June 2009

Building Low Latency Applications for Financial Markets

Rapid Addition leverages Microsoft .NET 3.5 Framework™ to build ultra-low latency FIX and FAST processing

Applies to: FIX and FAST message processing Low Latency Financial Application Architecture, Microsoft .NET 3.5 Framework™

Kevin Houstoun
Rapid Addition Limited

Ed Briggs
Microsoft Corporation

Summary

This whitepaper describes how Rapid Addition built their ultra low latency FIX and FAST message processing software using the Microsoft .NET 3.5 Framework. By following a disciplined design and development, Rapid Addition was able to meet stringent latency requirements while retaining the advantages that managed code brings.

Contents

Introduction

Latency matters. And Rapid Addition, the leading supplier of front office messaging components to the global financial services industry knows this better than most. Their founder and chairman, Kevin Houstoun, has been one of the leading innovators in this space since helping bring the FIX Protocol to Europe in the 1990's. "Being a few microseconds slower than their competitors can literally cost our clients millions of pounds, dollars or yen. It means the difference between a high speed arbitrage trade being profitable or a waste of time and money; for a hedge fund, it means the difference between posting an updated price and being hit on a stale price for a market maker; and for an exchange it means being the venue of choice for many legs of various trading strategies. For many of our clients low latency is not an option it is a necessity."

Houstoun has lead the FIX Protocol Limited's (FPL) Global Technical Committee (GTC) through the introduction of a data model behind the collection of protocols the group supports, the introduction of the FAST messaging compression standard, FAST, and the release of 3 versions of FIX targeting the exchange to sell side communication. FAST stands for FIX Adapted for Streaming Transport; it was a response to growing market data volumes and the exchange communities desire to avoid inventing further costly proprietary protocols. FAST is an open specification that was developed by FPL with financial support from Archipelago Exchange, the Chicago Mercantile Exchange, the International Securities Exchange, London Stock Exchange, Microsoft and the Singapore Stock Exchange.

In 2003 Houstoun left Salomon Brothers, then part of Citigroup, and teamed up with Clive Browning, to write components based on FPL standards utilizing the recently introduced repository. They had a vision of using the FPL data model to write better performing and easier to use components.

Motivation for Using .NET

Conventional wisdom has been developing low latency messaging technology required the use of unmanaged C++ or assembly language. But RA saw advantages to building this sort of technology in managed code. Asked about this choice Clive Browning, Rapid Addition's Chief Technology Officer said, “When you look at the unmanaged C++ solutions, you see that the approach the best of breed solutions use is to develop their own specialized engine and then generate dedicated handlers for each pattern that uses its specialized engine. Our approach is actually very similar to this except that our specialist engine is a sub set of Microsoft’s .Net CLR.” Browning continues, “This gives our clients certain advantages, we don’t have to update our engine for every hardware change, Microsoft does that for us; we have the full set of .NET features available for other modes of operation such as start up, and of course there is no overhead for communicating between the managed and unmanaged code if our end customers are using .NET for other parts of their project.”

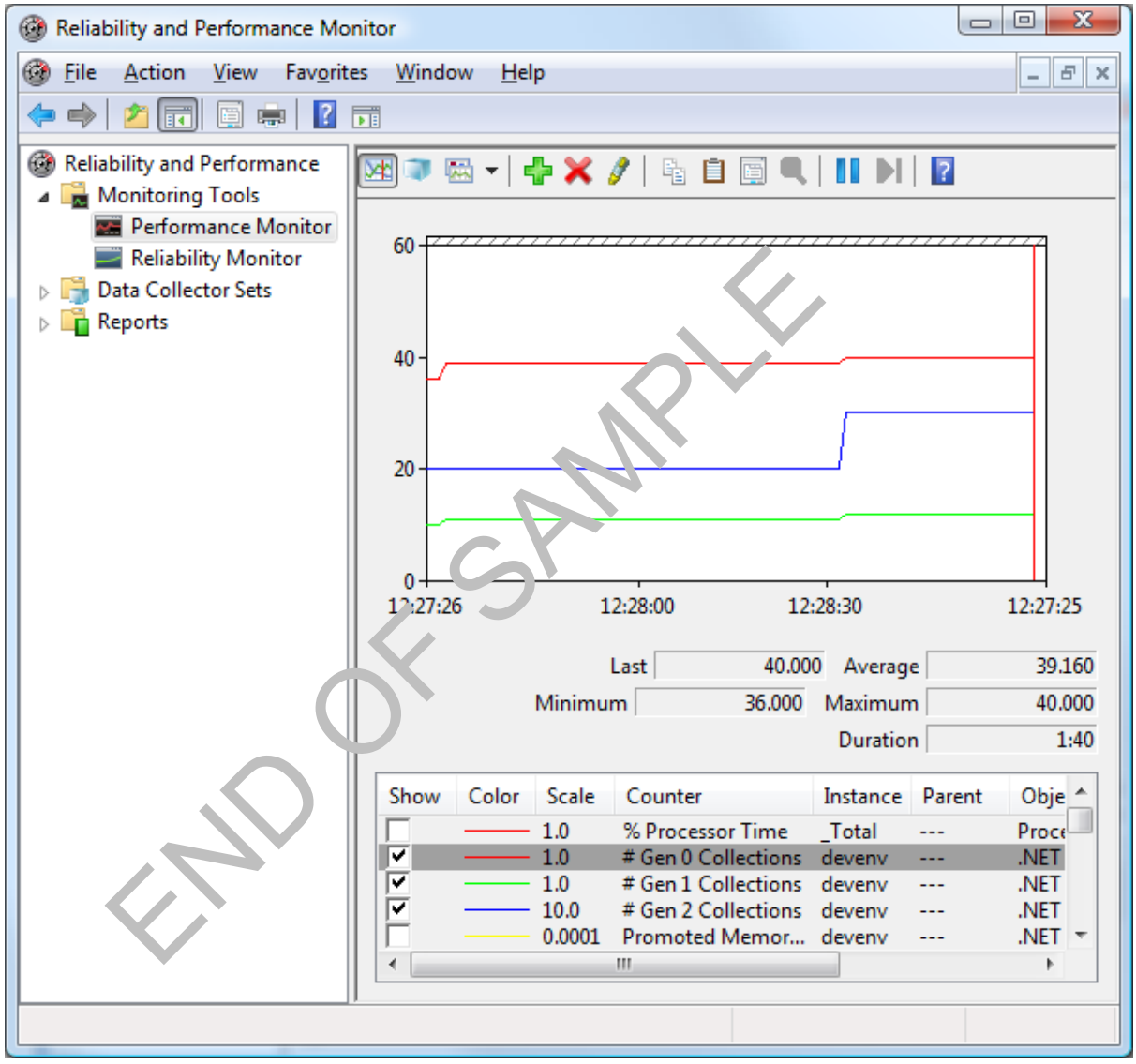
To meet the demanding latency requirements inherent in FIX and FAST message processing, there are two overriding rules Rapid Addition employs in their designs:

1) Actively manage any resources that are used in the steady state operation of the program. This is achieved through the use of resource pools. A number of resources are assigned in the startup phase and then these are recycled throughout the continuous operations phase.
2) Do not cause garbage collection in the continuous operation phase of the system.

From an overall design perspective when designing low latency systems in .NET, Rapid Addition adopts a number of disciplines:

1) Structure code into three distinct phases; startup, continuous operations and shut down sections. Code in the start-up is allow to make memory allocation both for use in resource pools and temporarily to initiate the process but at the end of the start-up mode the garbage collector is invoked to ensure that any unreferenced memory is released at this point. In the continuous operations phase no memory allocation is tolerated and in the shutdown phase the garbage collector is allowed to become active again.

Figure 1 below illustrates this, the program is started around 12:27:26 and we see some GC activity related to start-up, it runs in continuous mode sending 40,000 messages per second until 12:28:30 when it is shut down, as part of the shut down the resource pools are released and we see the Garbage Collector activity associated with cleaning up these resources. The important thing to note is that *there is no Garbage Collector activity* in the continuous operation phase. In live client systems this Garbage Collector free phase can be 10's or 100's of hours.



2) Tight coding standards and guidelines ensure that once in the continuous operating mode garbage is not created. Apart from the obvious approaches such as avoiding manipulating immutable objects, principle the .Net string data type, it also involves avoiding parts of the .Net runtime that Rapid Addition know create garbage.