Performance in the Infragistics® xamGrid™ for Microsoft® Silverlight™

An Infragistics Whitepaper

Published September 1st, 2009

Last updated January 15th, 2013

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 Silverlight xamGrid 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 xamGrid, 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 Silverlight 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 xamGrid

For NetAdvantage® for Silverlight, the majority of our performance testing was focused on the xamGrid, 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 xamGrid, 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 7, CPU 2.13GHz, 4 GB of RAM.

• The server machine is running Microsoft SQL Server® 2005 as the backing data store.

• 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.

• On client machine is installed Silverlight 4 (4.0.50917.0)

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 and memory consumption of each operation, and then calculated the differences.
Memory consumption was calculated using the GC.GetTotalMemory function, and time was calculated using normal TimeSpan functions.

Performance Results

We developed test applications using the latest public bits of our and top five different 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.

In this white paper we quote numbers Infragistics xamGrid™ and its best two competitors grids from performance perspective and name them "Competitor I" and "Competitor II".

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)

• Hierarchical data binding

• Expand/Collapse hierarchical data

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 1,000,000 rows of data.

Data Binding

To test the performance of basic data binding we created tests that measure the amount of time needed to bind the xamGrid to different ItemSources and render up to the LayoutUpdated event (or similar event). We believe this most accurately represents what and end user experiences when waiting for data to load.

Table 1: Binding to IList - 10 columns and 1,000,000 rows of data.

	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	1,028	2,050	99% Slower	1,962	90% Slower
Memory Usage Before in KB	91,852	88,964	3% Less	90,683	1% Less
Memory Usage After in KB	91,762	144,394	57% More	128,511	40% More

Table 2: Binding to ObservableCollection - 10 columns and 1,000,000 rows of data.

	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	1,028	2,025	96% Slower	1,962	90% Slower
Memory Usage Before in KB	91,852	91,360	~ Equal	90,660	1% Less
Memory Usage After in KB	91,762	113,795	24% More	128,207	39% More

Vertical Scrolling

The xamGrid offers two options when it comes to scrolling with the scrollbar thumb –Real Time (Immediate) or Deferred. When you have hundreds of thousands of rows, deferred scrolling lets your users preview the record's value as they scroll (for example, in a tool tip) without having to populate all of the cells until the user has reached the point in the data grid where they want to go. We created tests to measure the performance of both options.

For scrolling scenarios we use a Silverlight automation framework which drag & drop the thumb from up to down scroller buttons and calculate the time duration.

Real Time Scrolling

As you will see in the performance results in table 3, real time (or immediate) scrolling is a big problem in Silverlight. Most vendor samples you see will load limited amounts of data because the scrolling performance of the control is very poor.

Table 3: Real-time scrolling with 91 columns and 600 rows of data.

	IG xamGrid™	Competitor I	
Time duration in ms	883	1,448	64% Slower
Memory Usage Before in KB	26,801	33,599	25% More
Memory Usage After in KB	27,934	21,847	21% Less

* It seems that "Competitor II" has issue with scrolling because thumb cannot be moved to the end.

Deferred Scrolling

We could only test one competing grid with deferred scrolling, as it was the only one that offered this feature.

Table 4: Deferred scrolling with 91 columns and 600 rows of data.

	IG xamGrid™	Competitor I	
Time duration in ms	750	752	~ Equal
Memory Usage Before in KB	26782	33614	25% More
Memory Usage After in KB	26726	21888	18% Less

* It seems that "Competitor II" has issue with scrolling because thumb cannot be moved to the end.

Sorting

Sorting performance can vary based on the data type that is being sorted but based on feedback from customers; we test numeric and string sorts against an IList and ObservableCollection data source.

Table 5: Sorting numeric column values bound using an IList - 10 columns and 1,000,000 rows of data.

	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	1,957	2,764	41% Slower	7,889	303% Slower
Memory Usage Before in KB	91,447	125,025	36% More	128,897	40% More
Memory Usage After in KB	129,083	150,777	16% More	141,029	9% More

Table 6: Sorting string column values bound using an IList - 10 columns and 1,000,000 rows of data.

	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	7,172	7,964	11% Slower	9,566	33% Slower
Memory Usage Before in KB	91,378	132,320	44% More	128,707	40% More
Memory Usage After in KB	117,313	149,873	27% More	128,007	9% More

Table 7: Sorting numeric column values bound using an ObservableCollection - 10 columns and 1,000,000 rows of data

	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	1,968	2,818	43% Slower	7,890	300% Slower
Memory Usage Before in KB	91,809	114,765	25% More	128,415	39% More
Memory Usage After in KB	97,388	118,945	22% More	145,979	49% More

Table 8: Sorting numeric column values bound using an ObservableCollection - 10 columns and 1,000,000 rows of data.

	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	7,293	8,038	10% Slower	9,598	31% Slower
Memory Usage Before in KB	91,987	115,063	25% More	128,361	39% More
Memory Usage After in KB	116,807	115,590	1% More	126,567	8% More

Binding to Hierarchical Data

To test the performance of hierarchical data binding we created tests that measure the amount of time needed to bind the xamGrid to different ItemSources and render up to the LayoutUpdated event (or similar event). We believe this most accurately represents what and end user experiences when waiting for data to load.

Table 9: Binding to NorthWind tables Customers-Order-OrderDetails using WCF service and ObservableCollection.

	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	950	1,504	58% Slower	6,486	582% Slower
Memory Usage Before in KB	3,471	3,747	7% More	3,908	12% More
Memory Usage After in KB	6,507	7,315	12% More	11,510	76% More

Expand Hierarchical Grid

These are the number for expand all records functionality tests.
When a button is clicked the whole grid is expanded.

Table 8: Expand all records bound to NorthWind tables Customers-Order-OrderDetails using WCF service and ObservableCollection.

	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	1,398	46,466	3223% Slower	3,217	130% Slower
Memory Usage Before in KB	6,659	7,947	19% More	11,576	73% More
Memory Usage After in KB	17,743	66,916	277% More	20,250	14% More

Collapse Hierarchical Grid

These are the number for collapse all records functionality tests.
When a button is clicked the whole grid is collapsed.

Table 9: Expand all records bound to [NorthWind](#) tables Customers-Order-OrderDetails using WCF service and ObservableCollection.
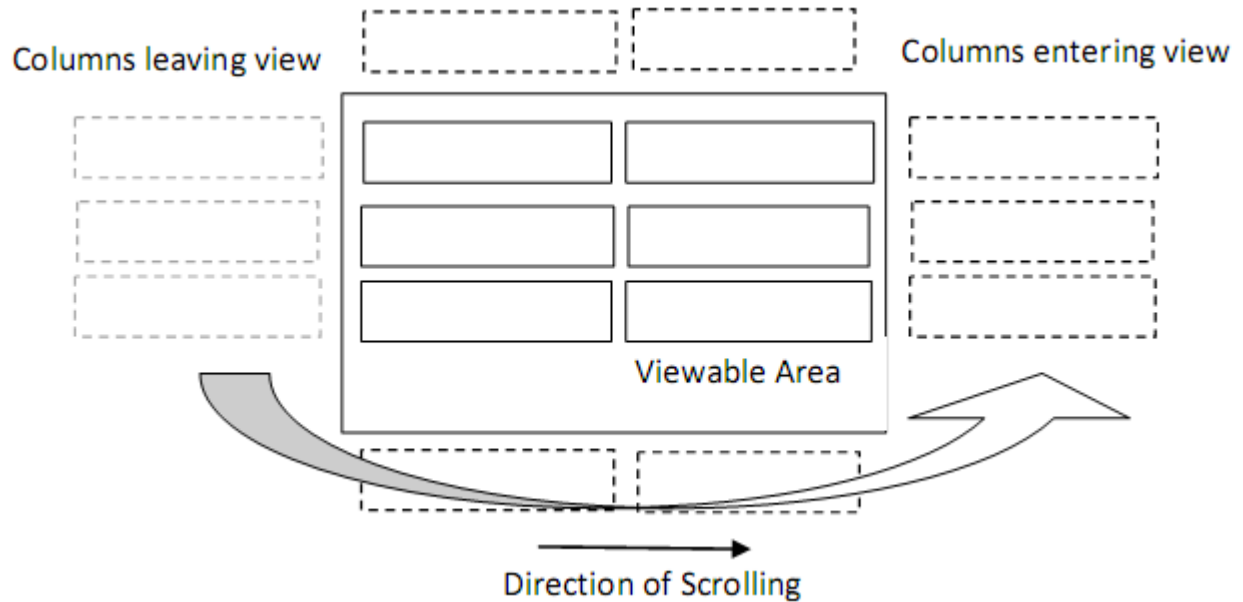
	IG xamGrid™	Competitor I		Competitor II	
Time duration in ms	249	1,849	642% Slower	1,060	325% Slower
Memory Usage Before in KB	18,183	68,790	278% More	20,457	12% More
Memory Usage After in KB	18,045	68,790	281% More	19,982	10% More

Achieving Amazing Performance

So how do we achieve such amazing performance in our xamGrid?
The simple answer – a rock solid architecture that leverages UI Virtualization for almost every user interaction operation. UI Virtualization allows the xamGrid to create the minimum set of UI objects necessary to render the grids viewable area. Additionally, our UI Virtualization infrastructure not only reduces the number of UI elements needed by the xamGrid, but it also recycles UI objects when the viewable area changes, vastly reducing the number of object instantiation and garbage collection operations the control must perform. For large data sets with hundreds or thousands of rows and tens or hundreds of columns these object operations can have a significant impact on the performance of the application.

In the figure below you can see how both the xamGrid's UI Virtualization infrastructure actually reuses both Row and Column UI elements as they scroll out of the visible area of xamGrid.

Columns leaving view

Columns entering view

Viewable Area

Direction of Scrolling

When working with data sets of any size, UI Virtualization is the key to an exceptional user experience.

Understanding the Visual Tree

Silverlight uses a concept called a Visual Tree to track all of the visual elements in an application. Any object that offers a visual representation (anything that derives from FrameworkElement) must be added to the Visual Tree in order to have Silverlight render it. UI elements can have other UI elements as children, thus forming a tree-like hierarchical structure of elements that Silverlight renders to the screen. As you can imagine for all but the simplest applications the number of nodes (or UI elements) in the Visual Tree can grow very large.

As an example, let's take a look at the main screen of the NetAdvantage for Silverlight samples application:

This single screen loads over 800 individual UI elements into the application's visual tree. As you can imagine, when you start to add data-bound controls that repeat themselves like ListBoxes and DataGrids, it's quite easy to quickly get several thousand UI elements in the applications visual tree.

Allowing the Visual Tree to grow too large, or causing a lot of churn within the tree by adding and removing tree nodes, can cause Silverlight's rendering performance to decline. The xamGrid leverages its UI Virtualization to keep an extremely lightweight UI element footprint, adding only the UI elements that are visible to the end user to the Visual Tree.

The ROI of Choosing the Infragistics xamGrid

In order to get an idea of what the overall ROI (return on investment) is for a high performing grid in a typical line of business application, let's take to a look at the normal operations an end user might experience during the daily use of the applications you build. Imagine that a call center uses your application and each employee process a customer for 5 minutes. Usually, in order to process a customer, a call center application needs to do following operations:

- Load data (binding)
- Search
- Sorting
- Scrolling
- Add new record

- Modify record
- Filtering of records
- Paging
- Etc …

Let's compare the xamGrid against "Competitor I" and "Competitor II".

Table 9: ROI of choosing xamGrid in a Silverlight line of business application.

Duration in ms	IG xamGrid	Competitor I		Competitor II	
Observable Collection Binding	1,028	2,025	96% Slower	1,962	90% Slower
Numeric Sorting	1,968	2,818	43% Slower	7,890	300% Slower
String Sorting	7,293	8,038	10% Slower	9,598	31% Slower
Real-time vertical scrolling	883	1,448	64% Slower	*	
Deferred vertical scrolling	750	751	~ Equal	*	
Binding to hierarchical data	950	1,504	58% Slower	6,486	582% Slower
Expand All	1,398	46,466	3223% Slower	3,217	130% Slower
Collapse All	249	1,849	642% Slower	1,060	325% Slower
TOTAL	**14,519**	**64,898**	**346% Slower**	*****	
TOTAL without scrolling*	**12,886**	**62,699**	**386% Slower**	**30,213**	**134% Slower**

* It seems that "Competitor II" has issue with scrolling because thumb cannot be moved to the end.

Using xamGrid you could save 20 seconds per operation. At the end of the day the call center will have a minimum of 10% increase in productivity, which 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 architectural foundation based on a virtualized element behavior. 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 xamGrid, 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 xamGrid today, download the NetAdvantage for Silverlight product at http://www.infragistics.com/downloads