechanism remains the same across the two libraries and Object parenting is similar. The difference is that FMX does not restrict parenting to container like controls like the VCL and Child Objects share attributes from it's parents.

In FMX TCanvas is not a direct device wrapper as it is in VCL

As in the VCL the FMX.TControl is the base class for on-screen components. However, in FMX subclasses are divided into primitive shapes (TShape) and styleable controls (TStyledControl). FMX.TControl extends TFmxObject to present objects that are seen, by adding properties and methods for size, position, margins, alignment, visibility, mouse and keyboard interaction, focus, animations, effects and painting.

What is new in FM2

While some changes were brought in to improve performance other changes were clearly targeting enhanced crossed platform including mobile devices. One can clearly see this pattern in the new items introduced and to some extent on the changes made to the framework.

FM2 brings framework refinements, a new Multimedia components, a new Layout components, a new Platform Services class, Styled Non-Client areas, Actions, Anchors, Sensors, Touch and Gestures. It also has enhanced Styles, 3D

Unneeded properties where prevented from being surfaced everywhere. That speeds up loading and saving form info at design time and loading of forms at run time

Bitmap performance enhancements where brought in FM2. FM2 switches to native bitmap as soon as possible. That means that bitmaps are moved into the GPU's memory. This brings the side effect of not allowing direct access to bitmaps. It is possible to map pixel data to a buffer and push changes to the GPU.

FM2 now offers support to capturing data from any capture devices. For that you can use TCaptureDevice and TCaptureDeviceManager. A new Multimedia wrapper was introduced to allow playing of media files. The TMedia, TMediaPalyer and TMediaPlayer control wrap around the host OS native multimedia system.

With cross-platform development come some unique challenges. Specially if you throw into the mix some smaller devices such as phones and tablets. So, some changes were introduced to allow for better screen layout management.

Every VCL programmer knows the worth of components Anchors. Anchors were missing from the first installment of FireMonkey. But never fear, Anchors were introduced on FM2 along with some welcome layout managers; TFlowLayout and TGridLayout.

Anchors were introduced to work together with Layout Managers

TFlowLayout arranges components as if they were words in a paragraph. It allows the developer to select spacing between components, component alignment and even forced breaks using a TFlowLayoutBreak.

TGridLayout allows controls to be arranged in a grid of equally sized cells. This layout manager rearranges objects in it's grid every time the layout changes. The components inside the a TGridLayout are resized to fit the sizes of the cells. Controls can be arranged in vertical or horizontal cells.

FM2 obsoleted TPlatform as a means to find information about supported features on a host OS because it was too rigid, desktop centered and did not adapt well to targeting diverse software and hardware platforms with different/disparate services.

In it's place TPlatformServices (FMX.Platform) was introduced.TPlatformServices can be used to to dynamically figure out what is available. This is a registry class that can be queried and uses *Supports* syntax. It also allows the programmer to easily implement custom devices and services.

FM2 introduced Touch and Gestures. This feature is modeled after the VCL Gesture engine. The are a few differences between the VCL implementation and the FMX implementation.

FireMonkey does not support fewer interactive gestures on Mac OS X than the number of supported gestures on a Windows PC. In the Mac only igZoom, igPan and igRotate are supported. FireMonkey does not support custom Gestures. Mouse gestures only work on Windows 7 and Windows 8. And on Windows interactive gestures and standard gestures cannot be used at the same time. Also FireMonkey adds TouchTargetExpansion which allows for expanded touch target around a control by adding a specified zone to be behave as if the user had touched the control itself.

FMX has introduced non-visual components that implements location and motion sensors. A great way of looking at what is coming in the sensors framework is to examine the unit System.Sensors. *(One way to test location in Windows without a location device attached to your computer is to use Geosense for Windows (http://geosenseforwindows.com). Geosense is a free software driven location sensor for Windows.)*

Speaking of units, any unit that starts with “FMX.” is a FMX unit only. However, units such as System.Sensors are framework agnostic. So, based on this statement, sensors components can be used in both VCL and FMX applications

Cross-platform programming

Last but not least, we need to talk about some best practices in cross-platform programming.

Always think cross-platform. Up to now our deployment OS was Windows and our way of thinking was base of the Windows programming model. Depending on your target(s) certain services may not available. For example, a desktop will certainly have a mouse, however a phone/tablet will be just the opposite.

Same recommendation goes for resources such as storage, connectivity, CPU power, battery, screen size, and so on. While on the desktop environment such resources are virtually endless in a mobile scenario most resources are limited and at times not available at all.

And while a desktop computer is a generic device a smartphone is a specialized device and as such it has one primary function that overrides any other function - the ability to receive and place calls at will. So programs need to be able to deal with such interruptions in a graceful manner.

One must program accordingly to such limitations and specific functions. And in some cases program to the lowest common denominator.

Unless you have a very good reason, prefer a feature that is implemented thru the framework as opposed to natively. Most often there is no valid reason to do the opposite. Let the framework do it's job and abstract you from the OS. That will buy you compatibility with newer platforms that come in to the framework.

If you must use a platform specific functionality make sure you document why yu are doing so and provide implementations to all platforms that your project is targeting. Also, provide an easy way to warn others that that specific feature was not supported by your implementation either at compile time or at run-time on any other platform.

A good way to implement this is to surround the implementation with a set of conditional pre-compiler directives as demonstrated in the code snippet below:

```
{$IFDEF MSWINDOWS}
  uses Winapi.Windows;
{$ELSE IFDEF MACOS}
  uses Macapi.Mach;
{$ELSE}
  {$MESSAGE FATAL 'Feature not implemented.!}
{$ENDIF}
```

Final thoughts

FireMonkey is a serious contender for the seasoned Delphi/VCL developer. It has significantly improved on it's second version. Furthermore, Embarcadero's product development strategy is pushing the product into a very desirable position. That position is the ability to develop once and deploy across many platforms. The iOS version is imminent, followed soon by the Android version, and later by the Windows ARM and Linux version. From where I stand I can see an exciting future ahead.

Resources

http://docwiki.embarcadero.com/RADStudio/XE3/en/Main_Page

http://www.youtube.com/user/EmbarcaderoTechNet

https://forums.embarcadero.com

http://stackoverflow.com/questions/tagged/firemonkey

About the Author

Mr. Fletcher is a Sr. Software Developer with over 28 years of experience in the area. 16 of those years were mostly spent in Delphi/ Pascal – his language of choice. Mr. Fletcher has 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. he has also developed in Assembler, C, C++, PHP, and other languages.
Mr. Fletcher worked on software development in Brazil, Canada, Austria, France and the United States. Currently Mr. Fletcher calls Seattle home where he works for WideOrbit and runs the Pacific Northwest Delphi User Group (http://www.pnwdelphi.org/). When Mr. Fletcher is not at the computer he loves spending time with his family, cycling, cooking, teaching spin classes and taking pictures.