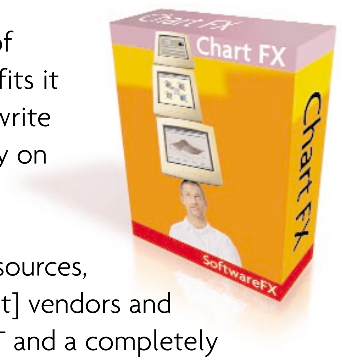
SoftwareFX

Any Chart, Anywhere!

A Component's Road to .NET

Software FX

Chart FX for .NET



Now that Microsoft has released Visual Studio .NET, many of you have started to take a closer look at .NET and the benefits it brings to the development process. I was given the task to write about .NET and its impact on components; more specifically on Software FX's leading charting solution, Chart FX.

While there's a wealth of .NET information and technical resources, little has been said about components and how [component] vendors and their customers will be affected by the introduction of .NET and a completely new execution environment.

My first goal is to assist a Chart FX potential customer in making an informed comparison between COM and .NET components. I found that Microsoft has made a considerable investment to ensure that the CLR-2-COM interoperability layer works as smoothly as possible, making it even harder to determine if you should get a .NET component or let your app run unmanaged code (e.g. ActiveX) under a .NET wrapper.

During this process, I have learned that embarking on .NET without considering how components will affect the migration process can prove detrimental in the long run. For example, you will find out that many of the security and deployment advantages you get from porting your apps to .NET will be lost as soon as you use an unmanaged component.

Also, there's no doubt the .NET hype creates an excellent marketing opportunity for component vendors — most of which have rushed to announce their .NET offerings. However, the pressure to generate immediate revenue on .NET upgrades has negatively influenced and dampened the introduction of more powerful components that can take full advantage of the shared class libraries built into the CLR.

Whether you are actually engaged in .NET development or you are simply looking into the migration issues associated with this technology, I hope you find the information and the insights contained in this document useful. One thing is for sure; if you're involved in Windows development, don't question IF, but rather WHEN you will commit to .NET.

.NET/COM Interoperability Basics

To understand how .NET impacts a COM component, I would like to refer to some basic interoperability concepts between .NET and COM.

.NET runs within a new execution environment called the CLR (Common Language Runtime). Code executing under the control of the CLR is called managed code. Conversely, code that runs outside the runtime is called unmanaged code. COM components, ActiveX interfaces, and Win32 API functions are examples of unmanaged code.

To simplify interoperation between the .NET Framework and unmanaged code and to ease the migration path, the common language runtime conceals from both clients and servers the differences in these object models. At runtime, the common language runtime automatically marshals data between COM objects and .NET objects as needed; creating what is called a Runtime Callable Wrapper or a COM Callable Wrapper, depending on the direction of the call.

Microsoft's rationale behind .NET and COM interoperability is simple— 99.9% of developers have legacy COM code. In other words, no developer will start a .NET project from scratch, they'll need to determine which tier is to be migrated first. In the meantime, a strong interoperability layer must exist and be provided for the remaining layers that have not yet been migrated to .NET.

Luckily for you, if components were central to your initial development strategy, migrating this functionality should be as simple as expecting a .NET version of your preferred controls. In this sense, component vendors act as a development arm of your organization.

How will component vendors approach the migration to .NET?

As a component vendor, our first option to make a COM component work under .NET is by generating a PIA (Primary Interop Assembly) against a COM object's type library using the *tlbimp* utility provided with .NET, creating what's called a Runtime Callable Wrapper. This RCW, by default, exposes the same interfaces (and by definition the same properties and methods) as your existing component.

This approach is obviously the quickest way to migrate COM components into .NET. However, when it comes to components beware of the high price you'll end up paying by forfeiting some of the most interesting benefits .NET has to offer. My suggestion is that you stay away from those vendors trying to convince you that you can simply use tlbimp to make their components work in .NET

In the meantime, other vendors will concentrate on writing a Custom Managed Wrapper, which is no more than coding portions in .NET while keeping tasks that perform a significant amount of work internally in COM.

For components like charts, grids, reports and other design oriented controls [which provide a considerable amount of customization capabilities to the developer and thus a significant amount of properties] custom managed wrappers repartition of code minimizes the interop call overhead. However, you'd still be facing deployment and other interoperability problems without getting a significant benefit for translating part of the COM code to .NET.

As a rule of thumb, if your development language provides a native component architecture, you must avoid unmanaged code whenever possible. Unmanaged classes are not ideal and the reason is simple: *Translation*. For example, wrappers in .NET must translate their native types into COM types, which not only takes time, but everyone knows how awkward real-time human language translation can become.

Finally, component vendors have the option of rewriting their components entirely in C# and porting their components to run fully managed within the CLR. I'm convinced that vendors who are truly committed to providing quality products to the development community will recode their products and provide a solid .NET offering. These are the ones you should keep!

What benefits will .NET bring to a product like Chart FX?

For us, porting Chart FX to a new language like C# is a time consuming and costly task. However, it will provide immediate benefits to users developing .NET applications. I would like to group these benefits into 2 categories: intrinsic and progressive.

By intrinsic benefits, I mean all those inherent features that come as a part of a technology like .NET and that does not require extra development beyond the language recoding. In other words, these benefits address some of the problems mainly attributed to the component specification or OS rather than a particular component or vendor. For example, code portability, deployment, side-by-side versioning and security are issues that have always troubled developers and that were elegantly addressed in the .NET framework.

Progressive benefits are features that did not exist in a product before, whether they are related to .NET or not. Some of these benefits will be discussed later in this document. For now, I would like to quickly mention some of the intrinsic benefits that .NET will bring to a product like Chart FX.

The .NET framework will influence Chart FX in the following areas:

Deployment & Distribution Layer

Consider the Chart FX case, a full featured charting solution that helps thousands of developers in need of integrating charts into their applications that will ultimately reach millions of users' worldwide. Deployment advantages rank at the top of the list among the most valuable intrinsic benefits of .NET components. Some of these are:

- No-impact components, which provides application isolation and eliminates DLL hell. Components do not affect other applications.
- Private components by default. Components are deployed to the application directory by default and are only visible to the containing application.
- Side-by-side versioning. Multiple versions of a component can co-exist, you can choose which versions to use, and versioning policy is enforced by the common language runtime.
- XCOPY deployment and replication. Self-described and self-contained components can be deployed without registry entries or dependencies.
- On-the-fly updates for components without the need to restart IIS (important for production servers)
- Ease of Enterprise deployment. With .NET there is no need to deploy an application to the end user's desktop. Instead, a user can invoke the application simply by typing a URL in a browser. The application will download to the client machine, run in a secure execution environment, and remove itself upon completion

API Layer

.NET standardizes on a universal set of types that are shared across all managed languages. This means developers can quickly leverage their knowledge in other platforms. In other words, everything you use from .NET works the same way and exposes a consistent programming interface. Previously, if you programmed in Visual Basic you used one API , while C programmers used the Win32 API, C++ used MFC and web developers used ASP. No more. One API is sufficient to allow a programmer to write applications in virtually any language they choose.
Consider that Chart FX is a charting solution that serves many platforms and this creates a competitive advantage over its COM predecessor.

Presentation Layer

.NET offers GDI+, which provides richer image control including support for more image formats. It also provides high-end graphics features such as alpha blending, anti-aliasing, gradients and transparency. With these features, Chart FX can create richer, more complex charts that still have the performance our customers demand.

Security Layer

A computer that downloads managed code from a distrusted source can protect itself in a way that unmanaged code can't. Code access security in the CLR acts as the traffic cop to assemblies, keeping track of where they came from and what security restraints should be placed on them. However, if a component calls unmanaged code, it can bypass code access security measures, becoming terribly dangerous.

With .NET, identity is based on the code rather than the user, policy is set by the administrator, and no certificate dialog boxes appear.

In the Chart FX context having a secured full-interactive component means more functionality and a richer, yet secure, experience for end users.

Debugging Layer

.NET solves one of the nastiest problem developers face today: memory corruption and leaks. The CLR takes care of pointers and memory management. This does not only means more stable components but also let us concentrate on solving the user's problems instead of fiddling around looking for that memory corruption that can ultimately affect the performance of your application and the stability of users systems.

Portability Layer

.NET puts us one step closer to truly portable components through implementations of the CLR for other operating systems or hardware platforms or through proprietary translation tools that allow component vendors to leverage their knowledge in C# and automatically translate them into Java.

Where does Software FX go from here?

So far you've read about intrinsic benefits of using .NET vs. COM components. However, once you're on the .NET wagon and looking for .NET components, you may ask what really sets Chart FX apart. In other words, why should you choose Chart FX over another charting solution?

The answer to this question embodies more than just technical issues. I'm sure that if you are a current Chart FX user, you'll understand this connotation. Nevertheless, I promised I wouldn't make this paper a sales pitch for our organization. Therefore, I will limit myself to what has been the foundation on which we have developed Chart FX: ***Innovation!***

With every major technology introduction, Software FX has found a need to produce innovative charting technology and the advent of .NET is no exception.

Our way is all the way

.NET and COM share many central themes, including component reuse, language neutrality and extensibility. Essentially, .NET incorporates the best aspects of COM while alleviating many of the limitations associated with COM components.

Like any other .NET component, Chart FX will benefit from many of the .NET intrinsic advantages. In truth, you will see these benefits have a greater impact on the tools and techniques, rather than the components, you use to develop and deploy applications.

To illustrate, suppose for a moment we go through a costly and time consuming process of porting Chart FX from C++ to C# without adding additional features to the product. In other words, suppose we give you a .NET version of Chart FX with the same feature set as its COM counterpart. Even though you will take advantage from the .NET core benefits, you would still demand additional graphics and interface pizzazz as well as other tool or platform features that are brought by a new technology like .NET. For example, most of you will want and expect Chart FX .NET to take full advantage of the new features in VB.NET or ASP.NET. This is why a simple feature-for-feature migration would not render the desired results.

You see, for Software FX, .NET represents a natural progression; we see this as an opportunity to incorporate many new features and customer feedback into our product line, whether they are related to .NET or not. In this sense, .NET will help us enhance and eventually replace our existing product line with a more powerful charting technology.

For example, we have devoted significant energy to new interfaces in our .NET offering. Our GUI-designers have worked to minimize the number of steps required for a developer to create and quickly deploy a chart. This means, as a developer, you would only be required to "connect" your data to create an attractive chart that displays well on a browser, on a client-server application or on a Pocket PC. *Now that's progress!*

More than a pretty face

While adding an innovative interface to create and manipulate charts doesn't relate to .NET at all; there are other areas where .NET does have a great impact and influence on Chart FX.

For example, let's consider the presentation layer on today's existing charting products. Some of our competitors have incorporated DirectX as a way to create stunning charts while Chart FX has remained neutral using native Windows API functions to paint the charts.

100
80
60
40
20
0
MCFX

GDI+ will allow Chart FX to generate dazzling charts without compromising its platform reach and other important features like small memory footprint, scalability, Real-time display and easy deployment.

Using DirectX has little or no effect on a 2D chart; however, it can provide significant differences when displaying 3D charts by allowing the use of lighting, textures and shadows.

While charts rendered with a DirectX engine can be visibly astounding, the fact is that the majority of our customers use Chart FX for data analysis purposes. Therefore, for a long time, our focus has remained on improving the chart's readability and availability for a variety of platforms; these features have far outweighed the need to display a stunning 3D chart.

With Chart FX for .NET, our developers and designers were able to take advantage of the GDI+ high-end graphics features such as alpha blending, anti-aliasing, gradients and transparency. The result is an enhanced presentation layer that will greatly complement your client/server apps and web servers without degrading the chart's readability, speed, scalability, portability and memory footprint.

One Size Does Not Fit All

The introduction of a new technology like .NET unlocks the doors for discussions on controversial topics like migration and portability. Although it is entirely possible to see implementations of the CLR built for other operating systems and hardware platforms, I would like to limit the scope of this discussion to the different Windows platforms and tools on which you can use components.

Many vendors would lead you to believe a .NET component will be as suitable to be used in Visual Basic as it is as a web control in ASP.NET. Some may try to convince you that because you will likely see a version of the CLR for Windows CE, that automatically makes a .NET component suitable for a mobile application. As far as Chart FX is concerned, to be efficient and robust, one component does not fit all applications.

Software FX has characterized itself by designing different Chart FX versions with different features for different platforms thus maximizing its benefits to developers and end users. For example, Chart FX Internet provides Internet developers a higher level interface on top of typical HTML and ASP functionality. This product provides not only browser-based components but server-side software and tools to assist the developers in writing scripts for different servers and browsers available in the market.

Likewise, there are particular technologies and facilities specifically built for our Financial, Pocket PC, Real-Time and Wireless offerings, making Chart FX the only charting tool to provide specialized functionality for these vertical markets.

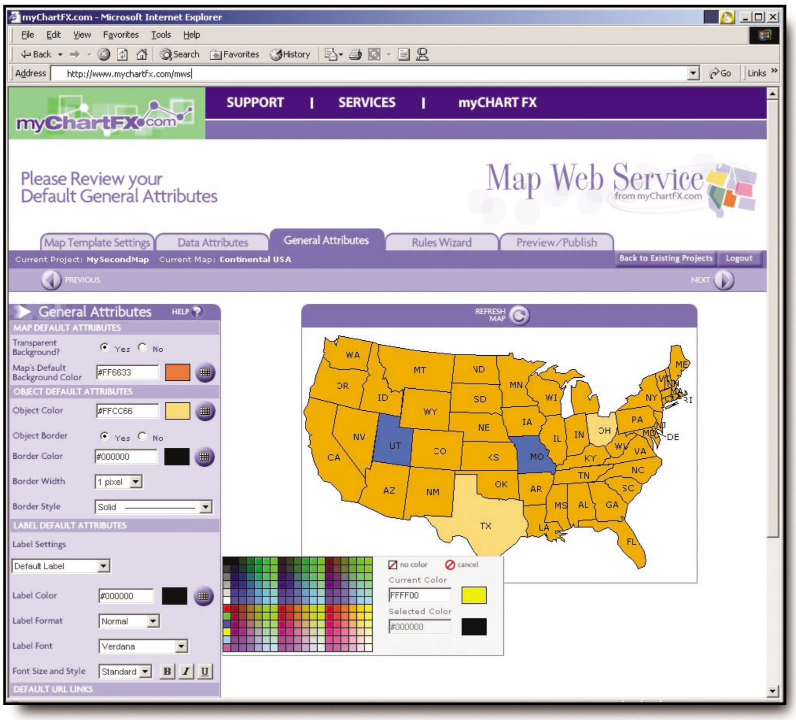
There's no doubt the .NET framework will complement the way these products are designed, built and deployed. However, .NET can never replace all the development effort and the solid foundation on which Chart FX rests today.

.NET and beyond...

.NET coincides with the release of a series of promising technologies for the next wave of internet computing, like XML, Web Services and SOAP. For Chart FX, the most important of all is web services.

Web services are a new class of web application that will simplify distributed computing on multiple (Linux, Windows, Unix, etc.) platforms, thus increasing application deployment opportunities for the entire Chart FX product line. In other words, web services present the opportunity to make Chart FX agnostic. Similarly, with the option of subscribing to these services, organizations can make a "build or rent" decision as to where they want to spend their development resources.

For us, web services are not an abstract concept. With an already released Map Web Service at myChartFX.com, we hope our customers can visualize some of the revolutionary charting products that will follow. To us, .NET provides an excellent platform to materialize this vision.


myChartFX.com - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address Go Links
SUPPORT | SERVICES | myCHART FX
Map Web Service from myChartFX.com
Map Template Settings
Data Attributes
General Attributes
Rules Wizard
Preview/Publish
Back to Existing Projects
Logout
PREVIOUS
NEXT
HELP
REFRESH MAP
no color
cancel
WA OR CA ID NV UT AZ MT WY CO NM ND SD NE KS OK TX MN IA MO AR LA WI IL IN MS AL TN KY OH MI GA FL SC NC VA WV PA NY MD DE NJ ME VT NH MA RI

Address: http://www.mychartfx.com/mws

Please Review your Default General Attributes

Current Project: MySecondMap   Current Map: Continental USA

General Attributes

MAP DEFAULT ATTRIBUTES

Transparent Background? [✓ Yes] No

Map's Default Background Color: #FF6633

OBJECT DEFAULT ATTRIBUTES

Object Color: #FFCC66

Object Border: [✓ Yes] No

Border Color: #000000

Border Width: 1 pixel

Border Style: Solid

LABEL DEFAULT ATTRIBUTES

Label Settings: Default Label

Label Color: #000000

Label Format: Normal

Label Font: Verdana

Font Size and Style: Standard B I U

DEFAULT URL LINKS

Current Color: FFFF00

Selected Color: #000000

Conclusion

For over a decade, components have brought pre-built functionality to millions of developers—simplifying development, reducing costs and shortening time-to-market. Some of these components, like Chart FX, have blossomed into full-featured products that embed and deliver incredible functionality for a very reasonable price.

.NET promises to bring a set of benefits that address some of the common limitations associated with component development today. Nevertheless, the effort that puts .NET development into motion comes with more strings attached for developers than with previous technology changes. Keep in mind that migrating to .NET is not just changing the way you write your syntax. You are encouraged to use the built-in class libraries of the CLR for areas such as user interface construction, database access and XML processing.

If you take the time to learn and take advantage of .NET, then we, as component vendors, should do so as well. We understand that your migration efforts will be concentrated on other application tiers. But keep in mind some of these benefits can quickly fade as soon as you run an unmanaged control. Replacing your components makes sense and should be as simple as an affordable upgrade fee.

My advice for complementing your .NET strategy with components is centered on a three-tiered approach.

First, carefully inspect the components you need and make sure they are 100% written in C#, do not even consider the ones that have been retrofitted for .NET using a custom manageable wrapper. Ask your vendor if they have recoded their components in C# or VB.NET.

Secondly, make sure vendors have taken the time to incorporate new features that take advantage of the functionality built into the CLR. Carefully inspect what kinds of enhancements are provided in comparison to their COM offerings.

Finally, remember .NET doesn't necessarily mean a component is suitable to run on all platforms. You need to inspect these components further and take a closer look on how they really exploit the platform for which you are developing.

We know Chart FX has always been a welcome addition to thousands of developers looking to manipulate, format and display data from varying sources. If you're one of those developers we thank you for your continued support. On the other hand, if you're about to upgrade to a .NET version of your charting component, what better time to investigate the upgrade and competitive upgrades available for Chart FX. Let Software FX show you the Chart FX advantages and help you on the road to .NET.

Chart FX Internet
Chart FX Client Server
Chart FX Financial
Image Toppings
WebBar FX
WebTree FX
myChartFX.com
Map Web Service
Chart FX for .NET
Pocket Chart FX
Chart FX Real-Time
Chart FX Wireless
Chart FX Internet
Chart FX Client Server
Chart FX Financial
Image Toppings
WebBar FX
WebTree FX
myChartFX.com
Map Web Service
Chart FX for .NET
Pocket Chart FX
Chart FX Real-Time
Chart FX Wireless

About Software FX

Software FX, Inc., a company founded in 1993, leads the way in bringing graphical development tools for the development community worldwide, and products for emerging technologies such as Internet. The company is 100% committed to serve their customers and to provide top of the line components for development environments. Our product line includes the royalty-free 32-bit Chart FX Client Server, the seamless web integration of Chart FX Internet, the technical analysis of Chart FX Financial, the convenient mobility of Chart FX Wireless/AireLogic and Pocket Chart FX, the state-of-the-art Chart FX Real-Time and Chart FX for .NET, and the amazing Map Web Service at myChartFX.com and the organizational tools, WebBar FX and WebTree FX.

Software FX
5200 Town Center Cir., Suite 450
Boca Raton, FL 33486
(800) 392-4278
(561) 999-8888
tech support (561) 392-2023
fax (561) 998-2383
www.softwarefx.com
www.mychartfx.com

www.softwarefx.com