USING C# AND WPF TO CREATE FAST PLOTS FOR
TELEMETRY ANALYSIS ON LARGE DATA SETS

By Steven Burns and William Endress
Raytheon Missile Systems

ABSTRACT
Upon completion of a test where telemetry(TM) data was collected, the resulting TM file will usually contain millions of data points. Traditionally MatLab™, Mathlab™, or some third party software is used to plot the data. These methods may not always be desirable due to the expense of licensing, restrictions on the ability to create custom graphs and the inability to quickly plot large amounts of data. These problems were solved by using Windows Presentation Foundation (WPF) graphics capabilities in conjunction with C# to develop a unique set of algorithms to display custom graphs of unlimited size with quick response.

Keywords: Telemetry, Data Analysis, MatLab, C#, WPF, Fast Plotting

INTRODUCTION
When testing a unit with TM output, whether during the design, manufacturing, field testing or pre delivery tests, a large amount of raw data is generated. Upon completion of a test, the resulting data file will usually contain millions of raw values that were generated by sensors, processors and transducers of the unit under test. Transforming the collected raw data into data points that are plotted out in a series of graphs has traditionally been accomplished by using MatLab™, Mathlab™, or some third party software. These traditional methods may not always be desirable due to the expense of licensing, restrictions on the ability to create custom graphs and the inability to quickly plot large amounts of data. This paper discusses the advantages of using Windows Presentation Foundation (WPF) graphics capabilities in conjunction with C# to
input, process and present the data. In addition it will cover the algorithms used to plot large data sets of unlimited size with fast display times.

SuperTM charting was developed using the .NET Framework 4.0 and the Windows Presentation Foundation (WPF). The .NET Framework consists of the Common Language Runtime (CLR) and the Framework Class Library (FCL). The CLR manages memory, thread execution, code execution, code safety verification, compilation and other system services. The FCL is a collection of reusable types that tightly integrate with the CLR. The FCL types include objects to accomplish a range of common programming tasks, including string management, data collection, database connectivity, file access and WPF. (Microsoft reference 1 & 2)

**INTRODUCTION TO WPF**

WPF is a next generation presentation system for building windows client applications. WPF was first released in the .NET Framework 3.0 (2006) and is included in every subsequent .NET Framework release. At the core of WPF is a resolution-independent and vector-based rendering engine that performs hardware graphics acceleration on the Graphics Processing Unit (GPU) to minimize CPU usage. WPF applications also start out with 2 threads, one for rendering and one for managing the user interface increasing the User Interface (UI) responsiveness. On computers using DirectX 9 or higher, the combination of using the GPU for rendering and the 2 threaded GUI result in high performance presentation capabilities. WPF is part of the Microsoft .NET Framework and can be used with C# and the other .NET languages.

WPF’s vector based graphics allows the graphics to scale according to screen resolution without losing image quality. WPF uses a resolution independent layout, with all dimensions specified using device independent pixels. Each resolution independent pixel is 1/96th of an inch; this allows the graphics to look similar when rendered to any device. WPF’s layout is dynamic, and the UI elements arrange themselves according to their content, parent layout container and the available screen area. This dynamic layout automatically adjusts the size and position of the UI elements when the size and position of their contents change.

WPF separates the graphics portion of the UI into Extensible Application Markup Language (xaml) which allows for user interface elements to be parsed and manipulated at design time or runtime and a partial class that contains the code behind. Xaml uses event handlers to communicate with the code behind. WPF also allows the UI elements to be developed in code.

**HOW C# AND WPF WAS USED FOR FAST PLOTTING OF TELEMETRY DATA**

The following contains the definitions and descriptions of the classes used for the building and presentation of the plot.

**Polyline**

**DataSeries**

The DataSeries class contains all the information for a single line plot. Each line plot will have its own DataSeries. In this container is the complete set of data points, along with their corresponding x axis values (time - such as IRIG, counts or other x axis data types). It also contains a structure to contain the actual data points used in the plot. A stack structure is implemented that contains the starting and ending addresses of the plotted line for use in zooming in or zooming out. This class creates a Polyline object for a given DataSeries. It then defines the line style for the Polyline, including the line color, line thickness, line pattern and series name. The SeriesName property is used in creating the legend for the chart. The line pattern is defined by a public enumeration called LinePatternEnum, in which five line patterns are defined including Solid(default), Dash, Dot, Dash-Dot and None. The None enumeration means there will be no line drawn (Xu, 2009, p. 169).

**DataCollection**

The DataCollection class contains a number of DataSeries objects, one DataSeries for each line to be plotted. For each DataSeries, a line is added to the chart canvas using the specified line style for that data series (Xu, 2009, p. 171). Also included is the maximum and minimum value of the DataSeries with the largest values. This is used in ChartStyle to appropriately size the graph.

**ChartStyle**

ChartStyle.cs is a C# class modeled from Jack Xu’s book *Practical WPF Charts and Graphics* (Xu, 2009, pp. 166-167). It is used to define all the chart layout information and is the base class for ChartStyleGridlines. It consists of an instance of the chartCanvas, the minimum and maximum values for the x and y axis and the NormalizePoint(Point pt) method. NormalizePoint(Point pt) uses the width and height of chartCanvas to convert the pixels from the real world coordinate system to device independent pixels. ChartStyle.cs also contains a method called OptimalSpacing(double original) which is used to determine the optimal space between tick marks on the axis (Xu, 2009, p. 305).

**ChartStyleGridlines**

ChartStyleGridlines is also modeled after the ChartStyleGridlines class in *Practical WPF Charts and Graphics* (Xu, 2009, pp. 174-178). It is derived from ChartStyle and it creates the gridlines, labels and ticks including the color and placement on the chart. To produce the gridlines ChartStyleGridlines relies on two methods, AddChartStyle() and AddLinePattern() and an Enum called GridLinePatternEnum.
AddChartStyle() does all the axis and tick setup and labeling of the gridlines for the plot.

AddLinePattern() takes the gridline and converts it to a pattern of GridLinePatternEnum (Solid, Dash, Dot, DashDot).

**TMChart**

TMChart is the main class used for creating the plot. It is made up of a .xaml file that is used to layout the WPF controls and a .cs file that holds the C# code behind. The .XAML file and the .cs file together make up the WPF form.

The structure of TMChart.xaml consists of an overall Grid control that forms the layout for several Label and TextBlock controls and an inner Grid control named chartGrid which is the layout control for the plot. The chartGrid control connects the SizeChanged event to its handler in the code behind. ChartGrid contains a Canvas control named textCanvas that serves as the outer area of the plot and is used for labeling the plot. TextCanvas has a child Canvas control called chartCanvas and it is the main control that is used to display the plot. Besides displaying the plot, chartCanvas also has a 2 menu controls as children, Change Color menu and Add Symbol menu.

TMChart.xaml.cs, the code behind for TMChart, requires an instance of the following classes on construction, ChartStyleGridlines, DataCollection and DataSeries. The method AddChart is used to draw each plot line. AddChart processes the data by looping through each DataSeries in the DataCollection. For each data series a polyline is created by combining a telemetry parameter (y) and its respective time stamp (x) to create a data point. The data point is normalized using ChartStyles’ NormalizePoint method and then it is added to a polyline. Once all the required points are added to the polyline, the polyline is added as a child to ChartStyleGridlines’ ChartCanvas and this in turn triggers a rendering of the UI causing the plot to be drawn in the ChartCanvas. In addition to creating the plot lines AddChart calculates the maximum number of points that can be plotted in pixel area of the plot.

**Speeding up graphs**

Even using the speed enhancements of WPF, plotting large numbers of data points can be very time consuming. One way to increase the speed at which plots are produced is to reduce the number of points to be plotted. For example, suppose the user needs to plot 8 million data points on a canvas that is 1,000 independent pixels (1,000 x 1/96th inch) wide. Plotting all 8 million points would result in painfully long wait times. On the other hand, plotting a polyline of only 1,000 points is very fast. So the question exists which data points should be used to do the plotting? One way to determine which data points to plot would be to divide the number of data points by the canvas width then we could plot every 8,000th data point. Doing so produces a nice curve approximation of the data present. But this does not account for any data outliers that would be of interest. A better representation of the data results is to make consecutive groups of
8,000 data points, getting the minimum and maximum value of each group, and then plotting them against the same x axis value. This will result in plotting 2,000 points per group which is still very fast. While skipping over most of the 8,000 data points will throw out a lot of data, it will produce a plot that will show areas of interest (outliers) that the user may wish to zoom in on. When the user selects the area to zoom in on, the number of data points now under consideration is reduced by one or more orders of magnitude. Eventually the user will zoom in far enough that all data points will be displayed.

The code fragment in Figure 1 demonstrates the portion of AddChart that controls the data points to be displayed:
The main routine that adds plots to a chart
startAddr is the starting address of the data to be plotted.
In the beginning it is zero. This number will change when zooming in on data
to reflect the starting address of the zoom.
endAddr is the ending address of the data and at the start is the last data
item. This will also change during zooming.
This is just a fragment of the AddChart routine demonstrating the selection
of a subset of data to be plotted

public void AddChart(startAddr, endAddr)
{
    int count;
    int timeCount;

    //Gets the total number of data points to plot
    numRecords = endAddr - startAddr;

    //Calculates the number of data points between points to be plotted
    //based on the screen size.
    sizingFactor = numRecords / chartCanvas.ActualWidth;

    for(int pointNumber = startAddr; pointNumber < endAddr; pointNumber += sizingFactor)
    {
        //MIN_RANGE is a number smaller than the smallest value expected in the data set
        //MAX_RANGE is a constant number larger than the largest value expected
        tempXmax = MIN_RANGE;
        tempXmin = MAX_RANGE;

        //Loops thru the data group finding the largest and smallest value.
        For(count = 0; count < sizingFactor; count++)
        {
            //RawData is a data structure that holds the complete set of
            //data points in the data set
            xAns = ds.RawData[pointNumber + count];
            if(xAns > tempXmax)
            {
                tempXmax = xAns;
            }
            if(xAns < tempXmin)
            {
                tempXmax = xAns;
            }
        }

        //This is the value used for the x axis in counts. A more useful value
        //might be the IRIG time for the selected data point.
        timeCount = pointNumber + count - 1);

        //LineSeries is a Polyline. This adds a point to the polyline
        ds.LineSeries.Points.Add(new Point(tempXmax, timeCount));

        //The following two statements add the data that was actually used
        //to the plottedData data structure
        ds.plottedData.Data.Add(tempXmax);
        ds.plottedData.Time.Add(timeCount);
    }

    //The following two statements add the start and end data address
    //in RawData. Used for keeping track of start and end points
    //when zooming in and out
    ds.StartAddr.Add(startAddr);
    ds.EndAddr.Add(endAddr);

    //Add the data series to the data list
dc.DataList.Add(ds);
}
Conclusion

When needing to produce plots from millions of connected data points, the traditional applications MatLab™, Mathlab ™ have the expense of licensing and the inherent limitations of plotting large volumes of data. Using WPF’s graphics capability combined with an algorithm that limits the actual number of points plotted produces a fast full featured charting capability, that is inexpensive, responsive and allows for customization.


REFERENCES

