Performance in the Infragistics® WebDataGrid™ for Microsoft® ASP.NET AJAX

An Infragistics Whitepaper

Contents

Performance and User Experience

Everyone talks about user experience today – it is a key differentiator in the new economy. When most people talk about the user experience of their application, the conversation is generally focused on the aesthetic design of their application, which while important is merely one aspect of the overall user experience. Often people neglect to think about an area of user experience that can have just as strong of an impact on the overall experience of their application as the application's aesthetics **– application performance!**

At Infragistics, we take the user experience of your application very seriously, which means we not only focus on helping you provide a world class visual experience, but also the performance experience of our products in your application. In this whitepaper, you will learn how we approach the architecture and testing of our ASP.NET WebDataGrid so we can deliver the fastest grid on the market.

Why does performance matter so much in the overall user experience?

- Higher Customer Satisfaction
- Improved End User Productivity
- Resource Efficiency in Hardware and Bandwidth

The customer does not want to be, nor should they be bothered with the details of how an application is developed; they just care about their experience when they use an application. They want an application that simply "makes sense", which translates to an application that provides an intuitive user interface, and performs as the customer would expect – in other words – they shouldn't notice or think about how it performs. When you choose the Infragistics WebDataGrid, you get a control that allows you to provide that "makes sense" experience to your end users, including the best performance on the market. And this is not at the cost of implementing a custom ORM (object relational mapper) solution or some custom data provider – this is out of the box data binding. No changes to your data access strategies.

Exceptional Performance Best Practices

During our product life cycle, we test our controls at various stages as new features are added or use cases updated. We follow a series of best practices in the **Planning**, **Configuration**, **Implementation** and **Review** of our performance tests to ensure the controls are tested in the correct scenarios and that the results we generate are accurate.

• Planning – During planning we determine what we are testing against (a previous build, a competitor, etc) and consider the options and tools available for performance testing on the specific platform. We also identify the specific scenarios that we want to test, and how we can create competitive tests that use as equal a configuration as possible.

• Configuration – In this phase we configure a testing environment that simulates as close as possible what customers are using based on the scenarios that describe the product requirements

• Implementation – We implement the tests outlined in the test plan and execute them many times to generate the performance statistics

• Review – During review, we evaluate the test results for consistency, looking for anomalies that can indicate a problem in the testing environment that could be affecting test results

Based on the above described best practices we implement and execute automated performance tests against each nightly build of our products. During the testing, performance results are automatically stored and reports are generated. This enables the development team to constantly monitor the performance through the development process.

Testing the WebDataGrid

For Infragistics ASP.NET controls, the majority of our performance testing was focused on the WebDataGrid, primarily because of its use in many data intensive applications and the large number of features it includes. Because we regularly talk to customers whose applications require grid controls that can load very large volumes of data at very high refresh rates, when setting our performance goals and designing our tests we are able to consider those real-world use scenarios to help guide us. To begin testing the WebDataGrid, the following test environment was created to execute the performance tests:

• Two physical machines (non-virtual), one server and one client – which would simulate a real life scenario.

• Both server and client are running Microsoft Windows Server® 2003 Standard Edition SP2 CPU 2.13GHz, 4 GB of RAM.

• The server machine is running Microsoft SQL Server® 2005 as the backing data store. We used its default settings.

• The server machine is also running IIS 6.0. We used its default settings.

• The client is running Internet Explorer® 8 for its default browser.

• Both server and client are using the same network switch, which is isolated from other corporate network traffic to help us focus on the performance of the grid apart from other performance-inhibiting factors that would be outside of our control.

• Both server and client are configured with a static IP address.

• Both server and client have Windows Update disabled.

To run the tests, we wrote code which ran all of the test scenarios 100 times. The scenarios ranged from simulated user clicks, to complete CRUD (Create, Read, Update, and Delete) operations with SQL Server 2005. Each run used our latest nightly build and the most recent build of competitor controls available. To determine test results, we recorded the time of each operation, and then calculated the differences.

Performance Results

We developed test applications using the latest public bits of our and competitors' grids. These test applications are designed to ensure to the maximum extent that we compare apples-to-apples, and they simulate real world scenarios found in banks, call centers, insurance companies, etc. During this comparison we used all public information which we found on our competitors' websites.

The primary scenarios for grids identified the following areas as critical to the overall user experience of using a grid for data display and editing:

- Flat data binding
- Scrolling (real time, deferred)
- Sorting (string, numeric)

In this whitepaper, unless otherwise specified, we are showing the results of tests executing these scenarios using a data set that includes 10 columns (with all main types of data) and 300,000 rows of data.

In all scenarios we use LinqDataSource, because it gets only needed data from SQL Server which is faster than extracting all data from SQL data tables and getting only needed data outside of SQL Server. SQL Server does it for you.

In all scenarios we use pages with paging (10 rows per page) and filtering on each column. We will split testing and results in two major groups:

- Single user operations in browser
- Load testing

Single user operations in browser

These scenarios include full cycle - browser rendering (Internet Explorer 8) and server response time. These tests more closely represent the customer experience when using the tested grids.

Table 1: Single-user Browser Operations - 10 columns and 300,000 rows of data.

Time (duration in ms)	Infragistics WebDataGrid	Competitor I		Competitor II	
Paging	174	210	21% Slower	447	157% Slower
Numeric Sorting	535	561	4% Slower	800	49% Slower
String Sorting	549	576	4% Slower	828	50% Slower
Numeric Filtering	172	193	12% Slower	491	185% Slower
String Filtering	191	223	16% Slower	515	169% Slower

So how do we achieve such **amazing performance** in our WebDataGrid?
The simple answer: it's the rock solid architecture of the **Aikido™ Framework** on which the WebDataGrid is based. Additionally, our payload size is smaller than our competitors.

What is "payload size?"
The intuitive definition: the amount of data exchanged between server (IIS) and client (browser).

Why smaller payload size is better for performance?
Exchanges of smaller amounts of data will be processed and rendered faster by the browser. Also if you have a constrained connection between server and client, this will be a huge benefit for you - because less data is transported on the wire.

Payload size

Using Fiddler 2 we monitor the data exchanged between server and client. In the table below is shown a summary (sent + received) per operation.

Table 2: Payload size - 10 columns and 300,000 rows of data.

Payload size in bytes	Infragistics WebDataGrid	Competitor I		Competitor II	
Initial Request	208,242	286,168		309,064	
Numeric Filtering	30,062	43,135		22,554	
Unset Numeric Filtering	30,238	43,100		21,943	
Paging	30,491	43,435		26,904	
Numeric Sorting	32,208	43,381		23,914	
TOTAL	**331,241**	**459,219**	**39% More**	**404,379**	**22% More**

Load Testing

When we saw how big our advantage is with a single user, we decided to see what happens when we load our and competitors' grids. We repeated the following steps for WebDataGrid and its competitors:

• We used the same environment and pages for load testing as mentioned above. We used a commercial Web loading tool on the client PC to simulate the load, but you can repeat this load testing with some of the available free Web loading tools.

• Scenario:
    - Initial request
    - Numeric filtering ("Less than")
    - Clear Numeric filtering
    - Paging("Next")
    - Paging("Previous")
    - Numeric Sorting

• Run-Time Settings: We simulated "think time" between all operations, and fixed it to 2 seconds.

• Ramp-Up: Start 2 users every 15 seconds. We used 50 users.

• After the ramp up has been completed, the load tool continues to run the scenario for 1 hour.

Notes: All load tests were repeated several times to remove chance that something can be wrong with the environment. Server and client PC were restarted after each test run. "Competitor II" cannot handle 50 users on this environment. IIS crashes each time with an out of memory error.

Table 3: Load testing with 50 users for 1 hour.

	Infragistics WebDataGrid	Competitor I	
Average Throughput* (bytes/second):	584,873	884,022	51% More
Average Hits per Second:**	55,339	49,006	11% Less

* Amount of throughput (in bytes) on the Web server during the load test. Throughput represents the amount of data that the users received from the server at any given second.

** The number of hits made on the Web server by users during each second of the load test.

As you can see on the first row, our *Average Throughput (bytes/second)* is **51% better** than "Competitor I". This comes directly from our excellent payload size.

On the second row you see that using our WebDataGrid, users have around **11% better** productivity per unit time.

So from that load test we can conclude that users will have **11% increases in productivity** if they use WebDataGrid.

Finally, you should not forget that these results are achieved in environments where both PCs are on one network switch. In real life, often when the server and client are thousands of miles away and network bandwidth is limited, this percentage will increase because of the amazing payload size of WebDataGrid. These benefits could result in tens of thousands of dollars a week, and potentially millions of dollars per year in savings.

Summary

You can see that by employing a solid performance testing framework, you can further drive value in terms of actual saved time and perceived experience. We achieve this by using a solid AJAX architectural foundation and compact payloads. We also follow strict best practices methodology of setting up and running tests - each test is performed on the latest releases of Infragistics and our competitor's latest service releases. Using the WebDataGrid, your applications will not only look better and be easier to use, but they will outperform anything available on the market today. This leads to higher end user satisfaction and actual dollar savings in productivity of the applications that you deploy.
To get the Infragistics WebDataGrid today, download the corresponding product at ~~http://www.infragistics.com/downloads~~