Chart FX Internet

Chart FX Client Server

Chart FX Financial

Image Toppings

SoftwareFX

Any Chart, Anywhere!

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 Internet Performance & Scalability Test

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

SoftwareFX

Chart FX Internet Performance & Scalability Test

Table of Contents **Page**

Introduction

Today, most developers look for a variety of innovative tools and techniques to make web application development faster and easier. While it is true that these tools have shortened the development cycle and allowed quicker deployment of web applications; in the end, this new development paradigm has revealed a whole new dilemma: ***server scalability and performance.***

The real issue in web application design is user experience: to make the application perform as a traditional desktop application when it is actually a distributed application that could be simultaneously servicing hundreds or even thousands of users. Therefore, we'll define scalability as the ability of a system to accommodate a growing number of users and to give each user satisfactory levels of responsiveness.

One of the most successful and effective development techniques is the use of COM components to encapsulate business logic into reusable modules. In fact, it is component reusability what made ASP and IIS a big success. However, in web development this flexibility is known to have caused performance problems if components were not designed with the ASP and server implementation in mind.

Too often, developers pin down COM components as the primary cause of bottlenecks in their web applications. Today, diagnosing and addressing performance problems under ASP continues to be a challenge for many web developers. This is especially true with web applications that use COM components, because:

1. Many components do not internally address some of the issues associated with server performance and scalability. For example, a component that requires state or does not have a suitable threading model for ASP implementation will perform poorly under heavy load.

2. Even when the proper components are chosen, if used improperly, the application will yield poor performance. For example, an ADO object could cause a significant bottleneck if it is doing a complex or poorly designed SQL query. This does not mean that the object itself is slow, but that it is being used inefficiently.

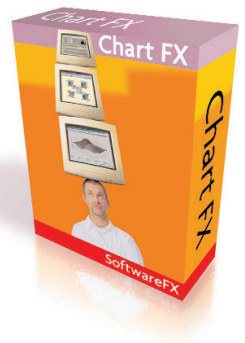
3. Many ASP components end up hosting code that was originally written for desktop applications (Wrapped Legacy). In reality, web applications are, for the most part, server applications and their components deserve implementations that apply to server technologies.

4. Many ASP components lack tune-up mechanisms or properties that will allow them to perform and scale accordingly.

5. Components can cause resource contention and/or leakage which are the main issues behind web applications that are not scalable. For example, a component with a minimal memory leakage has little or no impact on a desktop application where much of the processing load is handled by the client; however, it will prove catastrophic on a server under heavy user load.

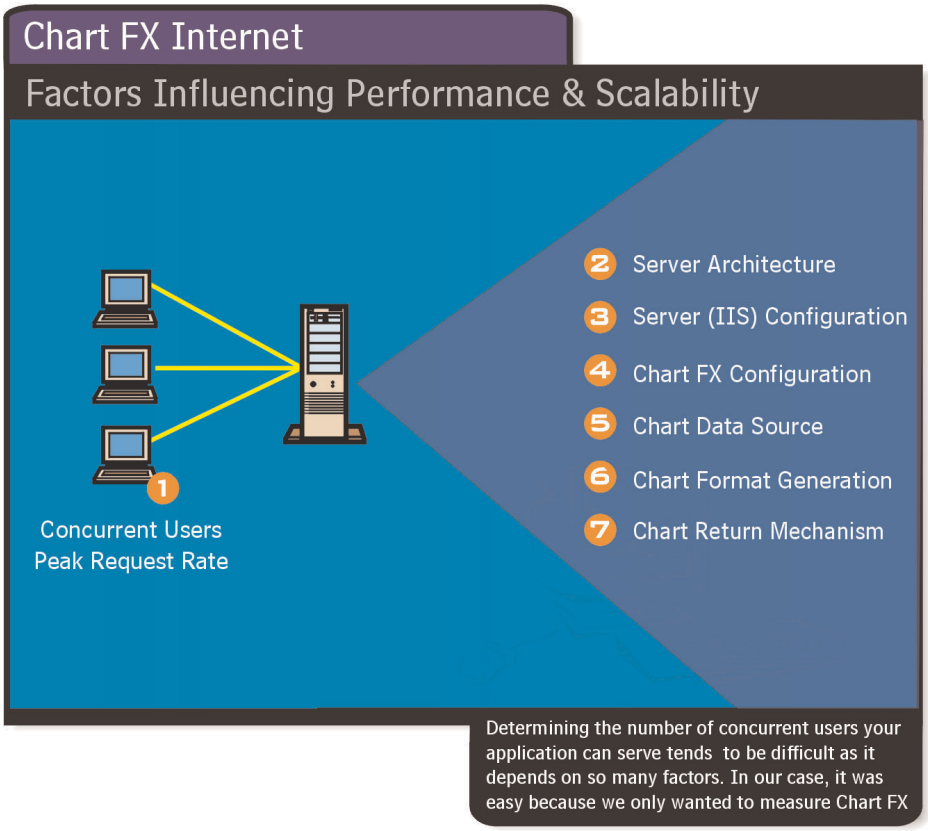
6. Developers using or writing ASP components fail to test their applications appropriately. In the end, when they deploy their applications on the Internet, they discover that they have miserable throughput and response times.

This paper is geared to developers using or planning to use Chart FX Internet in their web applications. In the following paragraphs, you will learn Chart FX Internet is a well-tested Internet charting solution designed with ASP and server implementation in mind. At the same time, we'll shed light on some of the most significant performance and scalability pitfalls when using Chart FX Internet and how to address them using tune-up mechanisms or properties provided by the product.



It is important to note that we ran a few specific isolated benchmark scenarios on Chart FX Internet to quickly identify scalability and performance issues and to reveal tips & tricks to fine-tune our charting engine. However, nothing will replace testing in a way that closely parallels the actual stresses your web application would undergo in a "real-world" setting.

Factors Affecting Performance & Scalability



One of the biggest challenges becomes determining how many concurrent users your web application can support.

Before you can measure, make improvements or do capacity planning you must identify factors that influence performance and scalability.

In a "real-world" scenario, you'll need to find potential bottlenecks in your server before you can take appropriate actions. In our case, we went straight to the Chart FX server component and determined issues that played an important role in its ability to perform and scale accordingly.

Server Architecture

Web applications are expected to see significant performance differences when tested on diverse server architectures. For this reason, we tested Chart FX on single CPU, dual CPU and web farm architectures. We "cleaned" all systems and installed Windows 2000, equal amounts of RAM, same hard disk and CPU speed to reduce the number of hardware factors that could affect the outcome on a particular architecture.

IIS Configuration

We configured IIS in a way to reduce performance and scalability issues due to web server features; we changed the following IIS settings:

1. We turned off ASP server-side debugging; we did this to prevent the ASP code to lock down to a single thread.
2. We turned off server side logging and directory indexing.
3. We turned off ASP session management. The problem with ASP session management is that it unnecessarily wastes precious resources such as processing cycles, memory, and network bandwidth.

No other changes were made to other ASP or IIS defaults. Changing defaults isn't necessarily bad, but you must be cautious of products that force you to change these defaults as they may have an impact on your server performance.

Also note none of these changes are necessary to make Chart FX work with your IIS server. They were simply made to reduce the number of factors affecting our performance tests. However, practically speaking, your application may use some of these features and therefore you will be forced to activate them. However, be aware that they will adversely affect the way your application performs under heavy load.

Chart FX Server Configuration

The Chart FX Internet server was designed to conform to ASP guidelines and to take advantage of the server-bound nature of web applications. While it is not the purpose of this paper to discuss many internet related features of our product, you should know that as an ASP server component Chart FX provides tune-up mechanisms and properties that allow developers and system administrators to easily boost performance under particular server configurations.

To understand how they apply, consider that when the ASP code is running, the Chart FX Server component will essentially create a chart in memory using a device context created on the server. This means, hardware and software settings on your server will affect how these charts are created and therefore have an impact on server performance and scalability.

In the next pages you will learn that the Chart FX server performance will depend to a great extent on the chart format generation and the chart return mechanisms you choose in your ASP code. However, when generating chart images on the server you should pay close attention to color dithering, image compression and chart size as major performance factors when the server is under heavy load.

Huh? Color Dithering?

Color dithering occurs when a server generates an image that uses colors not available to its palette. Windows will then pick two or more solid colors that are available and mix them together to achieve some resemblance of the intended color. Unfortunately, this color substitution process not only doesn't do a very good job at fooling the eye but it also has its toll on server performance.

A rule of thumb is that you must prevent color dithering to boost performance. To do this, you must change the server video card settings to 24-Bit Color and change Chart FX default 8-bit colors to 24-bit.

You can do this by editing the CfxSrv.ini file located on the `config` directory of the Chart FX installation directory on your web server and change the `[Png]` section `Color` setting to 24. This ini setting will apply to all charts created on the server. You can also do it on a page by page basis by using the ImgColors property of the chart control, like this

```
<% Set Chart1 = Server.CreateObject("ChartFX.WebServer")
       Chart1.ImgColors = 24

     //Remaining ASP goes here
%>
```

Please note this setting will have no effect if you do not set your server video card to 24-bit color. You can also prevent color dithering and achieve the same performance enhancements with 256 colors by choosing, in your chart, only solid colors that belong to the default 256 color Windows Palette.

Image Compression

The Chart FX server Png algorithm provides an image compression factor that can be used to reduce file size. This value ranges from 0-100, being 100 the maximum compression ratio.

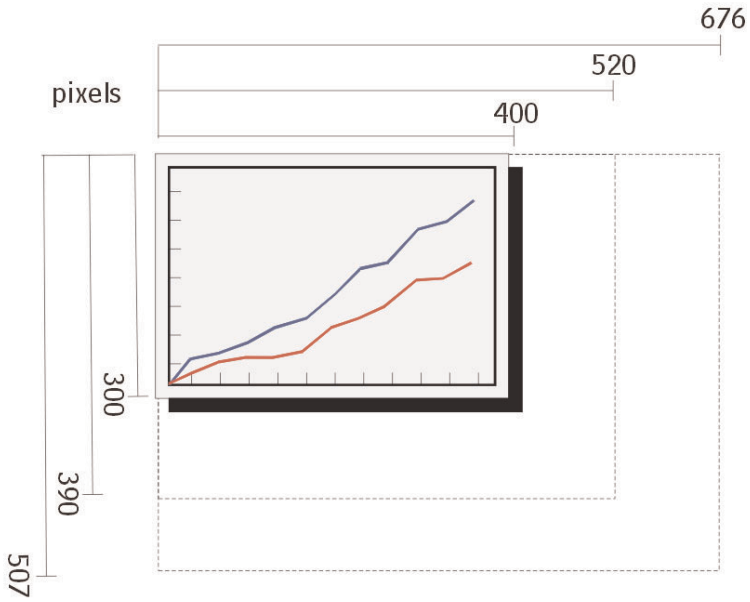
Maximum compression will enhance download time of chart images considerably. Unfortunately, this compression algorithm has an adverse effect on server performance. Similarly, uncompressed Png images will resemble pixilated bitmap images which are simply too big for a web application and will also have an adverse effect on server performance by significantly increasing hard disk usage.

During our tests, we committed to a compression setting of 30, this setting produced well compressed images without significantly affecting server performance.

You can do this by editing the CfxSrv.ini file located on the config directory of the Chart FX installation directory on your web server and change the [Png] section Compression setting to 30. This ini setting will apply to all charts created on the server. You can also do it on a page by page basis by using the ImgCompression property of the chart control the same way it was described in previous paragraphs.

Chart Render Size.

A common practice among web developers is to create a big chart that can be easily read on a browser. However, this practice can be an important factor on how your server behaves and performs under heavy load. Essentially, a bigger chart means a larger image that needs to be processed, generated, stored and finally downloaded; affecting, in one way or another, the overall application's performance. Therefore, you must be careful when choosing the final chart rendering size in the page if server performance is a real concern.



To illustrate this, we ran a series of tests on which we increase the size of a 400x300 Png image of a chart and measure its impact on server performance.

We found that for every 30% increase on the chart width and height, the server experienced a 65% decrease in the number of Requests Per Second it could process.

Resourcing to change the chart size may be costly on your overall page design. However, it may be an excellent source to boost performance if you have exhausted all other options.

While reducing the chart size considerably will benefit server performance, you must be careful not select a size too small will compromise the chart's readability. During our tests we selected a 400x300 pixel chart. We believed most charts displayed well at this size and were easy to read on the browser independently from the number of points contained in them.

Chart Data Source.

In most cases, developers will access relational database management systems (RDBMS) to populate charts. However, because there are so many variables that can greatly affect the application's performance like database vendor, complexity of SQL statements, number of records and database engine configuration; we ended up deciding to populate the charts with randomly generated data.

In other words, while database access becomes a real factor influencing performance and scalability, we decided that it was going to complicate our tests (geared to measure Chart FX as an isolated ASP component) unnecessarily. Nevertheless, during the course of our investigation, we ran a few tests to see how SQL Server would affect the Chart FX performance (these results are reported in subsequent pages). However, it is important to note, database access, its impact, or tuning for performance purposes was not the focus of this paper.

Finally, while Chart FX supports up to 2 billion points per chart, we decided to supply 100 randomly generated points to the charts. This limit will probably cover most of our users needs and will properly test drawing algorithms in Chart FX when a significant user load is imposed on the server. In no way, were these numbers prepared, sorted or optimized to boost performance results.

Chart FX Internet Supported Formats

As a server component, Chart FX allows developers to generate charts in a myriad of formats. Which one to choose depends not only on performance and scalability, but also on other important issues such as browser compatibility, interactivity, accessibility and security.

The Chart FX Internet server component can dynamically generate and render the following chart formats:

	Format	Comments
ActiveX	OLE	- Great for performance since server produces a small OLE file that will be opened by the ActiveX to paint charts on the client. - Small viewer download (Approx. 350KB). Full interactivity. - Windows-based clients only.
PNG	Raster	- Best Image format for producing charts. However, painting and compression algorithm negatively impact server performance. - Supported if browser ver 3.0 or above. - Provides Limited interactivity.
JPG	Raster	- Produces fuzzy charts that are difficult to read on client. However, algorithm is slightly faster than Png. - Universal access. - No interactivity.
SVG	Vector	- Enhances performance significantly; files are small and painted on the client. - Large viewer download (Approx 3MB). Limited Interactivity. - Accessible from many platforms.
GIF	Raster	- Good image format for producing charts. Impacts server performance as much as Png. - Universal access. - No interactivity. - Licensing restrictions and support for 256 color images only.

**** GIF format was not included in our tests since it's not provided with our standard package due to licensing restrictions imposed by Unisys.*

Since the purpose of this writing is to analyze factors that influence performance and scalability, we disabled automatic browser detection during our tests and forced Chart FX to generate a particular format and then measured its impact when the server was under a heavy load.

In general, chart formats that require a viewer (i.e. ActiveX & SVG) enhance server performance since chart files are very small and each client will provide much of the processing load in painting the chart. These viewers will also allow a better analytical experience by allowing browser interaction without additional intervention from the developer or trips back to the server. Some organizations do not allow the use of viewers because of security and accessibility reasons.

On the other hand, producing raster images (i.e. PNG, JPG) decreases performance since each chart needs to be painted and even stored on the server. Also, these charts will provide very limited or no analytical capabilities on the browser since they are rendered as static images –except for PNG images that support hot spots or URL links on most chart elements—However, their greatest advantage in using chart images is that they provide universal access to charts from any browser, platform or operating system.

Chart FX Internet Chart Return Mechanisms

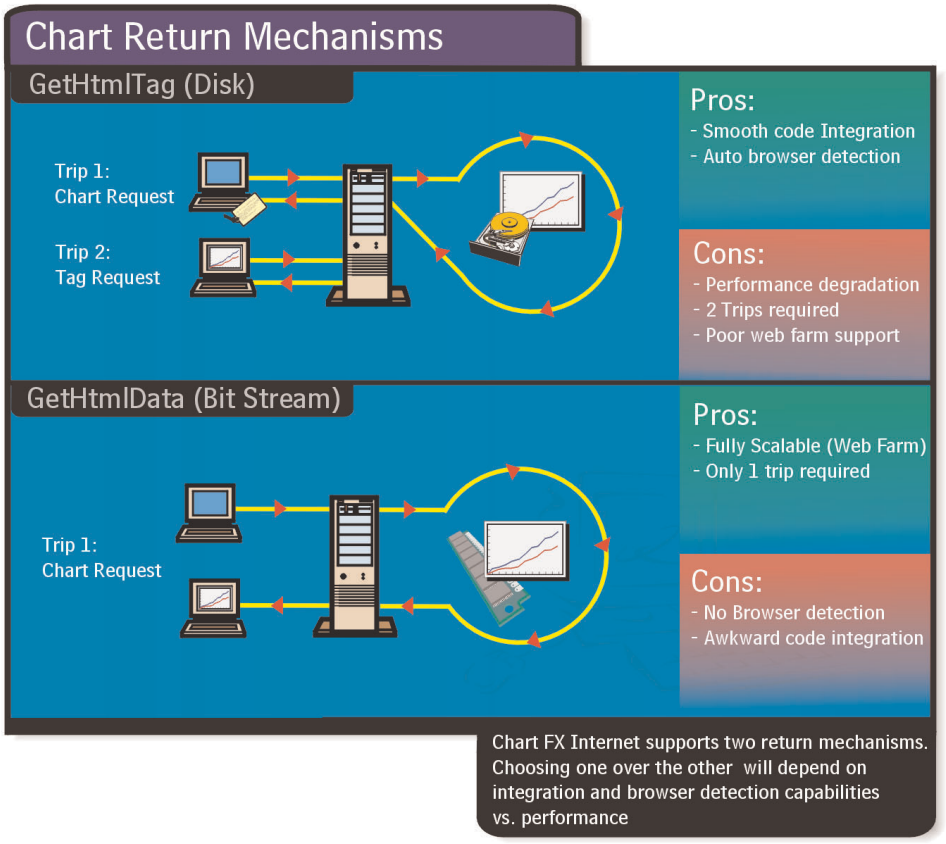


Chart FX Internet supports two return mechanisms. Choosing one over the other will depend on integration and browser detection capabilities vs. performance

To improve the level of responsiveness as well as enhancing support for different server architectures, Chart FX Internet provides two ways of processing and returning charts to the browser.

These mechanisms are provided by 2 methods that are invoked at the end of your ASP code, they are: ***GetHtmlTag*** *and* ***GetHtmlData****.*

Please note this paper's intention is to provide useful performance and scalability information on Chart FX. Therefore, we do not provide steps as to how these methods are used in your ASP.

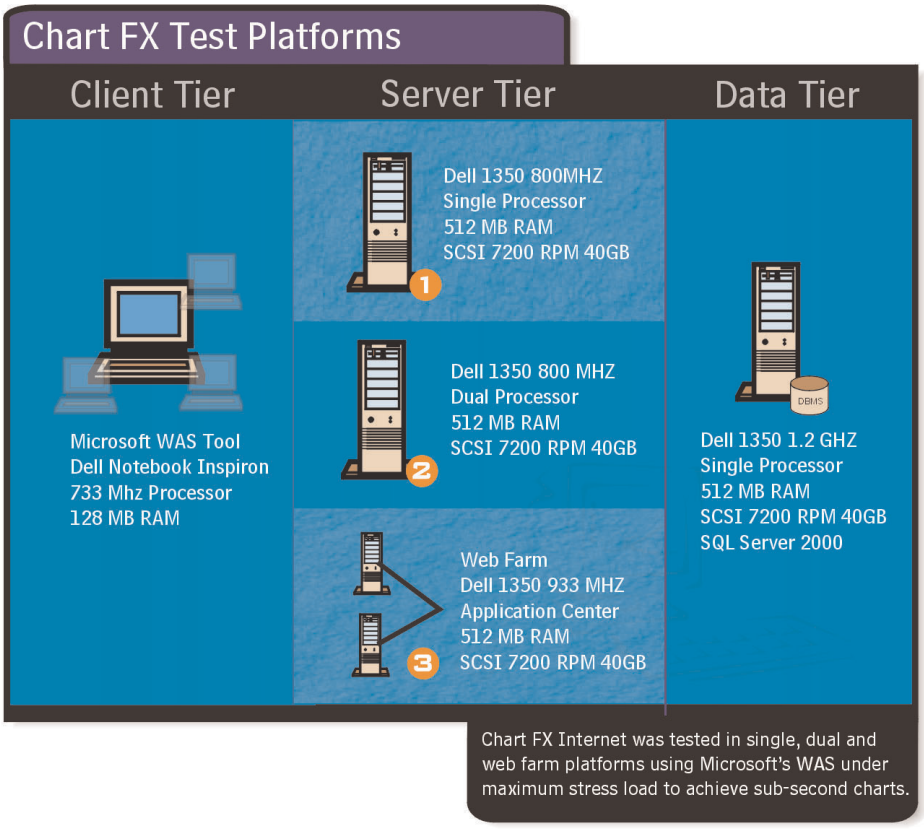
Our tests were performed on different server architectures with all chart formats and comparing these 2 return mechanisms. While it is not exact science, our measurements will give you an idea of where you can focus if you find Chart FX to be a factor influencing performance in your application.

When a user hits a page containing the GetHtmlTag method, a chart file is saved to disk and an HTML tag (IMG, OBJECT) is returned so the browser can know the location of the chart file on the web server. Finally a second trip back to the server is necessary to pull the chart from the server and display it on the page as a static image or an active component. One of GetHtmlTag's greatest advantages is that charts can be easily integrated into your existing ASP pages and it allows support for automatic browser detection. However, saving a file to disk can have its toll on server performance.

On the other hand, the GetHtmlData method prevents the chart file from being saved to disk by bit-streaming the chart directly to the browser; this process leads to fewer round-trips between the client and the server. However, integration to the ASP page is awkward because you must point to another ASP file that returns the bit-stream to the browser. Also, no browser detection is provided using this methodology preventing your application from making "smart" decisions based on the user's browser capabilities.

In terms of performance, you will learn that although the GetHtmlData method is processor intensive it allows developers to boost performance and scalability and it permits the use of Chart FX Internet on intricate server architectures such as web farms.

Chart FX Internet Testing Methodology



Web applications must perform well, but good performance when being used by a single user does not necessarily translate into scalability.

A common mistake for people writing ASP applications is to test their applications by using a single browser.

The standard ASP of hitting Refresh on the browser in rapid-succession won't let you know how your application copes with load. Stress testing your application is essential to discover performance and scalability problems.

In our case, we stressed several web server architectures, on which Chart FX Internet was installed, with a Microsoft client tool known as WAS (Web Application Stress Tool).

We increased the stress level (number of concurrent users) on each server until the Time To Last Byte was near 1 second. In other words, we wanted to setup the limits on each server architecture based on its capacity to produce sub-second charts. We then analyzed the number of Requests Per Second each server architecture could provide.

Because the ASPs were simple enough, each test ran for a period of 1 minute. Request delays and throttle bandwidth setups were not used. All tests ran on LAN connections of 10 Mbps (no slow links such as modem connections were tested), which means you may want to add the latency effects in a "real-world Internet" scenario.

During these tests, we gathered the numbers exposed by the following performance counters:

TTLB *(Time To Last Byte)*: Shows how fast the results are returned to the client (in milliseconds), the higher the value the slower the chart was produced on the server. Our goal is to keep this value under 1,000 milliseconds and then measure Requests Per Second.

RPS *(Requests Per Second)*: Our ASP contained only chart scripts. This means, the Requests Per Second counter can be read as Charts per Second produced by the server.

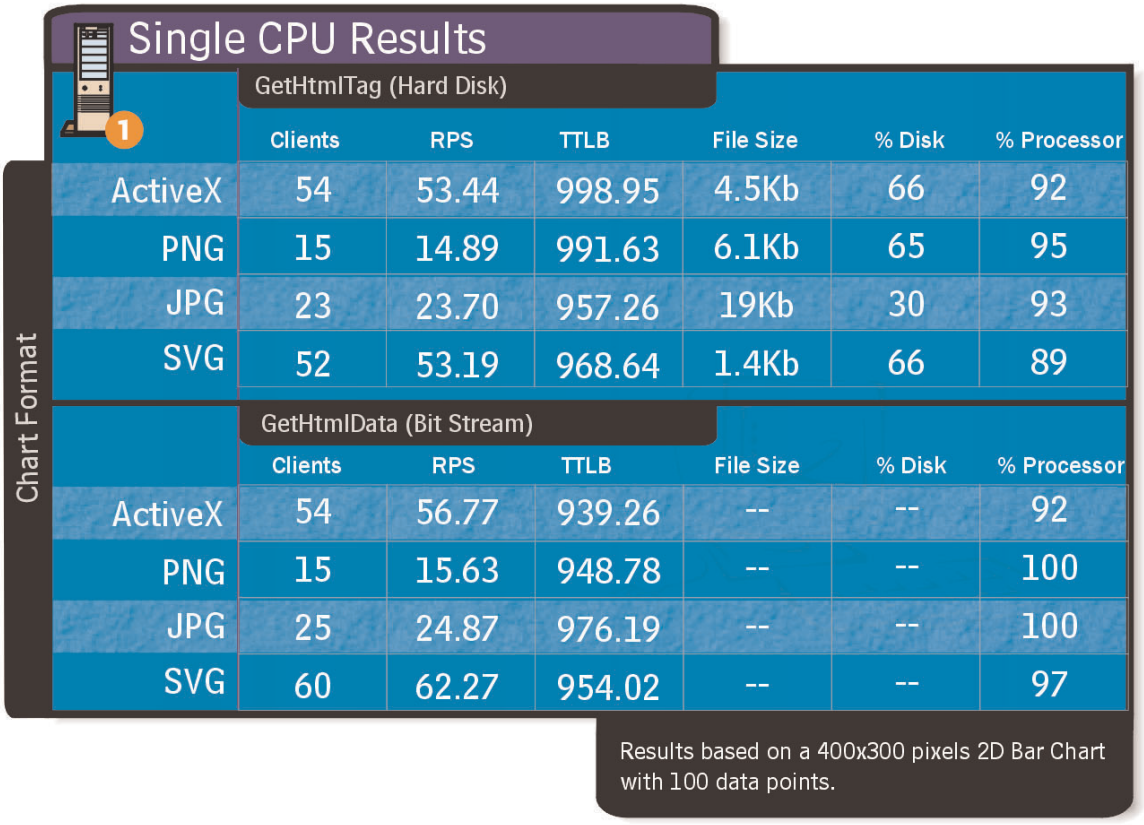
% Disk & % Processor: Percentage of Total Disk and Processor Time. In the case of multi-CPU system, each processor was measured independently and the average use was reported.

ASP Requests Queued: The number of requests waiting for service from the queue.

Available Bytes: This counter tracks the total amount of available memory in the server.

Page Faults/sec: The number of times the server has to page pieces to disk per second. Although this counter was not reported it was measured to be sure it did not increase significantly during our tests.

Chart FX Internet Performance on Single CPU Systems



Single CPU Results

GetHtmlTag (Hard Disk)

Chart Format	Clients	RPS	TTLB	File Size	% Disk	% Processor
ActiveX	54	53.44	998.95	4.5Kb	66	92
PNG	15	14.89	991.63	6.1Kb	65	95
JPG	23	23.70	957.26	19Kb	30	93
SVG	52	53.19	968.64	1.4Kb	66	89

GetHtmlData (Bit Stream)

Chart Format	Clients	RPS	TTLB	File Size	% Disk	% Processor
ActiveX	54	56.77	939.26	--	--	92
PNG	15	15.63	948.78	--	--	100
JPG	25	24.87	976.19	--	--	100
SVG	60	62.27	954.02	--	--	97

Results based on a 400x300 pixels 2D Bar Chart with 100 data points.

During our tests with a Single CPU system, we discovered the following:
No memory leaks or IIS disrupting errors (crashes, hung processes or internal server errors) were found in over 120,000 page hits. This is particularly important because a web server, when under heavy load, is expected to perform well without significant service interruptions.

Raster formats (PNG, JPG) are significantly slower than vector formats as charts must be painted (and sometimes compressed and stored) on the server while vector and binary formats, like SVG and ActiveX respectively, use relatively small files that will be loaded by a viewer on the client. Nevertheless, if you must provide support for heterogeneous client systems, generating chart images may be your only option. Please be aware that image generation has a significant cost on server performance.

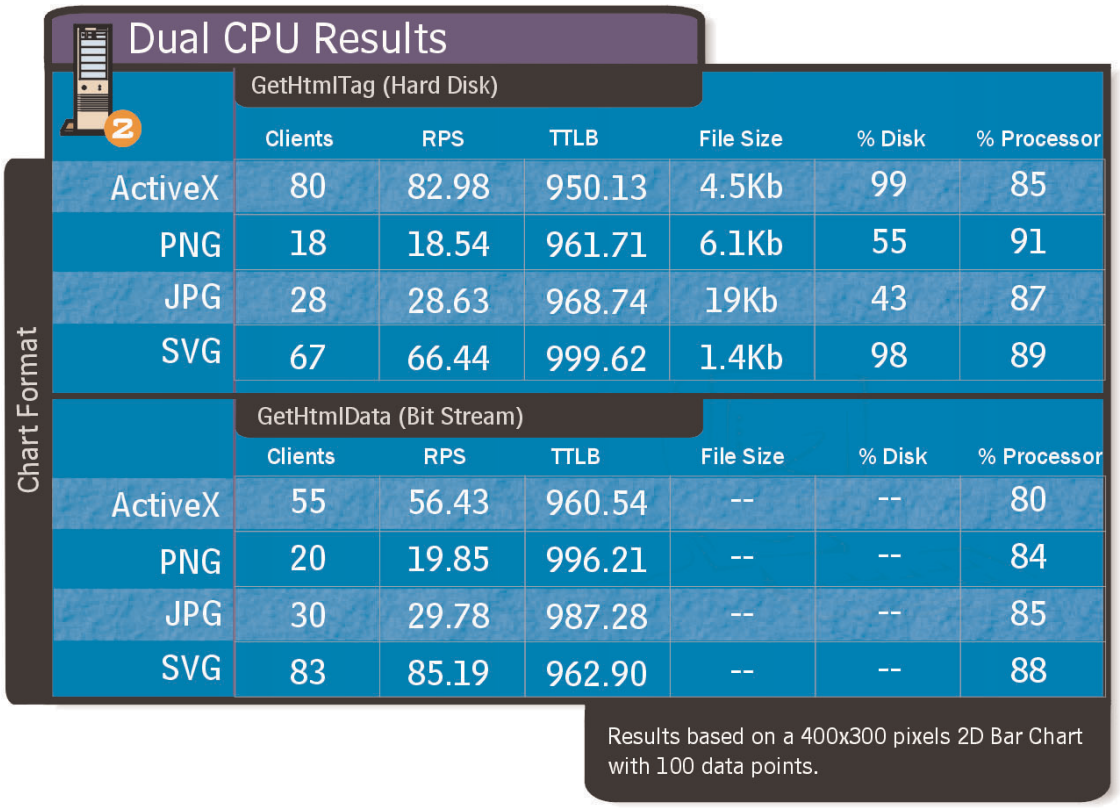
The hard drive was not the bottleneck when using GetHtmlTag method on a single CPU system (this was concluded after inspecting the low % disk utilization counter). This means you should not expect an increase in performance when using the GetHtmlData method on single CPU systems.

All of our single CPU tests indicated the processor was the main bottleneck. Therefore, one can expect a significant increase on performance or a bottleneck shift from the processor to another factor when upgrading the system to multiple CPUs.

Counters affected server performance around 8%. In other words, in a real-world scenario (no performance counter gathering), our results would have been slightly better on the same hardware architecture.

Even though SVG is as fast as ActiveX, if viewer usage on the client is not an issue, we strongly recommend ActiveX since the level of interaction is significantly better. This ActiveX is compatible with Windows, fully secure and signed by Software FX, Inc. If providing support for heterogenous client systems is a must; then the SVG viewer is a better option since it is available for a myriad of operating systems.

Chart FX Internet Performance on Dual CPU Systems



During our tests with a Dual CPU system, we discovered the following:
Just like in single CPU systems, no memory leaks or IIS disrupting errors were found in over 190,000 page hits.

In our dual CPU systems, both processors were working intensively, indicating a well balanced repartition of threads among processors. This indicates a good scalability potential for Chart FX Internet.

We started to see the hard drive playing a role as a performance bottleneck when using GetHtmlTag and therefore a little improvement was experienced when using GetHtmlData (Bit streaming that does not interact with the hard disk). Also, when comparing results; we started to see the bottleneck shifting from the processor (Single CPU) to the hard drive (Dual CPU), this indicates a good usage on dual processors in the server.

During our tests, with GetHTmlData and ActiveX, we experienced a contention problem when using OLE Stream and Storage. Our developers explained that the main problem is due to a MEMCOPY that isn't thread friendly causing degradation on multiple CPUs. In other words, if you are using ActiveX as your chart generation format, your server will perform significantly better when using the GetHtmlTag method.

Counters affected server performance around 7%. In other words, in a real-world scenario (no performance counter gathering), our results would have been slightly better on the same hardware architecture.

Our tests unveiled a better scalability factor on formats that make use of viewers (ActiveX and SVG). In average, these formats were capable of running 38% faster when upgraded from a single to a dual CPU system; while raster formats (PNG and JPG) experienced only an average increase of 23%. This could be easily explained by comparing the amount of work the server must do when creating and compressing images as opposed to saving a file that will be used by a viewer to paint the charts on the client. Our developers believe scalability issues on the C runtime libraries could also play a role.

Finally, we experienced an unusually low disk usage when generating JPG files with the GetHtmlTag method. We believe it may be a contention on the JPG algorithm. At the time of this writing, our developers were looking into this issue. Any significant findings or fixes will be posted on our support site *(support.softwarefx.com)*.

Chart FX Internet and Databases

Most web applications use information stored in databases to produce dynamic content; charts are not exempt to this process. However, when you consider database access there are many variables that can greatly affect an application's performance including but not limited to: database vendor, complexity of SQL statements and database engine tuning and configuration. For this reason, we decided to run most of our tests with charts containing random data.

Since most developers rely on database access to populate charts, we wanted to experiment and report how simple database access would affect results.

To do this, we installed SQL Server 2000 in a separate box (see testing methodology in previous pages for hardware configuration); we then created a single table with a single field containing 100 points that were accessed using ASP's ADO object and a simple SELECT * statement. The ADO object was then passed to Chart FX which would fetch these records and populate charts automatically.

No Session objects were used to keep the database connection alive and/or cached. In other words, a fresh database connection was created, accessed and fetched for every page hit. Further database configuration and/or development techniques could significantly improve results when accessing databases to populate charts. Finally, the ASP was hosted and accessed on the dual CPU system, so please refer to the results on the previous page to compare and analyze database impact.

Chart FX & Database Performance

GetHtmlTag (Hard Disk)

Chart Format	Clients	RPS	TTLB	File Size	% Disk	% Processor
ActiveX	50	49.81	990.10	4.5Kb	66	83
PNG	17	17.62	955.12	5.7Kb	58	89
JPG	27	26.73	999.03	19Kb	48	89
SVG	46	45.86	993.98	1.5Kb	66	86

GetHtmlData (Bit Stream)

Chart Format	Clients	RPS	TTLB	File Size	% Disk	% Processor
ActiveX	60	62.66	948.27	--	--	87.55
PNG	19	18.80	999.94	--	--	85.03
JPG	28	28.16	984.09	--	--	86.82
SVG	55	55.24	985.79	--	--	88.56

Results based on a 400x300 pixels 2D Bar Chart with 100 data points extracted from a SQL Server database.

When comparing with Dual CPU results obtained in the previous page, our tests indicated an average decrease of 34% with binary formats (ActiveX and SVG), while production of raster images was only affected by an insignificant 5%. This means, algorithms that produce raster images on the server continue to be the bottleneck in producing charts, even when populating charts from a database.

By inspecting the low % disk utilization on the GetHtmlTag method and databases we can conclude the ADO object is processor intensive. This explains the difference in dual CPU without accessing databases where the bottleneck was the hard disk.

These tests were not conducted on a single CPU box. Therefore, we can not provide information regarding ADO object's scalability factor. Please refer to Microsoft's web site for additional information on performance and scalability information of ASP's ADO object.

Chart FX Internet and Web Farms

Once you hit a certain threshold, cost-effective scalability requires the use of multiple processors spread across multiple servers. In other words, you'll need a web farm. While there are many web farms and load-balancing solutions, we decided to conduct our tests with Microsoft's Application Center.

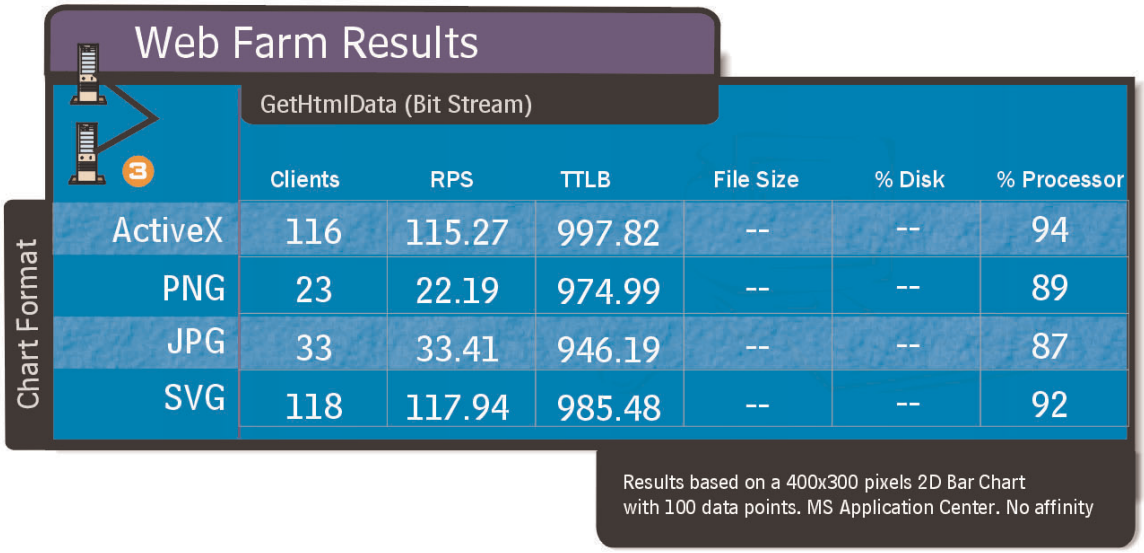
We mentioned that when using the GetHTmlTag method a chart file is saved to disk. one problem on a web farm architecture is that when the browser comes back to the server to retrieve such file; the HTTP request could be directed to another IIS computer on the farm, on which the chart was not originally saved, resulting in an empty chart placeholder on the page.

To address this issue, we attempted the following:

1. Changed the server affinity setting, which lets you instruct the service to route HTTP requests over the same physical server once the first request has gone through. However, affinity slows the system down by requiring it to maintain a user-to-server mapping and perform a lookup upon each request. In other words, changing server affinity compromises scalability and the benefits of request-based load balancing are lost.
2. Replicate the chart file on all servers in the web farm. Although this is possible in a variety of ways, there is no way to guarantee the chart file will be successfully replicated and stored on all servers by the time the browser requests it. This is especially true on intranets where latency effects are virtually non-existent.
3. Forced all servers in the farm to generate charts on a unique server. In other words, tweak the ASP code to have all servers generate charts on a specified directory in a particular box on the farm and make the browser look for charts in such directory. Although a viable solution, this makes chart integration and maintenance a lot more difficult. It will also shift performance bottlenecks to the hard drive on which the charts are being stored and prevent the application to be server fault tolerant, which is a nice addition when using a web farm.

Needless to say, generating a chart file that will be saved in disk poses many server performance and scalability problems on a Web Farm. Therefore, we finally decided to focus on the GetHtmlData method which bit-streams the chart directly to the browser, preventing many of the abovementioned issues and providing a more natural integration to a web farm.

Here are the results of our tests:



Important Licensing consideration: *Chart FX Internet is licensed on a per server basis, which means a separate license is required for each server on the web farm. For more information, please refer to our license agreement or contact Software FX sales department.*

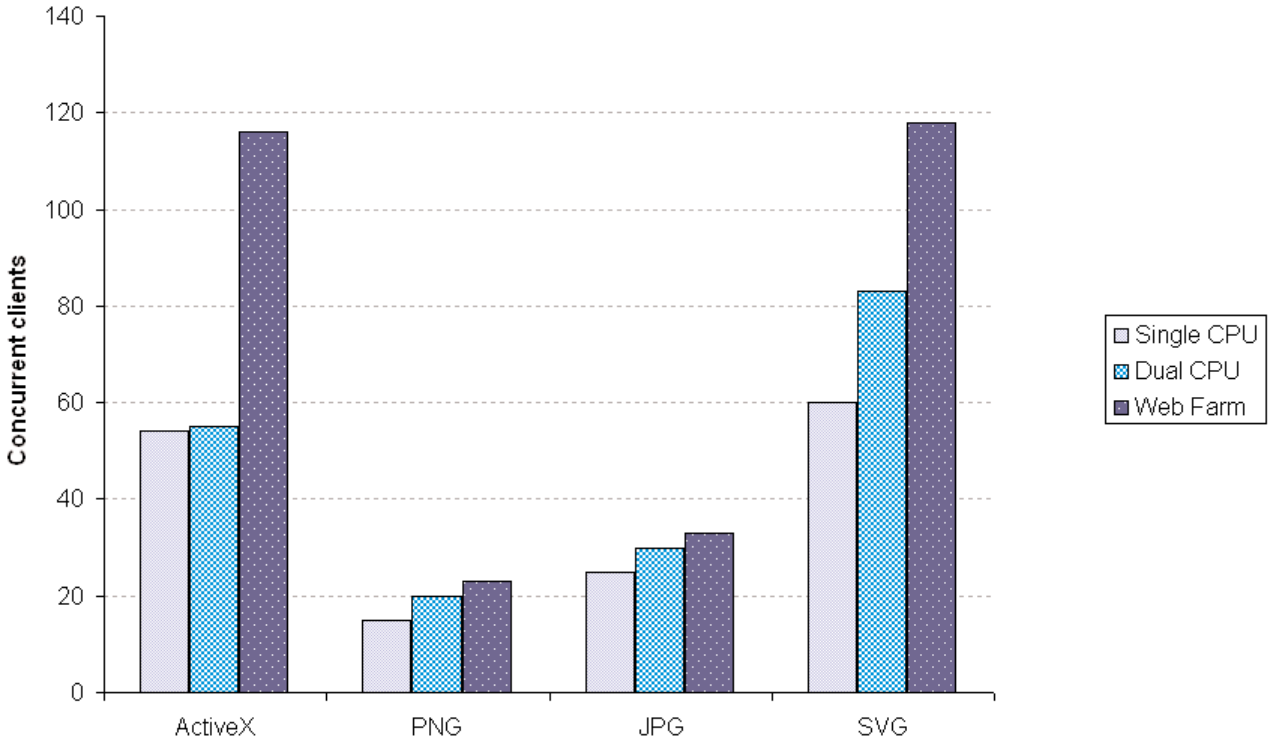
Chart FX Internet and Web Farms (cont'd)

During our web farm tests, we discovered the following:

As originally suspected, web farms produced the best results during our study. In other words, load balancing between 2 separate single CPU boxes produces better results than one with dual processors.

Binary formats (ActiveX and SVG) experienced a better scalability factor on web farm architectures. We were surprised that this factor could not be reproduced when producing raster images (PNG, JPG). We attribute this to a scalability problem on the Png and Jpeg algorithms (Not owned or authored by Software FX) which leaves little room for improvement on our part. Still, a decent number of concurrent users could be served with relatively inexpensive server architecture.

Finally, to put things in perspective, we created the following chart which depicts the performance results we experienced with Chart FX Internet on different server architectures using the **GetHtmlData** method and generating different chart formats:



At the time of this writing, Software FX was experimenting with other product features that would significantly increase performance and scalability by enhancing other areas in the product not related to the raster images algorithms. Please contact Software FX for additional information and availability.

Also, Chart FX for .NET was in its beta stages and although no official performance testing has been made on this product, preliminary tests indicated a significant improvement on performance due to the compiled nature of aspx files. Please stay tuned to a future Chart FX for .NET performance test on our site or contact Software FX for additional information.

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.

SoftwareFX

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