ARL-TR-7817 ● SEP 2016

US Army Research Laboratory

EventSlider User Manual

by Christian D Schlesiger

ARL-TR-7817 ● SEP 2016

US Army Research Laboratory

EventSlider User Manual

by Christian D Schlesiger
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* September 2016	**2. REPORT TYPE** Technical Report	**3. DATES COVERED (From - To)** 01/2014–08/2016
4. TITLE AND SUBTITLE EventSlider User Manual		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Christian D Schlesiger		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory ATTN: RDRL-CII-B 2800 Powder Mill Road Adelphi, MD 20783-1138		**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-7817
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The EventSlider is a Windows Presentation Foundation (WPF) control developed using the .NET framework in Microsoft Visual Studio. As a WPF control, it can be used in any WPF application as a graphical visual element. The purpose of the control is to visually display time-related events as vertical lines on a horizontal timescale slider. These events can take any form the developer needs for their specific application. Properties are set on the EventSlider control to instruct the control where to find the time information on these data objects. This report outlines how a developer can use the control and integrate it into an application. An application programming interface reference is described for all of the properties, events, and methods available on the control.

15. SUBJECT TERMS
Windows Presentation Foundation, WPF, control, C#, .NET framework, Microsoft Visual Studio

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Christian D Schlesiger
Unclassified	Unclassified	Unclassified	UU	36	**19b. TELEPHONE NUMBER (Include area code)** 301-394-2473

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

1. Introduction

The EventSlider is a Windows Presentation Foundation (WPF) control developed using the .NET framework in Microsoft Visual Studio. As a WPF control, it can be used in any WPF application as a graphical visual element. The purpose of the control is to visually display time-related events as vertical lines on a horizontal timescale slider. These events can be of any nature as long as there is some property indicating a time for the event. This gives the control great flexibility in the type of data it can represent and how they can be used in an application.

All visual elements of the EventSlider are customizable by the developer incorporating it into an application. Standard WPF styling practices can be applied. Specific elements of the slider, such as the color of the lines and the appearance of the scale and tick marks, can be set by the developer at design time and also changed at runtime, if desired.

During runtime, the control allows a user to move the scale forward or backward in time as well as zoom in and out of different-sized time windows (e.g., from a view of 1 day to down to 1 s). These time windows and zoom intervals are completely customizable by the developer. Behavior of the control during navigation (zooming, sliding, and moving the thumb) can also be changed through properties and events available to the developer.

The control is agnostic to the data presented: the events are presented to the control in generic object form. Properties on the control are set to specify what property on those event objects should be used for the time of that event. The event objects could be straight *DateTime* structures or a more complex data class that has a property defined on it that contains the relevant time.

The EventSlider has the notion of a current event, tracked visually by the position of the thumb on the slider as well as in code with a property. Additionally, there is a notion of a current time as well, as that is usually the time of the current event, but it can be different than the current event's time under certain conditions.

The control keeps constant track of the boundaries and size of the time window being shown through appropriately named properties. User interaction with the control at runtime will change these values, or they can be set directly in code, which will result in appropriate visual changes to the control. The control overrides the methods in the *ISupportInitialize* interface (defined on the *FrameworkElement* parent class), which allows for efficient batch setting of these main properties without changing the display until the initialization is complete.

Figure 1 shows an architectural diagram illustrating a typical scenario where the EventSlider control is incorporated into an application. An application might wish to show a series of time events collected and stored in some data store. These events are retrieved from the store by the application and put into an array or some other type of collection object and assigned to the EventSlider's Events property. The application can change other properties on the EventSlider to control its appearance, which events to show, and how it shows these events. The EventSlider notifies the application when the user interacts with the control.

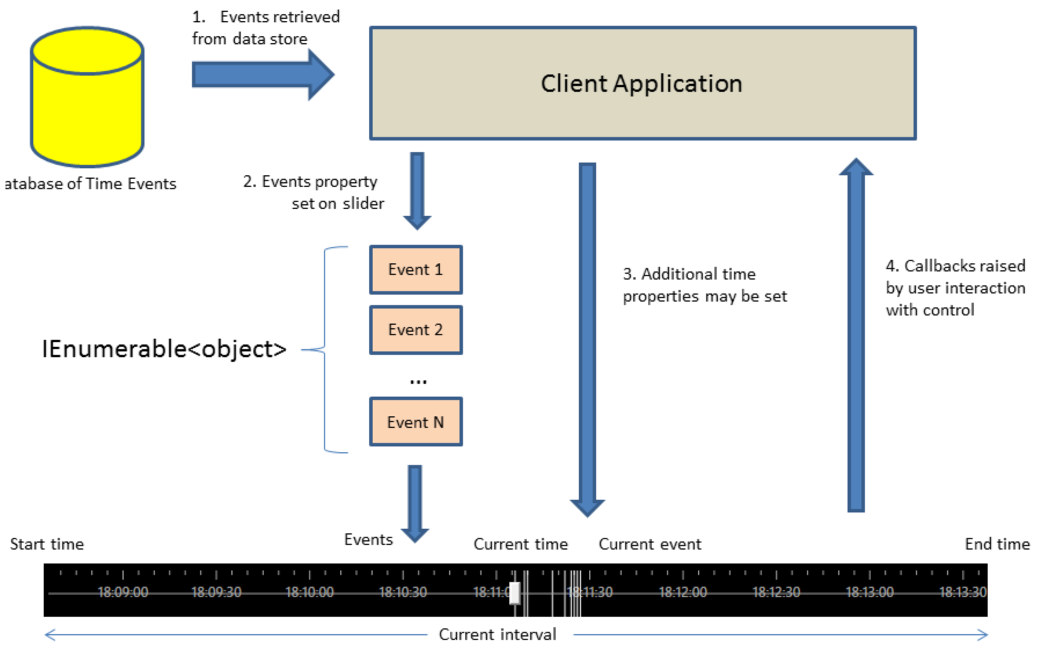

Fig. 1 **Architecture diagram of a client application using the EventSlider**

2. Visual Elements of the EventSlider

Figure 2 shows the EventSlider with some randomly generated time events. These events are shown as orange vertical lines. Across the slider is the current timescale, shown in light gray with tick marks for typical gradations. Time labels are shown at major ticks with smaller tick marks in between. In this example, each minor tick represents 12 h, thus 6 minor ticks are shown between each major tick where a date is shown, which is at 3-day increments. The dark red lines indicate that more than one event is displayed in that area. This is a visual indicator to the user that it would be useful to zoom in on those areas to see the time events more clearly if desired. The thumb has been moved to an event on the date 8/4/16. This indicates that the current event for the control is the one at that date. One could enhance the visual representation of the control further in the user interface by displaying more detail about the current event and its time in other visual elements external to the EventSlider.

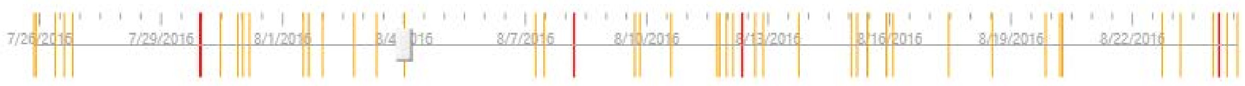

Fig. 2 The EventSlider control showing random time data

The EventSlider automatically chooses the best display for its tick marks and time labels based on the current time window being shown and how much visual space has been allotted for it in the user interface of the application. It recalculates and redraws these ticks and labels each time the zoom is changed on the slider. It also changes these values if the window containing it has been resized. Panning left and right does not change these values but merely scrolls the time and changes the events that are shown appropriately.

3. Manipulation

The EventSlider responds to both the mouse and touch, if a touch screen is available, for manipulation.

To move the currently shown time window to include events in the past, click the mouse or touch with a finger on the slider at any point except on the thumb and drag to the right. This moves the data to the right, revealing more events on the left-hand side of the slider. To show events in the future, drag to the left. This reveals events on the right-hand side of the slider.

To zoom in, hold the mouse cursor over the EventSlider and roll the mouse wheel forward. With touch, use a pinch and spread gesture. To zoom out, roll the mouse wheel backward or use a pinch and narrow gesture. The EventSlider changes the zoom based on preset time intervals. These intervals can be changed at development and at runtime. Because of this, there is a maximum and a minimum zoom that can be reached beyond which further zoom gestures have no effect. Each time the zoom changes, the control redraws all events that are in the current time window along with redrawing the tick marks and labels appropriately.

The current event can be changed by moving the thumb. This can be accomplished with the mouse by left-clicking and dragging on the thumb to the new event. With touch, this is accomplished with a long tap and then drag. By default, the thumb will snap to the nearest event once the mouse or touch is released. This behavior can be changed through a property setting. Because of this snapping behavior, moving the thumb and setting the current event can also occur by the user doing a single click, or a tap, near or on an event line.

4. Library and Namespace

The EventSlider control has been compiled into the following dynamic-link library (DLL):

```
WPFControls.EventSlider.dll
```

It was built using the .NET Framework 4.0 and can be used in any application using that framework version or higher. The control exists in the following namespace:

C#

```
using WPFControls.EventSlider;
```

XAML

```
xmlns:es="clr-
namespace:WPFControls.EventSlider;assembly=WPFControls.EventSlider"
```

Only a single class is exposed in this namespace, the EventSlider class. All methods, properties, and events necessary to use the control reside on this class.

5. Event Data

The data displayed by the EventSlider can be of any type. The only requirement is that they have some sort of property containing a *DateTime* structure or are in fact *DateTime* objects themselves. To be as flexible and agnostic to the data type as possible, event objects are passed to the EventSlider as an *IEnumerable<object>*. This allows the collection of event objects to be of any type as long as it is enumerable. It is assigned to the EventSlider either with a direct assignment or through databinding to the Events property of the EventSlider.

If the event objects are not *DateTime* structures but are instead business model objects, the more likely scenario, then the EventSlider needs to know which property to reference for the time data. This is set through the TimePath property on the EventSlider, where the name of the property on the event object is provided as a string:

C#

```
MyEventSlider.TimePath = "MyTime";
```

XAML

```
<es:EventSlider x:Name="MyEventSlider" TimePath="MyTime"/>
```

The default line color for events is set on the EventSlider through the LineColor property. This property is a WPF *Brush* object. As a WPF *Brush* object, one will commonly use a *SolidColorBrush*, but gradient brushes, for example, could be used for other visual effects. This brush is applied to all events that are visible on the slider. However, if the lines drawn would be so close together as to be indiscernible (usually because of a large zoom interval), the EventSlider draws a single line of a different color that is set with the MergedLineColor property, which has the same syntax as the LineColor property.

Individual events can override this behavior with their own colors. This would be useful in scenarios where events might fall into categories, and a simple color representation would indicate that on the EventSlider. In Fig. 3, some of the events are shown as blue lines. This could indicate a higher importance, for example. The merged line color behavior still overrides this custom color.

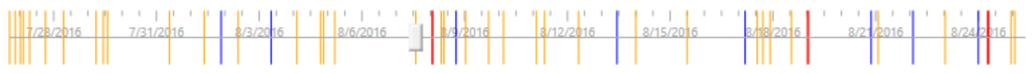

Fig. 3 Custom colors for certain events

To indicate that individual event colors should be used, set the ColorPath property on the EventSlider, which is set in the same way as the TimePath property. This is a string containing the name of the property on the event objects that contains a *Brush* object. Note that once the ColorPath property is set, responsibility for an event's line color is wholly up to the event object. The value in the LineColor property will no longer be used. Therefore, a null value would cause the line to have no color at all and therefore not be visible. This side effect can be exploitable if the application needs to hide event lines temporarily.

Additionally, an individual event's line color can be changed at runtime and be reflected as an immediate change to the EventSlider's visual appearance. This is accomplished by the EventSlider binding the line's color to the event's property that contains the *Brush* indicated by the ColorPath property. For the databinding to be notified of changes to the color, the event object should either use a *DependencyProperty* for the property indicated by the ColorPath or implement *INotifyPropertyChanged* and raise the appropriate event.

6. Initialization

All properties on the EventSlider are *DependencyProperty* types that participate in databinding operations. As such, any change to a single property will immediately propagate to a visual change in the control. In many cases, this also results in changes to other properties on the control, some of which might not be expected. For example, a change to the StartTime property will immediately cause the left-hand value of the visual slider on the control to change to the new date and time. This would cascade to a shift in the time window that is shown on the slider, which results in a change to the EndTime property, as well. The current zoom interval is preserved in this case, since only the StartTime was changed.

Alternatively, a change in the EndTime property does not affect the StartTime property. That would remain the same. Instead, the zoom interval would immediately change to fit in the new start and end times. This would cause a redraw of all events and change which events are seen and not seen.

This cascading effect is purposeful and allows runtime changes to the control external to the control's own functioning. The application could change the EventSlider's properties directly and expect the control to update accordingly. However, this effect is problematic for the control's initialization. To address this problem, the EventSlider supports the *ISupportInitialize* interface implemented on the generic *FrameworkElement* object. Two methods are used in this interface. The BeginInit method starts initialization and suspends all cascading effects on the EventSlider. Once this call is made, all the properties on the slider can be set without any subsequent effects. To end initialization and resume the normal behavior of the control, call the EndInit method. This also causes an immediate redraw of the control with the new values.

C#

```
List<MyObject> MyData = new List<MyData>();
EventSlider MyEventSlider = new EventSlider();
MyEventSlider.BeginInit();
MyEventSlider.TimePath = "MyTime";
MyEventSlider.ColorPath = "MyColor";
MyEventSlider.StartTime = DateTime.Now;
MyEventSlider.EndTime = start + new TimeSpan(30, 0, 0, 0);
MyEventSlider.Events = MyData;
MyEventSlider.EndInit();
```

In the previous example, a new EventSlider is created and initialized. The start time is set to the current time and the end time is set to 30 days later. This defines the zoom interval as 30 days. The MyData is a simple generic *List*<> defined to contain a custom business object type *MyObject*. The *List*<> generic type implements the *IEnumerable*<> interface. Generic collection interfaces in .NET are covariant, implicitly converting the more derived type *MyObject* to the less derived type *object*. Therefore, it is not necessary to do any explicit conversions when assigning collections to the Events property on the EventSlider.

The *MyObject* business object should define 2 properties. The MyTime property should contain a *DateTime* structure indicating the time of the event. The MyColor property should contain a *Brush* object indicating what color the event should be drawn with. The MyObject data should implement *INotifyPropertyChanged* if changes to the color or time on the event object during runtime are expected.

7. Currency

The EventSlider has the notion of a current event in the CurrentEvent property and time in the CurrentTime property. The current event is visually indicated on the slider by the position of the thumb. The user can move this thumb to a new position and it will snap to the nearest event when released. The SnapsToEvent property on the EventSlider is a *Boolean* that can be set to turn off that behavior. It is on by default to assist in keeping the CurrentEvent property more easily in sync with user intentions when moving the thumb.

The CurrentEvent property points to a single object in the collection contained in the Events property and this value can be changed at runtime through code as well as by manipulation of the control by the user interface. Whenever the current event is changed by either process, the CurrentTime property is set to the value of the time of the CurrentEvent.

When the CurrentEvent property is changed through code, the thumb on the EventSlider is moved to the correct line for that event. If the line is not currently being displayed, then the EventSlider moves the time window, preserving the current zoom interval, so that the event line is at the leftmost extent of the control. This results in the StartTime being the same as the CurrentTime.

The CurrentTime property can also be set through code. When this occurs, the CurrentEvent property remains unchanged. The EventSlider does not attempt to find an event near to this new time, because it could often be ambiguous. If the new current time is a value that is not in the currently displayed time window, then the EventSlider moves the time window such that the new current time is exactly in the center. The current zoom interval is preserved in this case, and the start and end times will be changed to new values so that the current time is in the middle. If the CurrentEvent is no longer visible in this new time window, then the thumb is merely moved to the leftmost extent of the control and is usually no longer on an event line. This may result in some confusion if there is an event line at that point, since that line does not correspond to the CurrentEvent. Because of this, caution should be used when changing the CurrentTime property in this manner.

8. Runtime Changes

As mentioned in the previous sections, changes to the start time, end time, current time, and current event can be made at runtime that immediately affect the EventSlider. Other properties similarly cause changes in the slider.

The ColorPath property can be changed at runtime, which will result in a redraw of the control using the new path on the event objects. This could be useful to show different views of the event data with different meanings for the colors.

The CurrentInterval property contains a *TimeSpan* indicating the current zoom interval. The Intervals property on the EventSlider contains a number of *TimeSpan* structures that are used by the slider when zooming. When the user zooms in or out, the slider searches for the next largest or smallest interval from this list to change the CurrentInterval property to this new value. When zooming and changing the CurrentInterval property in code, the CurrentEvent line and the thumb are moved to the center of the EventSlider display. Then the start and end times are adjusted according to the new interval while keeping the current event centered. If there is no current event, it will remain null, but the thumb and the EventSlider will still be centered via the CurrentTime property. The CurrentTime property will always have a valid time, usually set to the StartTime at initialization.

The Events property contains all the events that the EventSlider will display. Changing the value of this property will force a redraw of the entire control. The current time window will be preserved, but the CurrentEvent property will be set to null. The CurrentTime property will be set to the StartTime and the thumb will be set to the leftmost extent of the control. This can be an expensive operation if the number of events is very high. If a small change to the Events list is desired, then this can be accomplished through the UpdateEvents method. When this method is called with a new list of events as a parameter, then the new list is compared to the old list and appropriate additions and subtractions of events are made.

The MergedLineColor property indicates what color to draw lines where individual event lines are too close to be discernible. This property can be changed at runtime to have an immediate change in colors in the control.

The ShowMergedLines property is a *Boolean* flag that indicates whether a different color line should be drawn when event lines are drawn too close together to be seen at the current zoom interval. Changing this property at runtime causes a redraw of all events.

The ShowTicks property is a *Boolean* flag that indicates whether tick marks and labels should be drawn. This can be changed at runtime to hide or show these visual indicators.

The SnapsToEvent property is a *Boolean* flag that indicates whether the thumb should attempt to snap to the nearest event line when moved. Changing this value from false to true at runtime will cause the thumb to immediately attempt to find the nearest event and snap to it.

The TickColor property indicates what color the tick marks and labels should be drawn with. It is a *Brush* and changing it at runtime will cause these indicators to be redrawn with the new color.

The TickFontSize property indicates what size the labels on the tick marks should be. It can be changed at runtime and immediately be reflected on the control.

The TimePath property is a string indicating what property on the event objects to look for the time to display as a line on the slider. This property can be changed at runtime to a different property on the event objects causing a full redraw of the control. This could be useful for events that have secondary time information.

The ZoomInAtPointer property is a *Boolean* flag that indicates whether a zoom interval change should occur at the mouse pointer or at the center of the control. By default, zooms occur at the center of the control. Zooming at the mouse pointer is useful for a single zoom change at a certain point in time, but for more than one zoom change, the time window may shift in an unexpected manner. This value can be toggled at runtime for an immediate change to how zooming behaves.

The ZoomInCommand property contains a RoutedCommand that can be used to execute a zoom-in operation through code as if the user had used the mouse-wheel or touch to do a single zoom in.

The ZoomOutCommand property contains a RoutedCommand that can be used to execute a zoom-out operation through code as if the user had used the mouse-wheel or touch to do a single zoom out.

9. Application Programming Interface (API) Reference

9.1 Properties

9.1.1 ColorPath

Type: string

Default value: null

This property contains the name of the property that should be accessed on each event object in the Events property to indicate what color to use when drawing the line for that event. This allows the user to have individual events have different colors if so desired. It is expected that the property specified by this property on the Event object contains a *Brush* object. If the property path is invalid or points to a property that is not a *Brush*, the line for this event will be drawn with no color, resulting in it appearing invisible. If this property is not set, the default line color behavior will occur.

This property can be changed at runtime to point to a new property on the Event objects. Doing so will cause the EventSlider to update and draw the event lines using the *Brush* indicated in the new property reference. The most typical brush used would be a *SolidColorBrush*, but other brushes such as the *GradientBrush* may be used for interesting effects. There is a useful collection of common predefined solid-color brushes on the *System.Windows.Media.Brushes* static object.

C#

```
public class Data : INotifyPropertyChanged
{
    private object _Color = Brushes.Blue;
    public Brush Color
    {
        get { return this._Color; }
        set
        {
            if (this._Color != value)
            {
                this._Color = value;
                this.RaisePropertyChanged("Color");
            }
        }
    }
}

EventSlider MyEventSlider = new EventSlider();
MyEventSlider.ColorPath = "Color";
```

XAML

```
<es:EventSlider x:Name="MyEventSlider" ColorPath="Color"/>
```

As mentioned previously, the event objects should implement *INotifyPropertyChanged* or be a *DependencyProperty* in order for changes to an individual event's color property to be reflected in the EventSlider at runtime. This is shown in the previous example.

9.1.2 ColorPathProperty

Type: System.Windows.DependencyProperty

This is the static definition of the ColorPath property required by the WPF framework.

9.1.3 CurrentEvent

Type: object

Default value: null

This property contains the current event object and can be null when there is no current event selected. This property can be set by the user by clicking on the slider near an event line or moving the slider thumb. When set by code, the visual thumb is moved to the line indicating the event. If the event is not in the current time window, the time window is moved so the current event is equal to the start time and is at the leftmost edge of the slider window. This does not change the zoom interval, but can change the StartTime and EndTime properties.

Any time the CurrentEvent property is changed, the CurrentTime property is also changed to the time of the current event.

C#

```
List<Data> MyData = new List<Data>();
MyData.Add(new Data { Time = DateTime.Parse("4/1/16") });
MyData.Add(new Data { Time = DateTime.Parse("5/1/16") });
MyData.Add(new Data { Time = DateTime.Parse("6/1/16") });

EventSlider es = new EventSlider();
es.BeginInit();
es.StartTime = DateTime.Parse("1/1/16");
es.EndTime = DateTime.Parse("12/1/16");
es.Events = MyData;
es.CurrentEvent = MyData[1];
es.EndInit();
```

The previous example creates a list of event data objects for the months of April, May, and June. It then creates a new EventSlider and sets the start time to January and the EndTime to December. This will result in a time window of 12 months, which automatically sets the CurrentInterval to that value. The CurrentEvent property is set to the event at May. The result of the example is shown in Fig. 4. Note that because of the width of the window, the tick marks may not exactly be as expected. In this example, the length of the year is used and because each month has a different number of days, the labels did not always fall on the same day of each month.

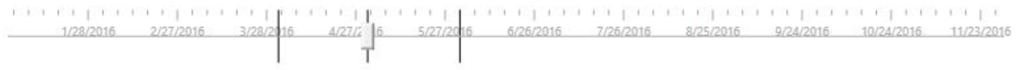

Fig. 4 Setting the CurrentEvent property

9.1.4 CurrentEventChangedEvent

Type: System.Windows.RoutedEvent

This is the static definition of the CurrentEventChanged event required by the WPF framework.

9.1.5 CurrentEventProperty

Type: System.Windows.DependencyProperty

This is the static definition of the CurrentEvent property required by the WPF framework.

9.1.6 CurrentInterval

Type: System.TimeSpan

Default value: TimeSpan.Zero

This property contains the current width of the visual window of the slider in units of time. This is also known as the zoom interval. This property is set with user interaction with the EventSlider control by zooming in or out with the mouse wheel or touch pinches. This property can also be set through code via the ZoomInCommand or ZoomOutCommand, or by changing the EndTime property.

When setting this property directly with code, the value of StartTime and EndTime are changed appropriately with the new interval centered on the current time of the current event.

When the EventSlider object is constructed, the CurrentInverval property has a default value of *TimeSpan.Zero*. Once the StartTime and EndTime properties are set, the CurrentInterval property is set as the difference between them. Also, when the SetDefaultIntervals method is invoked, the CurrentInterval property is set to the largest value in the Intervals property or 7 days.

9.1.7 CurrentIntervalChangedEvent

Type: System.Windows.RoutedEvent

This is the static definition of the CurrentIntervalChanged event required by the WPF framework.

9.1.8 CurrentIntervalProperty

Type: System.Windows.DependencyProperty

This is the static definition of the CurrentInterval property required by the WPF framework.

9.1.9 CurrentTime

Type: System.DateTime

Default value: default DateTime (1/1/0001 12:00 AM)

This property contains the current time, usually of the current event. If this property is set directly in code, the value of the SnapsToEvents property is ignored, and the current event's time will no longer match the current time. This will move the thumb to the new value of the CurrentTime property on the scale of the slider.

If the new time is not contained in the visual time window of the slider, the StartTime and EndTime values will be changed such that the new time is centered in the window with the CurrentInterval as its width.

9.1.10 CurrentTimeChangedEvent

Type: System.Windows.RoutedEvent

This is the static definition of the CurrentTimeChanged event required by the WPF framework.

9.1.11 CurrentTimeProperty

Type: System.Windows.DependencyProperty

This is the static definition of the CurrentTime property required by the WPF framework.

9.1.12 EndTime

Type: System.DateTime

Default value: default DateTime (1/1/0001 12:00 AM)

This property contains the rightmost value of the time window currently displayed by the control. This value can be changed through user interaction with the control or with code. When set through code, and the new EndTime value is larger than the current value of the StartTime property, the StartTime is preserved and the CurrentInterval is changed to the time between the new start and end time values. This presents an alternative means of zooming in or out.

If the new EndTime is before or equal to the StartTime, the time window is simply shifted with the CurrentInterval retained and the StartTime changed accordingly.

9.1.13 EndTimeChangedEvent

Type: System.Windows.RoutedEvent

This is the static definition of the EndTimeChanged event required by the WPF framework.

9.1.14 EndTimeProperty

Type: System.Windows.DependencyProperty

This is the static definition of the EndTime property required by the WPF framework.

9.1.15 EqualityComparer

Type: System.Collections.Generic.IEqualityComparer<object>

Default value: null

This property allows the setting of a custom equality comparer for use with the UpdateEvents method. This comparer is used to determine if new events passed as a parameter to that method are already in the list of events on the control. If this property is null, the default object comparison between the events will be used.

Using a custom equality comparer may be useful for certain design models were objects might be the same event because of their time, but not necessary be the same objects by memory reference.

9.1.16 EqualityComparerProperty

Type: System.Windows.DependencyProperty

This is the static definition of the EqualityComparer property required by the WPF framework.

9.1.17 Events

Type: System.Collections.Generic.IEnumerable<object>

Default value: null

This property contains the event objects displayed by the EventSlider. When this property is set, all previous values are cleared and replaced in favor of the new events and new lines are drawn on the slider control. This forces a complete redraw of the EventSlider control.

This value can be set directly with code, or participate in property databinding. However, because of its type, *IEnumerable<object>*, changes to the collection will not result in changes to the lines already drawn. It is necessary to change the whole value of the property to reflect an update. For finer control in how Events can be updated in the collection, see the UpdateEvents method (Section 11.6).

9.1.18 EventsProperty

Type: System.Windows.DependencyProperty

This is the static definition of the Events property required by the WPF framework.

9.1.19 Intervals

Type: System.Collections.ObjectModel.ObservableCollection<TimeSpan>

Default value: ObservableCollection<TimeSpan> with values listed below

This property contains all the intervals that are used for zooming in and out. By default, this collection is as follows:

1 s	5 min	6 h
5 s	10 min	12 h
10 s	30 min	1 day
30 s	1 h	3 days
1 min	3 h	7 days

This collection can be modified through code to contain custom intervals as appropriate for the times reflected in the collection of events. The control will sort the values each time a zoom is requested and pick the next larger or smaller entry as appropriate. Therefore, it is possible to increase or decrease the amount of zoom levels during runtime. Once the smallest zoom interval in this collection is reached, the control can no longer be zoomed in any further. Likewise, once the largest zoom interval in this collection is reached, the control can no longer be zoomed out any further.

To reset this collection to the default values again, invoke the SetDefaultIntervals method.

9.1.20 IntervalsProperty

Type: System.Windows.DependencyProperty

This is the static definition of the Intervals property required by the WPF framework.

9.1.21 LineColor

Type: System.Windows.Media.Brush

Default value: System.Windows.Media.Brushes.Black

This property contains the *Brush* object that should be used to draw the vertical event lines on the control. Any brush can be used, though the most common would be a *SolidColorBrush*. There is a useful collection of common predefined solid-color brushes on the *System.Windows.Media.Brushes* static object.

C#

```
EventSlider MyEventSlider = new EventSlider();
MyEventSlider.LineColor = Brushes.Orange;
```

XAML

```
<es:EventSlider x:Name="MyEventSlider" LineColor="Orange"/>
```

This property can be changed at runtime causing a redraw of the EventSlider event lines using the new color.

9.1.22 LineColorProperty

Type: DependencyProperty

This is the static definition of the LineColor property required by the WPF framework.

9.1.23 MergedLineColor

Type: System.Windows.Media.Brush

Default value: System.Windows.Media.Brushes.Red

This property contains the *Brush* object that should be used to draw the vertical event lines when the lines are so close together as to be overlapping and indiscernible at the current zoom interval. This is a visual cue that zooming in is required to see the separate events. This property can be changed at runtime causing a redraw of the event lines using the new color.

This behavior of changing colors can be turned off by changing the ShowMergedLines property.

9.1.24 MergedLineColorProperty

Type: System.Windows.DependencyProperty

This is the static definition of the MergedLineColor property required by the WPF framework.

9.1.25 ShowMergedLines

Type: Boolean

Default value: true

This property contains a flag indicating whether to change line colors when event lines on the control are too close to be discerned individually. When true, if more than one event line is drawn in the same space, the line changes to the color indicated in the MergedLineColor to visually indicate that more than one event is occurring at, or near to, this time. This property can be changed at runtime causing a redraw of the EventSlider event lines as appropriate.

9.1.26 ShowMergedLinesProperty

Type: System.Windows.DependencyProperty

This is the static definition of the ShowMergedLines property required by the WPF framework.

9.1.27 ShowTicks

Type: Boolean

Default value: true

This property contains a flag indicating whether to draw tick marks and time labels on the EventSlider. This value can be changed at runtime to hide or show the tick marks as specified.

9.1.28 ShowTicksProperty

Type: System.Windows.DependencyProperty

This is the static definition of the ShowTicks property required by the WPF framework.

9.1.29 SnapsToEvent

Type: Boolean

Default value: true

This property contains a flag indicating whether the thumb should snap to the nearest event when released or not. The user can move the thumb on the control to change the current time and event. When released, if this value is true, the thumb will attempt to snap to the closest event. When it does so, it sets the CurrentEvent to this event and the CurrentTime to the time of the event.

When this value is false, the control will set the CurrentTime to where the thumb was released. This value depends on the extent of the current time window being shown. The CurrentEvent will remain as it last was set.

9.1.30 SnapsToEventProperty

Type: System.Windows.DependencyProperty

This is the static definition of the SnapsToEvent property required by the WPF framework.

9.1.31 StartTime

Type: System.DateTime

Default value: default DateTime (1/1/0001 12:00 AM)

This property contains the leftmost value of the time window currently displayed by the slider control. This value can be changed through user interaction with the control by a variety of means. When panning left or right in the user interface, the value of this property will be set. The time window can be zoomed in or out, which can also change the StartTime property by virtue of centering the control on the new value of the CurrentTime property.

When the StartTime property is set through code, the EndTime property is also changed the amount of time equal to the value of the zoom interval indicated in the CurrentInterval property. This presents a straightforward way of panning the time window through code simply by changing this property.

9.1.32 StartTimeChangedEvent

Type: System.Windows.RoutedEvent

This is the static definition of the StartTimeChanged event required by the WPF framework.

9.1.33 StartTimeProperty

Type: System.Windows.DependencyProperty

This is the static definition of the StartTime property required by the WPF framework.

9.1.34 TickColor

Type: System.Windows.Media.Brush

Default value: System.Windows.Media.Brushes.DarkGray

This property contains the *Brush* object that should be used to draw the ticks when the ShowTicks property is set true. This property can be changed at runtime resulting in the tick marks and labels to be redrawn using the new brush.

9.1.35 TickColorProperty

Type: System.Windows.DependencyProperty

This is the static definition of the TickColor property required by the WPF framework.

9.1.36 TickFontSize

Type: Double

Default value: 9.0

This property contains the value that should be used for the font size of the labels displayed on the tick marks. This property can be changed at runtime resulting in a redraw of the tick marks and labels using the new value.

9.1.37 TickFontSizeProperty

Type: System.Windows.DependencyProperty

This is the static definition of the TickFontSize property required by the WPF framework.

9.1.38 TimePath

Type: String

Default value: null

This property contains the name of the property that should be accessed on each event object in the Events property for the time to use for that event's line. If this property is not set, the control will assume the objects are *DateTime* structures. If the property path is invalid or points to a property that is not a *DateTime*, no lines will be drawn on the slider control.

This property can be changed at runtime to point to a new property on the Event objects. Doing so will cause the EventSlider to update and draw the event lines using the new time indicated in the new property reference.

C#

```
public class Data : INotifyPropertyChanged
{
    private object _Time;
    public DateTime Time
    {
        get { return this._Time; }
        set
        {
            if (this._Time != value)
            {
                this._Time = value;
                this.RaisePropertyChanged("Time");
            }
        }
    }
}

EventSlider MyEventSlider = new EventSlider();
MyEventSlider.TimePath = "Time";
```

XAML

```
<es:EventSlider x:Name="MyEventSlider" TimePath="Time"/>
```

As mentioned previously, the event objects should implement *INotifyPropertyChanged* or be a *DependencyProperty* in order for changes to an individual event's time property to be reflected in the EventSlider at runtime. This is shown in the previous example. This would allow, for instance, an event to move itself to a new time on the slider, which could pose interesting scenarios.

9.1.39 TimePathProperty

Type: System.Windows.DependencyProperty

This is the static definition of the TimePath property required by the WPF framework.

9.1.40 ZoomInAtPointer

Type: Boolean

Default value: false

If this property is set true, when the user zooms in on the control via the user interface, such as a mouse wheel, the zoom will center on where the mouse pointer is located.

If false, the default value, the slider control moves the current event to the center of the visual window and then zooms, changing the size of the time window zoom interval around it.

Setting this value true can have some unexpected behavior when multiple zooms are initiated, which is often the case when using a mouse wheel. Caution should be exercised.

9.1.41 ZoomInAtPointerProperty

Type: System.Windows.DependencyProperty

This is the static definition of the ZoomInAtPointer property required by the WPF framework.

9.1.42 ZoomInCommand

Type: System.Windows.RoutedCommand

This is the static definition of the ZoomInCommand required by the WPF framework.

The ZoomInCommand is executed each time a zoom in operation is requested by the control. As a *RoutedCommand*, different input bindings may be assigned allowing zooming in with different keyboard, touch, and mouse gestures.

Similarly, the command can be executed in code causing the control to zoom in one zoom interval.

C#

```
EventSlider.ZoomInCommand.Execute(null, this.MyEventSlider);
```

9.1.43 ZoomOutCommand

Type: System.Windows.RoutedCommand

This is the static definition of the ZoomOutCommand required by the WPF framework.

The ZoomOutCommand is executed each time a zoom out operation is requested by the control. As a *RoutedCommand*, different input bindings may be assigned allowing zooming out with different keyboard, touch, and mouse gestures.

Similarly, the command can be executed in code causing the control to zoom out one zoom interval.

C#

```
EventSlider.ZoomOutCommand.Execute(null, this.MyEventSlider);
```

10. Events

10.1 CurrentEventChanged

Event argument type:

System.Windows.RoutedPropertyChangedEventArgs<object>

This event fires whenever the CurrentEvent property is changed. The oldvalue and newvalues are passed to any event handlers subscribed.

Subscription to this event allows the developer to know when the currency of the EventSlider control has been changed from one business model event object to another. This event is raised when the property changes by any means, whether it be by user manipulation on the user interface or through code.

During the *ISupportInitialize* process, when BeginInit is called, this event is suppressed. When the EndInit method is invoked, if there is a change to the CurrentEvent property as part of the initialization, this event is raised.

10.2 CurrentIntervalChanged

Event argument type:

System.Windows.RoutedPropertyChangedEventArgs<TimeSpan>

This event fires whenever the CurrentInterval property is changed. The oldvalue and newvalues are passed to any event handlers subscribed.

Subscription to this event allows the developer to know when the EventSlider control has been zoomed in or out. This event is raised when the property changes by any means, whether it be by user manipulation on the user interface or through code such as executing the ZoomInCommand or ZoomOutCommand, or changing the CurrentInterval property directly.

During the *ISupportInitialize* process, when BeginInit is called, this event is suppressed. When the EndInit method is invoked, if there is a change to the CurrentInterval property as part of the initialization, this event is raised.

10.3 CurrentTimeChanged

Event argument type:

System.Windows.RoutedPropertyChangedEventArgs<DateTime>

This event fires whenever the CurrentTime property is changed. The oldvalue and newvalues are passed to any event handlers subscribed.

Subscription to this event allows the developer to know when the thumb of the EventSlider control has been changed to a different time. This event is raised when the property changes by any means, whether it be by user manipulation on the user interface or through code.

During the *ISupportInitialize* process, when BeginInit is called, this event is suppressed. When the EndInit method is invoked, if there is a change to the CurrentTime property as part of the initialization, this event is raised.

10.4 EndTimeChanged

Event argument type:

System.Windows.RoutedPropertyChangedEventArgs<DateTime>

This event fires whenever the EndTime property is changed. The oldvalue and newvalues are passed to any event handlers subscribed.

Subscription to this event allows the developer to know when the time window of the EventSlider control has been changed. This event is raised when the property changes by any means, whether it be by user manipulation on the user interface or through code.

During the *ISupportInitialize* process, when BeginInit is called, this event is suppressed. When the EndInit method is invoked, if there is a change to the EndTime property as part of the initialization, this event is raised.

10.5 StartTimeChanged

Event argument type:

System.Windows.RoutedPropertyChangedEventArgs<DateTime>

This event fires whenever the StartTime property is changed. The oldvalue and newvalues are passed to any event handlers subscribed.

Subscription to this event allows the developer to know when the time window of the EventSlider control has been changed. This event is raised when the property changes by any means, whether it be by user manipulation on the user interface or through code.

During the *ISupportInitialize* process, when BeginInit is called, this event is suppressed. When the EndInit method is invoked, if there is a change to the StartTime property as part of the initialization, this event is raised.

11. Methods

11.1 BeginInit

Parameters: none

This method is defined in the System.ComponentModel.ISupportInitialize interface.

This method begins a batch initialization of properties on the slider. No events are fired or property changes made to the slider control until the batch initialization is completed with the EndInit method. This allows code to set a number of properties together without them changing each other as during normal operation of the control.

This is the preferred method for initializing the EventSlider through code.

11.2 EndInit

Parameters: none

This method is defined in the System.ComponentModel.ISupportInitialize interface.

This method completes a batch initialization of properties on the slider. All changes made to the properties of the control are applied, and any appropriate events are fired to note changes.

This is the preferred method for initializing the EventSlider through code.

11.3 EventSlider

Parameters: none

This method is the public constructor of the EventSlider control and takes no parameters.

11.4 OnApplyTemplate

P*arameters*: none

This method is a required override for custom controls. This method hooks up the template with objects and events internal to the operation of the slider.

11.5 SetDefaultIntervals

Parameters: none

This method resets the intervals contained in the Intervals property back to the defaults described in this document under the Intervals property. When this method is called, the CurrentInterval property is set to the largest default interval, which is 7 days.

11.6 UpdateEvents

Parameters: System.Collections.Generic.IEnumerable<object>

This method allows for updating the list of events on the slider instead of replacing them completely by setting the Events property. The *IEqualityComparer* set in the EqualityComparer property is used to determine if the events in the new list of events passed as a parameter has overlap with the existing list in the Events property. New events from the list are added, and events not in the new list are removed from the Events property so it matches the parameter. This should result in faster updates to the slider for small changes in the events collection.

12. Conclusion

The EventSlider is a custom WPF control that displays vertical lines representing events on a sliding time scale. With this control integrated into an application, one can visualize any kind of time-related data in a sensible graphical manner. A user is able to scroll through and select events in time in such a manner that the rest of the application can do a more detailed presentation about these events.

The detailed instructions contained in this manual allow a developer to smoothly integrate the control into any WPF application. The API reference fully described all of the properties, events, and methods available on the control that may be used to manipulate the events shown and select a single event or time that a user is interested in.

1	DEFENSE TECH INFO CTR
(PDF)	DTIC OCA
2	DIRECTOR
(PDF)	US ARMY RSRCH LAB
	RDRL CIO L
	IMAL HRA MAIL & RECORDS MGMT
1	GOVT PRNTG OFC
(PDF)	A MALHOTRA
4	US ARMY RSRCH LAB
(PDF)	RDRL CII B
	C SCHLESIGER
	A RAGLIN
	S RUSSELL
	RP WINKLER

INTENTIONALLY LEFT BLANK