Chart FX OLAP

Chart FX Maps

Chart FX Statistical

Chart FX Financial

Chart FX Real-Time

Chart FX Wireless

Chart FX Developer Studio

Chart FX for Java

Chart FX for Java CE

Chart FX for .NET

Chart FX Lite for .NET

Chart FX for Web Matrix

# Chart FX for Java Performance

Chart FX Internet

Chart FX Client Server

Pocket Chart FX

Chart FX for Delphi

Chart FX OLAP

Chart FX Maps

Chart FX Statistical

Chart FX Financial

Chart FX Real-Time

Chart FX Wireless

SoftwareFX

# Chart FX for Java Performance

**Table of Contents** **Page**

# Factors Affecting Performance

## Server Architecture

Web applications are expected to show significant performance differences when tested on diverse server architectures. With so many platforms, operating systems and application servers available to run Java web applications, it is difficult to arrive at performance figures that represent an average.

We have selected a basic testing platform to start, which can be cataloged as a small Java Application server on its own; but it can also represent a single server in a web farm.

## Application Server Configuration

Each Java Application server has its own way of dealing with performance. There are many variables that will affect the performance in a Java product; from the processor utilization, to the memory and memory heap, cache, etc.

For our test, we have decided to use the default settings, with only the addition of the "-server" option to be sure the VM was working in server mode.

## Chart FX for Java Server

Chart FX for Java was designed to conform to server-side Java bean guidelines and to take advantage of the server-bound nature of web applications. While it is not the purpose of this paper to discuss the many Internet related features of our product, you should know that as a server-side Java bean 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 Java 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 may have an impact on server performance and scalability.

In this document, 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 that you choose in your Java code. However, when generating chart images on the server you should pay close attention to some visual attributes, chart size and format as major performance factors when the server is under a heavy load.

## Chart Visual Attributes

As a data visualization component, we must provide designers and developers with a large list of visual attributes that can be used to customize the way the charts are presented. Artistic borders, anti-aliasing, gradients, transparencies, color palettes, patterns, etc. are all available features that will help developers design elegant and high quality charts.

However, each visual attribute will have an impact on performance. The challenge is to find the perfect balance between form and function.

## Chart Render Size

A common practice among web developers is to create a large chart that can easily be read in a browser. However, this practice can be an important factor on how your server behaves and performs under the heavy load. Essentially, a larger chart means more processing, generation, storage and finally downloading; 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.

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

## Output Format

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.

Chart FX for Java dynamically generates and renders the following chart formats:

| Format | Comment |
|---|---|
| **PNG (Raster)** | · Best Image format for producing charts. However, painting and compression algorithm may negatively impact server performance.<br>· Supported by most popular browsers.<br>· Provides Limited interactivity. |
| **JPG (Raster)** | · Image quality is not the best; however, algorithm is slightly faster than PNG.<br>· Universal access.<br>· No interactivity. |
| **SVG (Vector)** | · Enhances performance significantly; files are small and painted on the client.<br>· Large viewer download.<br>· Limited Interactivity.<br>· Accessible from many platforms. |
| **Flash (SWF)** | · Relatively better performance significantly; files are small and painted on the client.<br>· Large Viewer download.<br>· Limited Interactivity.<br>· Accessible from many platforms |
| **.NET (Binary)** | · Great performance: the server produces a small binary file will be .NET Client Component to paint charts on the client.<br>· Small viewer download (Approx. 350KB). Full interactivity, includes user toolbar.<br>· Windows clients only, requires .NET framework. |

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. .NET & 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, the greatest advantage in using chart images is that they provide universal access to charts from any browser, platform or operating system.

## Chart Returns Mechanism

To improve the level of responsiveness as well as enhancing support for different server architectures, Chart FX for Java 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 JSP 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 JSP.*

Our tests were performed on different server architectures with all chart formats and comparing these 2 return mechanisms. While it is not an exact science, our measurements will give you an idea of where you can focus if you find Chart FX to be a factor to influence 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 easily be 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 JSP page is awkward because you must point to another JSP 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 may become processor intensive it allows developers to boost performance and scalability and it permits the use of Chart FX for Java on intricate server architectures such as web farms.

## Testing Methodology

Web applications must perform well, but good performance when being used by a single user does not necessarily translate into scalability. 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 used a web application stress tool that allowed us to increase the stress level (number of concurrent users) on the server until the "Time To Last Byte" was near 1 second. In other words, we wanted to setup the limits of the server based on its capacity to produce sub-second charts. We then analyzed the number of Requests Per Second that each server architecture could provide.

Because the JSPs were simple enough, each test ran for a period of 1 minute. All tests ran on LAN connections of 1 Gigabit, 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:

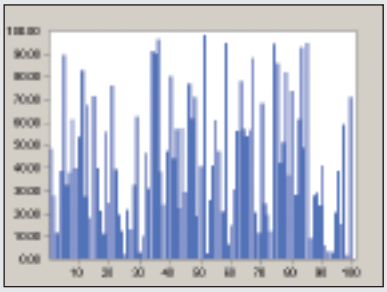
**TTLB (Time To Last Byte):** Shows how fast the results are returned to the client (in milliseconds), the higher the value the slower the chart was produced on the server. Our goal is to keep this value under 1,000 milliseconds.

**% Disk & % Processor:** Percentage of Total Disk and Processor Time. In the case of multi-CPU system, each processor.

### Testing Scenario

*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.*

*We elected to do a 400 x 300, 2D, Bar Char, with no borders or anti-aliasing, and no titles, legends or gridlines. Here is an image of the chart:*



**The following Hardware was used to run the Performance Test:**

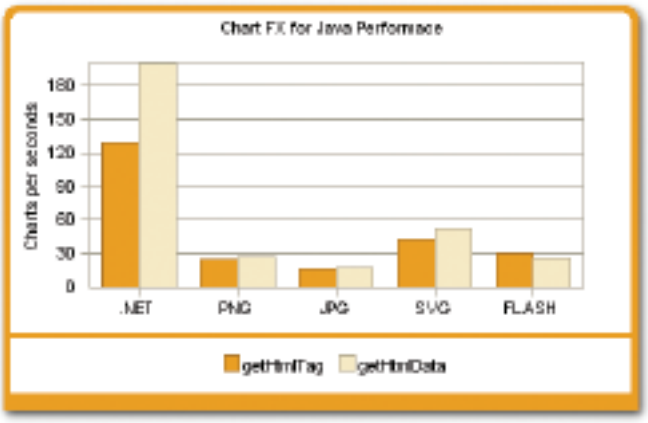
*Server:* *Xeon 2.4, 512KB RAM, 35GB.SCSI HD. Windows 2000 Server SP3*

*Client:* *PIII 800, 256 MB RAM, 20GB IDE HD Windows 2000 Server SP3.*

## Chart FX for Java Performance Results

The number of chart per seconds (CPS) obtained by each format and return mechanism are detailed in the following:

| | GetHtmlTag | | | | GetHtmlData | |
|---|---|---|---|---|---|---|
| | File Size | CPS | % Disk | % Proc | CPS | % Proc |
| **.NET** (Binary) | 3K | **130** | 82.18 | 83.61 | **200** | 90.36 |
| **PNG** (Raster) | 3K | **25** | 33.47 | 90.35 | **28** | 94.83 |
| **JPG** (Raster) | 25K | **17** | 49.71 | 88.96 | **18** | 92.28 |
| **SVG** (Vector) | 14K | **43** | 75.00 | 86.56 | **53** | 91.88 |
| **FLASH** (SWF) | 15K | **30** | 75.15 | 81.04 | **26** | 91.21 |




## Conclusions

Raster formats (PNG, JPG) are significantly slower than vector formats as the charts must be painted (and sometimes compressed and stored) on the server while vector and binary formats, like SVG and .NET 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 to server performance.

While the hard drive does not seem to be a bottleneck when using GetHtmlTag method; the combination of writing a temporary file and the 2 trips required to deliver the file to the browser has an impact on the performance. When using the GetHtmlData method, the increase on performance is clear by, in some cases, being significantly faster.

It is very important to mention that the server used for this test has only one CPU. The GetHtmlData method will take full advantage of a multi-CPU server, producing even better results. At this moment, we are proceeding with the dual-CPU performance testing and the results will be added to this document in the future.

SVG seems to be the right format, as it is available for a myriad of operating systems and a relatively small download. However, the performance obtained with the PNG format gives the best image quality and makes it a strong contender.

While the .NET format is the faster by far, it may not be suitable for all cases because of its dependency to a particular OS and extra software requirements. However, we strongly recommend its use whenever possible, not only because of performance, but also because of the level of interaction.