

Election 2012: Real-Time Monitoring of Election Results

A simulation using the Syncfusion PivotGrid control.

by Clay Burch and Suriya Prakasam R.

Contents

- Introduction 3
- Ticking Pivots..... 3
 - Background Information..... 3
- Electoral College..... 4
- Sample Overview..... 4
- The Data 5
- The Real-Time PivotGrid Control 7
- The Real-Time Simulation..... 8
- Another Way to Handle Underlying Data Changes 9
 - Running the Sample..... 10
- Sample in Action 11
- In Summary 12

Introduction

In this paper, we discuss how to display dynamically changing data in a Syncfusion PivotGrid control using the United States presidential election as a practical sample.

Ticking Pivots

The initial part of this paper shows two approaches to display dynamically changing information in a PivotGrid control. The first technique is to have the control automatically respond to changes in the underlying data. In order for this automatic response to occur, there are requirements that the data must implement in the form of interfaces. The second technique is to periodically repopulate the control. In this case, the data is not required to meet any additional requirements, but the technique does impose heavier computational loads so this technique may not be useful in many situations.

We will discuss in detail the steps needed to implement both techniques. We will use a MVVM sample with a WPF window to illustrate them. The window will display a pair of PivotGrid controls: one set up to automatically respond to underlying changes in the data, and the other set up to be completely repopulated through the use of a timer. This paper discusses this sample in detail to illustrate how to support dynamically changing data in a PivotGrid control.

Background Information

It is a presidential election year in the United States with a Republican, Mitt Romney, challenging the incumbent Democrat, Barack Obama, for the presidency. Our sample will simulate the voting results dynamically tallied on election night. You might think that this election would be determined by counting the votes as they come in and then seeing who has the most votes after all votes have been tallied. This type of election is referred to as a popular election because the most *popular* candidate wins. But this is not the case with the U.S. presidential election. The person with the most votes may not be the winner. In fact, in 2000, Al Gore won the popular vote, but George W. Bush was elected president.

Electoral College

The president of the United States is elected by a group of people who are members of the Electoral College. Each of the fifty states plus the District of Columbia elects members to this Electoral College. Currently, there are 538 members. The number of members from each state is the same as the state's representatives in the U.S. Congress. Since seats in the U.S. House of Representatives are allocated based on population, the membership in the Electoral College is also based on population to a certain degree. However, each state has the same number of representatives in the Senate, causing membership in the Electoral College to lean in favor of less populous states over states with greater populations. For example, Montana elects 3 members to the Electoral College for its population of about 989,000, or 330,000 per Electoral College member. California has a population of 37.2 million people with 55 members of the Electoral College, or 676,000 per member.¹

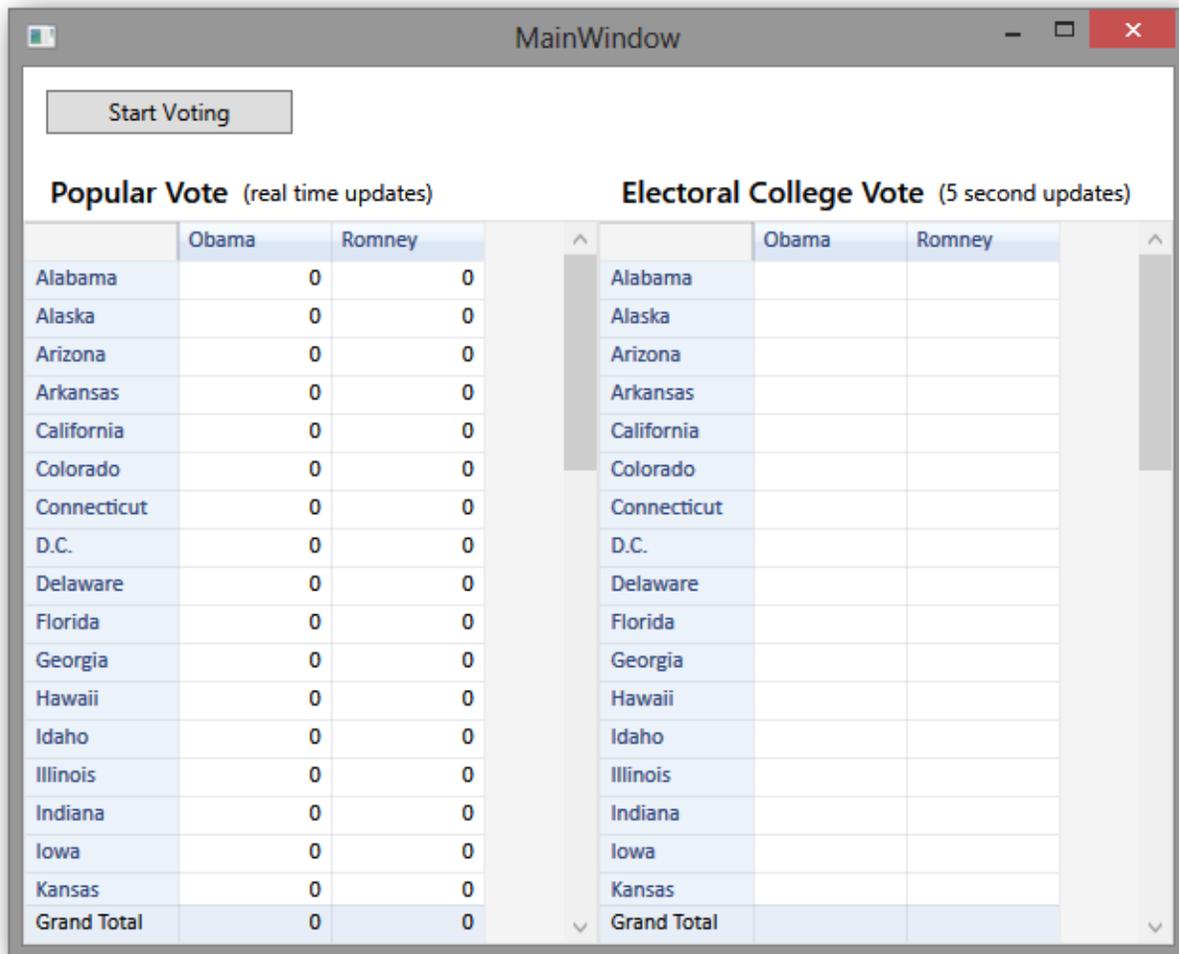
On a state by state basis, the popular vote in a state *usually* determines who that state's Electoral College members will vote for. Normally, it is an all-or-nothing allocation of the member's votes. Only two states, Maine and Nebraska, require their Electoral College members to proportionally divide their votes based on the popular vote within their state.

To be elected president, a candidate must receive at least 270 votes in the Electoral College. This process of electing the president through the Electoral College dates back to the beginning of the United States. The process is set forth in Article Two of the U.S. Constitution.

Sample Overview

The sample we create has two PivotGrid controls. One displays the popular vote results, and the other shows the Electoral College results based on the popular vote. The information being processed simulates receiving the vote count for each candidate at a particular voting site in a state in real time as the results become available. This is the information that will be processed by the pivot grid shown on the left side of the following window.

¹ Population statistics based on 2010 U.S. Census data.



The pivot grid on the right will show the Electoral College results as the simulation runs. It is updated once every five seconds, whereas the pivot grid on the left is updated in real time—twice every 50 milliseconds.

The Data

The following code sample shows the underlying business object that is being processed by the real-time pivot grid. This class is in the project's **Model** folder.

```
public class VoteRecord : INotifyPropertyChanged, INotifyPropertyChanging
{
    private string state = string.Empty;
    public string State
    {
        get { return state; }
    }
}
```

```

        set { OnPropertyChanging("State"); state = value; OnPropertyChanged("State"); }
    }

    private string candidate = string.Empty;
    public string Candidate
    {
        get { return candidate; }
        set { OnPropertyChanging("Candidate"); candidate = value; OnPropertyChanged("Candidate"); }
    }

    private int votes = 0;
    public int Votes
    {
        get { return votes; }
        set { OnPropertyChanging("Votes"); votes = value; OnPropertyChanged("Votes"); }
    }

    public event PropertyChangedEventHandler PropertyChanged;
    protected virtual void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    public event PropertyChangingEventHandler PropertyChanging;
    protected virtual void OnPropertyChanging(string propertyName)
    {
        if (PropertyChanging != null)
        {
            PropertyChanging(this, new PropertyChangingEventArgs(propertyName));
        }
    }
}

```

A [VoteRecord](#) records the state, the candidate, and the candidate's vote count. This represents the information that might be generated for one candidate from one voting site. Each site would submit two such records—one for each candidate. These records are collected into an [ObservableCollection<VoteRecord>](#), and it is this collection that is driving the real-time pivot grid. Notice that this [VoteRecord](#) object implements both [INotifyPropertyChanged](#) and [INotifyPropertyChanging](#) interfaces. These are the two interfaces that your business object must implement so that the Syncfusion PivotGrid control can automatically respond to changes in the underlying data.

The reason for the two interface requirements is that when a change occurs, the underlying pivot engine tries to "adjust" each summary, as opposed to re-computing each summary from scratch. For example, to adjust a summation, you simply have to subtract the old value and add the new value to get the properly adjusted sum. This means the pivot engine needs the previous value as well as the new value to make these efficient adjustments. This is the reason for requiring both a changing event and a changed event.

The Real-Time PivotGrid Control

Here is the XAML used to declare this PivotGrid control:

```
<syncfusion:PivotGridControl Name="popularPivotGrid"
    ShowGrandTotals="True"
    EnableUpdating="True"
    AutoSizeOption="TotalRows"
    ItemSource="{Binding PopularRecords}"
    ShowGroupingBar="False">

    <interactions:Interaction.Behaviors>
        <behavior:PivotGridLoadedBehavior />
    </interactions:Interaction.Behaviors>

    <syncfusion:PivotGridControl.PivotRows>
        <syncfusion:PivotItem FieldMappingName="State" FieldHeader="State"
TotalHeader="Total"/>
    </syncfusion:PivotGridControl.PivotRows>

    <syncfusion:PivotGridControl.PivotColumns>
        <syncfusion:PivotItem FieldMappingName="Candidate" FieldHeader="Candidate"
TotalHeader="Total"/>
    </syncfusion:PivotGridControl.PivotColumns>

    <syncfusion:PivotGridControl.PivotCalculations>
        <syncfusion:PivotComputationInfo CalculationName = "Votes" Description =
"Sum of votes"
                                fieldName = "Votes" Format = "#,##0"
SummaryType="DoubleTotalSum"/>
    </syncfusion:PivotGridControl.PivotCalculations>
</syncfusion:PivotGridControl>
```

Notice the `EnableUpdating="True"` setting. This is the property that must be set to tell the PivotGrid control to monitor the underlying data for changes so it can automatically adjust its display to reflect them. Additionally, the list holding these business objects must implement change notifications. The Syncfusion PivotGrid control recognizes change events from ObservableCollections and lists that implement IBindingList, as well as DataTables and DataViews. In our case, `PopularRecords` is an `ObservableCollection<VoteRecord>`, which is part of the ViewModel and is bound through the DataContext.

To summarize, in order for the PivotGrid control to automatically respond to changes in the underlying data, these three things must occur:

- The business object must implement `INotifyPropertyChanging` and `INotifyPropertyChanged`.
- The underlying list must be an `ObservableCollection`, an `IBindingList`, a `DataTable`, or a `DataView`.
- You must set the `EnableUpdating` property of the `PivotGrid` control to `True`.

The Real-Time Simulation

As voting occurs, the `ItemsSource` of the `PivotGrid` control has a new `VoteRecord` added to it every 50 milliseconds. This work is being done using a `Task` so the process of actually adding votes will not affect the UI thread directly. Here is the code being used to manage this work. It is part of a command associated with the button that initiates the updating.

```
//Use a Task to update the vote count off the UI thread.
this.cancelVote = new CancellationTokenSource();
Task streamVotes = Task.Factory.StartNew(new Action(() =>
{
    while (true)
    {
        //Add 2 new vote counts every 50 milliseconds.
        for (int i = 0; i < 2; ++i)
        {
            this.data.AddAVoteRecord();
        }

        if (cancelVote.IsCancellationRequested)
        {
            break;
        }
        Thread.Sleep(50);
    }
}), cancelVote.Token);
```

Even though this business object updating is being done off the UI thread in the `PivotGrid` control change event handlers, the `PivotGrid` control must invoke its code back onto the UI thread since it has to interact with the UI to refresh itself properly. The actual refreshing of the `PivotGrid` control objects is done on the UI thread. The goal here is to do things in an efficient way so that the UI is not bogged down by the `PivotGrid` control automatically handling these updates.

To make this sample more realistic, the number of votes being added with each update has been weighted to reflect the population of the state where the voting occurred. This means the vote tally for a populous state like California will be higher than the vote tally for a less populous state like Montana. This will make the popular vote simulation more realistic. We do not try to quantify extra information like the percentage of Republicans or Democrats in a particular state. You can watch the actual election results for that level of realism.

Another Way to Handle Underlying Data Changes

In our vote counter, the PivotGrid control on the right is not being updated in real time. Instead, an update is explicitly triggered every five seconds as part of the command associated with the UI button that starts the updates. This pivot grid shows the Electoral College vote based on the popular vote in the real-time pivot grid. The business object being used for the pivot grid on the right is again a `VoteRecord`. But this time, instead of each record holding a vote count for some polling place in a state, it holds the vote count for the entire state as reflected in the summary calculations in the popular vote pivot grid. So, every 5 seconds, a new data source is created to reflect the content of the left pivot grid at that point in time, and this information is then processed and displayed in the pivot grid on the right. This process will work even if the business object does not implement the change interfaces, and the list can be any `IEnumerable` collection.

Here is the code that queries the information in the pivot grid on the left, and then constructs a vote count to reflect the Electoral College process with the candidate leading the popular vote in a state receiving all the electoral votes for that state.

```
private static VoteRecords GetElectoralCollegeResults(PivotEngine engine)
{
    VoteRecords data = new VoteRecords();

    lock (engine)
    {
        for (int i = 1; i < engine.RowCount - 2; ++i)
        {
            //See which candidate has the most votes, give him all the electoral votes from that state.
            double d1 = engine[i, 1].DoubleValue;
            double d2 = engine[i, 2].DoubleValue;
            string candidate = (d1 > d2) ? data.candidates[0] : ((d1 < d2) ? data.candidates[1] : "");
            if (candidate.Length > 0)
            {
                data.Add(new VoteRecord()
                {
                    Candidate = candidate,
                    State = data.states[engine[i, 0].FormattedText].Name,
                    Votes = data.states[engine[i, 0].FormattedText].Delegates
                });
            }
        }
    }
}
```

```

    }
  }
  return data;
}

```

The sample does not try to handle the two states that cast their electoral votes in proportion with the popular vote.

Running the Sample

When you run the sample and start the voting, you will see something similar to the following window:

The screenshot shows a window titled 'MainWindow' with a 'Start Voting' button. Below the button are two tables. The first table, 'Popular Vote (real time updates)', has columns for 'Obama' and 'Romney' and rows for each state plus a 'Grand Total' row. The second table, 'Electoral College Vote (5 second updates)', has columns for 'Obama' and 'Romney' and rows for each state plus a 'Grand Total' row. The 'Grand Total' row is visible in both tables, even though it is below the bottom-most row of the grid.

Popular Vote (real time updates)			Electoral College Vote (5 second updates)		
	Obama	Romney		Obama	Romney
Alabama	92,004	41,820	Alabama	9	
Alaska	22,304	16,728	Alaska	3	
Arizona	112,453	143,122	Arizona		11
Arkansas	44,608	50,184	Arkansas		6
California	562,265	153,345	California	55	
Colorado	66,912	50,184	Colorado	9	
Connecticut	65,050	65,050	Connecticut		
D.C.	13,940	16,728	D.C.		3
Delaware	25,092	30,668	Delaware		3
Florida	161,706	242,559	Florida		29
Georgia	89,214	148,690	Georgia		16
Hawaii	26,019	11,151	Hawaii	4	
Idaho	18,585	22,302	Idaho		4
Illinois	260,218	148,696	Illinois	20	
Indiana	30,669	122,676	Indiana		11
Iowa	16,728	27,880	Iowa		6
Kansas	50,184	22,304	Kansas	6	
Grand Total	4,005,423	3,539,805	Grand Total	232	276

You will notice that the grand total row is displayed even though the bottom row of the pivot grid is not visible. Additionally, you will notice that the grand total column that is normally on

the right side of the pivot grid is missing. The PivotGrid control has a property that you can set to show or hide the grand total cells. However, this property is just like the Electoral College: it is either all or nothing. There is no setting to show only the grand total row and hide its column counterpart.

So, how do we make the PivotGrid control show only the grand total row since the grand total column isn't of much use in this situation? Well, you have to remember that the grid part of the Syncfusion PivotGrid control is actually the Syncfusion Grid control, and you have access to most of the functionality of this control through the `PivotGridControl.InternalGrid` property. The following code allows the grand total row to always be visible while hiding the grand total column. It works by setting properties on the `InternalGrid`. The Behavior shown in the previous XAML sample allows access to

```
void AssociatedObject_Loaded(object sender, System.Windows.RoutedEventArgs e)
{
    if (this.AssociatedObject.InternalGrid != null)
    {
        //Hides the column grand totals.
        this.AssociatedObject.InternalGrid.Model.ColumnCount -= 1;

        //Allows the bottom grand total to always be visible.
        this.AssociatedObject.InternalGrid.Model.FooterRows = 1;
    }
}
```

There is a slight technical issue in trying to apply these settings to the Electoral College pivot grid every time the data is regenerated. To handle this issue, for the Electoral College pivot grid, the sample listens to the `PivotEngine.PivotSchemaChanged` event to apply these settings. This work is also done in the Behavior class.

Sample in Action

You can download the sample code from https://bitbucket.org/syncfusion/pivotgrid_realtime_sample and run it using Visual Studio to see how well the PivotGrid control performs in this real-time updating scenario. It behaves well without the UI becoming sluggish.

A video of the sample in action is also available with the code.

Note: With the sample, you may receive the following exception when you try to open the window in the Visual Studio designer:



This issue occurs if you download the sample from Git as a zip file and the included DLLs are blocked as a result.

If you do run into this, follow these steps to unblock a downloaded DLL which will allow the designer to cleanly load the window:

1. Use Windows Explorer to navigate to **(SampleInstallFolder)\TickingPivot-MVVM\Binaries**.
2. Right-click the **System.Windows.Interactivity.dll** file and open its properties.
3. Click the **Unblock** button at the bottom of the properties window.
4. Rebuild the project.

In Summary

United States presidential elections are not determined directly by the popular vote. Instead, the winner is determined by a process involving an Electoral College that is prescribed by the U.S. Constitution. This process amounts to 51 smaller elections on a state-by-state basis with the winner of these elections receiving all of the states' electoral votes. You can model this process using two PivotGrid controls that are updated dynamically.

A PivotGrid control can automatically respond to changes in the underlying data provided the business objects in the data implement two interfaces: `INotifyPropertyChanging` and `INotifyPropertyChanged`. This allows the PivotGrid control to display rapidly changing data without bogging down the UI. This technique was used in our popular vote pivot grid to simulate the popular vote results for each state. A second way to update a PivotGrid control is to reload the information from scratch. While not as efficient as the automatic updates, this technique has the advantage of working with any type of `IEnumerable` data. This technique was used in our Electoral College pivot grid to simulate the Electoral College voting process.