

## Adding week to user control.

**Source:**

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-05/0014.html>

---

**From:** Chris Millar (*chris.millar\_at\_expresshr.com*)

**Date:** 04/30/04

Date: Fri, 30 Apr 2004 11:24:26 +0100

I have a user control that i wish to extend to change the date when the user selects the numeric up down button.

The code explains itself, hope someone can help.

any ideas appreciated..

Chris.

code :

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Globalization;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Text;
using Microsoft.Win32;
namespace ClearlyDotNet
{
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
public class TimeFrame : System.Windows.Forms.UserControl
{
    #region Component Designer generated code

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>

    protected override void Dispose( bool disposing )
    {
        if( disposing )
        {
            if( components != null )
            components.Dispose();
        }
        base.Dispose( disposing );
    }

    public System.Windows.Forms.NumericUpDown numericUpDown1;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Panel panel1;

    /// <summary>
    /// Required designer variable.
    /// </summary>

    private System.ComponentModel.Container components = null;

    /// <summary>
    /// Required method for Designer support – do not modify
    /// the contents of this method with the code editor.
    /// </summary>

    Adding week to user control.
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
private void InitializeComponent()
{
    this.numericUpDown1 = new System.Windows.Forms.NumericUpDown();
    this.label1 = new System.Windows.Forms.Label();
    this.panel1 = new System.Windows.Forms.Panel();

    ((System.ComponentModel.ISupportInitialize)(this.numericUpDown1)).BeginInit(
    );
    this.panel1.SuspendLayout();
    this.SuspendLayout();

    //
    // numericUpDown1
    //
    this.numericUpDown1.Location = new System.Drawing.Point(72, 2);
    this.numericUpDown1.Name = "numericUpDown1";
    this.numericUpDown1.Size = new System.Drawing.Size(40, 20);
    this.numericUpDown1.TabIndex = 0;
    this.numericUpDown1.Value = new System.Decimal(new int[] {
    2,
    0,
    0,
    0});
    this.numericUpDown1.ValueChanged += new
    System.EventHandler(this.numericUpDown1_ValueChanged);

    //
    // label1
    //
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
this.label1.Font = new System.Drawing.Font("Arial", 8.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((System.Byte)(0)));

this.label1.Location = new System.Drawing.Point(4, 6);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(56, 16);

this.label1.TabIndex = 1;

this.label1.Text = "Week No";

//

// panel1

//

this.panel1.Anchor = System.Windows.Forms.AnchorStyles.None;

this.panel1.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;

this.panel1.Controls.Add(this.numericUpDown1);

this.panel1.Controls.Add(this.label1);

this.panel1.Location = new System.Drawing.Point(0, 171);

this.panel1.Name = "panel1";

this.panel1.Size = new System.Drawing.Size(720, 24);

this.panel1.TabIndex = 2;

//

// TimeFrame

//

this.BackColor = System.Drawing.Color.White;

this.Controls.Add(this.panel1);

this.Name = "TimeFrame";

this.Size = new System.Drawing.Size(728, 224);

((System.ComponentModel.ISupportInitialize)(this.numericUpDown1)).EndInit();

Adding week to user control.
```

```
this.panel1.ResumeLayout(false);
```

```
this.ResumeLayout(false);
```

```
}
```

```
#endregion
```

```
#region Nested Classes
```

```
#region TimeToolTip
```

```
internal class TimeToolTip
```

```
{
```

```
public string Text;
```

```
}
```

```
#endregion
```

```
#region TimeFrameObject definition
```

```
internal class TimeFrameObject
```

```
{
```

```
public RectangleF Bounds;
```

```
private bool selected = false;
```

```
public bool Highlighted = false;
```

```
public virtual bool Selected
```

```
{
```

```
get
```

```
{
```

```
return selected;
```

```
}
```

```
set
```

```
{
```

```
selected = value;
```

```
Adding week to user control.
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
}  
}  
public TimeFrameObject( RectangleF rec )  
{  
    Bounds = rec;  
}  
public bool Contains( PointF p )  
{  
    return this.Contains( p.X, p.Y );  
}  
public bool Contains( float x, float y )  
{  
    return Bounds.Contains( x, y );  
}  
}  
#endregion  
#region _Date definition  
internal class _Date : TimeFrameObject  
{  
    public int Row;  
    public String dt1;  
    public String DT  
    {  
        get  
        {  
            System.TimeSpan duration = new System.TimeSpan(Row, 0, 0, 0);
```

Adding week to user control.



```
set
{
Row = (int)value;
}
}

public _Day( int row ) : this( new RectangleF(0,0,0,0), row )
{
}

public _Day( RectangleF rec, int row ) : base( rec )
{
Row = row;
}

public override bool Selected
{
get
{
int ColCount = 0;
for( int j = 0; j < ClearlyDotNet.TimeFrame.s_NumTimeIntervals; ++j )
{
if( CellMatrix[ Row, j ].Selected )
{
++ColCount;
}
}

return( ColCount == ClearlyDotNet.TimeFrame.s_NumTimeIntervals );
}
}
```

```
set
{
for( int j = 0; j < s_NumTimeIntervals; ++j )
{
CellMatrix[ Row, j ].Selected = value;
}
}
}
}
}

#endregion

#region TimeSlot definition

internal class TimeSlot : TimeFrameObject
{
public int Column;

public TimeSlot( int col ) : this( col, new RectangleF(0,0,0,0) )
{
}

public TimeSlot( int col, RectangleF rec ) : base( rec )
{
Column = col;
}

public override bool Selected
{
get
{
int RowCount = 0;
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
for( int i = 0; i < ClearlyDotNet.TimeFrame.s_NumDays; ++i )
{
if( CellMatrix[ i, Column ].Selected )
{
++RowCount;
}
}
return( RowCount == ClearlyDotNet.TimeFrame.s_NumDays );
}
set
{
for( int i = 0; i < ClearlyDotNet.TimeFrame.s_NumDays; ++i )
{
CellMatrix[ i, Column ].Selected = value;
}
}
}
}
#endregion
#region Cell Definition
internal class Cell : TimeFrameObject
{
public int X;
public int Y;
public override string ToString()
{
Adding week to user control.
```

```
return( "{" + X + ", " + Y + " }" );  
  
}  
  
public Cell( int x, int y ) : this( new RectangleF(0,0,0,0), x, y )  
  
{  
  
}  
  
public Cell( RectangleF rec, int x, int y ) : base( rec )  
  
{  
  
X = x;  
  
Y = y;  
  
}  
  
}  
  
#endregion  
  
#region TopLeftCell Definition  
  
internal class TopLeftCell : TimeFrameObject  
  
{  
  
public TopLeftCell() : base( new RectangleF(0,0,0,0) )  
  
{  
  
}  
  
public TopLeftCell( RectangleF rec ) : base( rec )  
  
{  
  
}  
  
public override bool Selected  
  
{  
  
get  
  
{  
  
int Count = 0;
```

```
for( int i = 0; i < s_NumDays; ++i )
{
for( int j = 0; j < s_NumTimeIntervals; ++j )
{
if( CellMatrix[ i, j ].Selected )
{
++Count;
}
}
}

return( Count == ( s_NumDays * s_NumTimeIntervals ) );
}

set
{
for( int i = 0; i < s_NumDays; ++i )
{
for( int j = 0; j < s_NumTimeIntervals; ++j )
{
CellMatrix[ i, j ].Selected = value;
}
}
}
}

#endregion

#endregion
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
#region Constants

private const int c_ToolTipXPad = 12;

private const int c_LeftMargin = 20;

private const int c_RightMargin = 5;

private const int c_Header = 10;

private const int c_Footer = 10;

private const int c_NumDays = 7;

// new const

private const int c_NumDates = 7;

private const int c_NumTimeIntervals = 48;

private const int c_TimeDisplayInterval = 4;

private const int c_TimeGridVerticalInterval = 2;

#endregion

#region Statics

private static TimeSpan c_ts = new TimeSpan( 1,0,0,0,0 );

public static int s_NumDays = c_NumDays;

//new static

public static int s_NumDates = c_NumDates;

public static int s_NumTimeIntervals = c_NumTimeIntervals;

#endregion

#region Member Variables

private int m_LeftMargin = c_LeftMargin;

private int m_RightMargin = c_RightMargin;

private int m_Header = c_Header;

private int m_Footer = c_Footer;

private int m_NumDays = s_NumDays;

Adding week to user control.
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
//new dude

private int m_NumDates = s_NumDates;

private int m_NumTimeIntervals = s_NumTimeIntervals;

private int m_TimeDisplayInterval = c_TimeDisplayInterval;

private int m_TimeGridVerticalInterval = c_TimeGridVerticalInterval;

internal static Cell[,] CellMatrix;

private Timer HoverTimerStart = new Timer();

private Timer HoverTimerStop = new Timer();

private MouseEventArgs LastMouseHover = null;

private _Day[] Days;

//new member variable

private _Date[] Dates;

private TimeSlot[] TimeSlots;

private TopLeftCell TopLeft;

private RectangleF Grid;

private float x_grid_inc = 0.0F;

private float y_grid_inc = 0.0F;

private TimeToolTip ttInfo = null;

private TimeFrameObject tfoLastHighlighted = null;

private TimeFrameObject tfoLastSelected = null;

private SolidBrush m_sbBack = new SolidBrush( SystemColors.Control );

private SolidBrush m_sbGridBack = new SolidBrush( Color.White );

private SolidBrush m_sbInfoBack = new SolidBrush( SystemColors.Info );

private SolidBrush m_sbInfoText = new SolidBrush( SystemColors.InfoText );

private SolidBrush m_sbHotTrack = new SolidBrush( Color.Red );

private SolidBrush m_sbSelected = new SolidBrush( SystemColors.Highlight );

Adding week to user control.
```

```
private SolidBrush m_sbForeColor = new SolidBrush(  
SystemColors.ControlText );
```

```
#endregion
```

```
#region Properties
```

```
protected override Size DefaultSize
```

```
{
```

```
get
```

```
{
```

```
return new Size( 310, 160 );
```

```
}
```

```
}
```

```
public TimeSpan[][][] TimeFrames
```

```
{
```

```
get
```

```
{
```

```
ArrayList TimeSpanDays = new ArrayList();
```

```
long t_inc = c_ts.Ticks / (long)NumTimeIntervals;
```

```
TimeSpan tsi = new TimeSpan( t_inc );
```

```
for( int i = 0; i < NumDays; ++i )
```

```
{
```

```
ArrayList TimeSpansPerDay = new ArrayList();
```

```
TimeSpan[] ts = null;
```

```
bool bAdjacent = false;
```

```
for( int j = 0; j < NumTimeIntervals; ++j )
```

```
{
```

```
if( CellMatrix[ i, j ].Selected )
```

```
{
if( ! bAdjacent )
{
ts = new TimeSpan[ 2 ];
ts[ 0 ] = new TimeSpan( j * t_inc );
bAdjacent = true;
}
}
else
{
if( bAdjacent )
{
ts[ 1 ] = new TimeSpan( j * t_inc );
TimeSpansPerDay.Add( ts );
}
bAdjacent = false;
}
}
if( bAdjacent )
{
ts[ 1 ] = new TimeSpan( NumTimeIntervals * t_inc );
TimeSpansPerDay.Add( ts );
}
TimeSpanDays.Add( TimeSpansPerDay );
}
TimeSpan[][][] all = new TimeSpan[ TimeSpanDays.Count ][][];
```

```
for( int i = 0; i < TimeSpanDays.Count; ++i )
{
    ArrayList al = (ArrayList)TimeSpanDays[ i ];
    all[ i ] = new TimeSpan[ al.Count ][];
    al.CopyTo( all[ i ] );
}
return all;
}
}

#region Display Intervals
[Description("The number of days to display, starting with 0==Sunday")]
[DefaultValue(c_NumDays)]
public int NumDays
{
    get { return m_NumDays; }
    set
    {
        m_NumDays = value;
        s_NumDays = m_NumDays;
        SetLimits();
    }
}
[Description("The number of dates to display, starting with first")]
[DefaultValue(c_NumDates)]
public int NumDates
{
    Adding week to user control.
```

```
get { return m_NumDates; }

set

{

m_NumDates = value;

s_NumDates = m_NumDates;

SetLimits();

}

}

[Description("The logical number of time intervals in the 24 hour time
period")]

[DefaultValue(c_NumTimeIntervals)]

public int NumTimeIntervals

{

get { return m_NumTimeIntervals; }

set

{

m_NumTimeIntervals = value;

s_NumTimeIntervals = m_NumTimeIntervals;

SetLimits();

}

}

[Description("The number of text time markers at the bottom of the grid")]

[DefaultValue(c_TimeDisplayInterval)]

public int TimeDisplayInterval

{

get { return m_TimeDisplayInterval; }
```

```
set { m_TimeDisplayInterval = value; }  
  
}  
  
[Description("The number of vertical lines to display in the time period")]  
  
[DefaultValue(c_TimeGridVerticalInterval)]  
  
public int TimeGridVerticalInterval  
  
{  
  
get { return m_TimeGridVerticalInterval; }  
  
set { m_TimeGridVerticalInterval = value; }  
  
}  
  
#endregion  
  
#region Colors  
  
public Color Selected  
  
{  
  
get  
  
{  
  
return m_sbSelected.Color;  
  
}  
  
set  
  
{  
  
m_sbSelected = new SolidBrush( value );  
  
}  
  
}  
  
public Color HotTrack  
  
{  
  
get  
  
{  
  
Adding week to user control.
```

```
return m_sbHotTrack.Color;

}

set

{

m_sbHotTrack = new SolidColorBrush( value );

}

}

public Color GridBack

{

get

{

return m_sbGridBack.Color;

}

set

{

m_sbGridBack = new SolidColorBrush( value );

}

}

public override Color BackColor

{

get

{

return m_sbBack.Color;

}

set

{

Adding week to user control.
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
m_sbBack = new SolidBrush( value );  
  
}  
  
}  
  
public override Color ForeColor  
  
{  
  
get  
  
{  
  
return m_sbForeColor.Color;  
  
}  
  
set  
  
{  
  
m_sbForeColor = new SolidBrush( value );  
  
}  
  
}  
  
#endregion  
  
#region Percentages  
  
private float Header  
  
{  
  
get  
  
{  
  
return this.Height * ( m_Header / 100.0F );  
  
}  
  
}  
  
[  
  
Category( "Percentages" ),
```

Description("Specifies the vertical percentage of space used for the Time Slots above the Grid."),

DefaultValue(c\_Header)

]

public int HeaderPercentage

{

get

{

return m\_Header;

}

set

{

m\_Header = value;

}

}

private float Footer

{

get

{

return this.Height \* ( m\_Footer / 80.0F );

}

}

[

Category( "Percentages" ),

Description("Specifies the vertical percentage of space used for the Grid."),

DefaultValue(c\_Footer)

Adding week to user control.

```
]
public int FooterPercentage
{
    get
    {
        return m_Footer;
    }
    set
    {
        m_Footer = value;
    }
}

private float LeftMargin
{
    get
    {
        return this.Width * ( m_LeftMargin / 100.0F );
    }
}

[
    Category( "Percentages" ),
    Description("Specifies the horizontal percentage of space used for the
displays of days."),
    DefaultValue(c_LeftMargin)
]

public int LeftMarginPercentage
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
{
get
{
return m_LeftMargin;
}
set
{
m_LeftMargin = value;
}
}

private float RightMargin
{
get
{
return this.Width * ( m_RightMargin / 100.0F );
}
}

[
Category( "Percentages" ),

Description("Specifies the horizontal percentage of space used for the space
next to the right of the Grid."),

DefaultValue(c_RightMargin)
]

public int RightMarginPercentage
{
get
```

Adding week to user control.

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
{
return m_RightMargin;
}
set
{
m_RightMargin = value;
}
}
#endregion
#endregion
public TimeFrame()
{
HoverTimerStart.Stop();
HoverTimerStart.Interval = 500;
HoverTimerStop.Stop();
HoverTimerStop.Interval = 4000;
HoverTimerStart.Tick += new EventHandler( OnHoverTimerStart );
HoverTimerStop.Tick += new EventHandler( OnHoverTimerStop );
SetLimits();
TopLeft = new TopLeftCell();
// This call is required by the Windows.Forms Form Designer.
InitializeComponent();
}
#region Member Functions
#region OnEvents
protected override void OnResize( EventArgs e )
```

Adding week to user control.

```
{  
if( this.Width < DefaultSize.Width ) this.Width = DefaultSize.Width;  
if( this.Height < DefaultSize.Height ) this.Height = DefaultSize.Height;  
Grid = new RectangleF( LeftMargin, Header,  
this.Width - ( LeftMargin + RightMargin ),  
this.Height - ( Footer + Header ) );  
TopLeft.Bounds = new RectangleF( 0, 0, Grid.X, Grid.Y );  
x_grid_inc = Grid.Width / NumTimeIntervals;  
y_grid_inc = Grid.Height / NumDays;  
//Reset sizes  
for( int i = 0; i < NumDays; ++i )  
{  
for( int j = 0; j < NumTimeIntervals; ++j )  
{  
CellMatrix[ i, j ].Bounds = new RectangleF( Grid.X + ( j * x_grid_inc ),  
Grid.Y + ( i * y_grid_inc ), x_grid_inc, y_grid_inc );  
}  
}  
for( float i = 0; i < NumTimeIntervals; ++i )  
{  
TimeSlots[ (int)i ].Bounds = new RectangleF( Grid.X + ( i * x_grid_inc ),  
0.0F, x_grid_inc , Header );  
}  
for( float i = 0; i < NumDays; ++i )  
{  
Days[ (int)i ].Bounds = new RectangleF( 0.0F, Grid.Y + ( i * y_grid_inc ),  
LeftMargin, y_grid_inc );  
}
```

```
}  
  
//resize dates  
  
for( float i = 0; i < NumDates; ++i )  
  
{  
  
    Dates[ (int)i ].Bounds = new RectangleF( 0.0F, Grid.Y + (i * y_grid_inc),  
    LeftMargin + 200, y_grid_inc );  
  
}  
  
this.Invalidate( true );  
  
}  
  
protected override void OnPaint( PaintEventArgs e )  
  
{  
  
    Draw( e.Graphics );  
  
}  
  
protected override void OnMouseDown( MouseEventArgs e )  
  
{  
  
    TimeFrameObject tfo = GetTimeFrameObject( e );  
  
    if( tfo != null )  
  
    {  
  
        if( e.Button == MouseButton.Left )  
  
        {  
  
            tfo.Selected = ! tfo.Selected;  
  
            tfoLastSelected = tfo;  
  
            this.Invalidate( true );  
  
        }  
  
    }  
  
}  
  
}
```

```
protected override void OnMouseMove( MouseEventArgs e )
{
    ClearToolTopState();

    TimeFrameObject tfo = GetTimeFrameObject( e );

    if( e.Button == MouseButton.Left )
    {
        if( tfo is Cell )
        {
            Cell cellNewSelected = (Cell)tfo;

            if( tfoLastSelected is Cell )
            {
                Cell cellLastSelected = (Cell)tfoLastSelected;

                if( cellLastSelected != cellNewSelected )
                {
                    SelectCellBlock( cellLastSelected, cellNewSelected );
                }
            }
        }
        else if( tfo is TimeSlot )
        {
            TimeSlot tsNewSelected = (TimeSlot)tfo;

            if( tfoLastSelected is TimeSlot )
            {
                TimeSlot tsLastSelected = (TimeSlot)tfoLastSelected;

                if( tsLastSelected != tsNewSelected )
                {
```

```
if( tsNewSelected.Column > tsLastSelected.Column )
{
for( int j = tsLastSelected.Column; j < tsNewSelected.Column; ++j )
{
TimeSlots[ j ].Selected = tsLastSelected.Selected;
}
}
else if( tsNewSelected.Column < tsLastSelected.Column )
{
for( int j = tsNewSelected.Column; j < tsLastSelected.Column; ++j )
{
TimeSlots[ j ].Selected = tsLastSelected.Selected;
}
}
}
}
}
else if( tfo is _Day )
{
_Day dNewSelected = (_Day)tfo;
if( tfoLastSelected is _Day )
{
_Day dLastSelected = (_Day)tfoLastSelected;
if( dLastSelected != dNewSelected )
{
if( dNewSelected.Row > dLastSelected.Row )
```

```
{
for( int i = dLastSelected.Row; i < dNewSelected.Row; ++i )
{
Days[ i ].Selected = dLastSelected.Selected;
}
}
else if( dNewSelected.Row < dLastSelected.Row )
{
for( int i = dNewSelected.Row; i < dLastSelected.Row; ++i )
{
Days[ i ].Selected = dLastSelected.Selected;
}
}
}
}
}
}
}
}
if( tfoLastHighlighted != null )
{
tfoLastHighlighted.Highlighted = false;
}
if( tfo != null )
{
tfo.Highlighted = true;
tfoLastHighlighted = tfo;
}
}
```

```
if( e.Button == MouseButton.None )
{
if( Grid.Contains( e.X, e.Y ) )
{
LastMouseHover = e;
HoverTimerStart.Start();
}
}
this.Invalidate( true );
}
protected override void OnLoad( EventArgs e )
{
SetStyle( ControlStyles.DoubleBuffer, true );
SetStyle( ControlStyles.AllPaintingInWmPaint, true );
SetStyle( ControlStyles.UserPaint, true );
}
#endregion
#region ToolTip Methods
private void ClearToolTopState()
{
LastMouseHover = null;
ttInfo = null;
HoverTimerStop.Stop();
HoverTimerStart.Stop();
}
private void OnHoverTimerStop( object sender, EventArgs e )
```

```
{
if( Grid.Contains( LastMouseHover.X, LastMouseHover.Y ) )
{
ClearToolTopState();
this.Invalidate( true );
}
}

private void OnHoverTimerStart( object sender, EventArgs e )
{
if( LastMouseHover != null )
{
if( Grid.Contains( LastMouseHover.X, LastMouseHover.Y ) )
{
HoverTimerStart.Stop();
HoverTimerStop.Start();
ttInfo = new TimeToolTip();
ttInfo.Text = GetToolTipText( LastMouseHover );
this.Invalidate( true );
}
}
}

private string GetToolTipText( MouseEventArgs e )
{
string TipText = string.Empty;
if( Grid.Contains( e.X, e.Y ) )
{
Adding week to user control.
```

```
Cell cell = (Cell)GetTimeFrameObject( e );

if( cell.Selected )

{

int end = cell.Y;

int start = cell.Y;

for( end = cell.Y; end < NumTimeIntervals; ++end )

{

if( ! CellMatrix[ cell.X, end ].Selected )

{

break;

}

}

for( start = cell.Y; start >= 0; --start )

{

if( ! CellMatrix[ cell.X, start ].Selected )

{

break;

}

}

++start;

long t_inc = c_ts.Ticks / (long)NumTimeIntervals;

DateTime dtStart = new DateTime( start * t_inc );

DateTime dtEnd = new DateTime( end * t_inc );

string strStart = dtStart.ToString( "hh:mm tt" );

string strEnd = dtEnd.ToString( "hh:mm tt" );

TipText = strStart + " to " + strEnd;

Adding week to user control.
```

```
}  
  
}  
  
return TipText;  
  
}  
  
#endregion  
  
private void SetLimits()  
{  
  
Days = new _Day[ m_NumDays ];  
  
//new for dates  
  
Dates = new _Date[ m_NumDates ];  
  
TimeSlots = new TimeSlot[ m_NumTimeIntervals ];  
  
CellMatrix = new Cell[ NumDays, NumTimeIntervals ];  
  
//Create Cells  
  
for( int i = 0; i < NumDays; ++i )  
{  
  
for( int j = 0; j < NumTimeIntervals; ++j )  
{  
  
CellMatrix[ i,j ] = new Cell( i, j );  
  
}  
  
}  
  
// Create TimeSlots  
  
for( int j = 0; j < NumTimeIntervals; ++j )  
{  
  
TimeSlots[ j ] = new TimeSlot( j );  
  
}  
  
// Create Days
```

```
for( int i = 0; i < NumDays; ++i )
{
    Days[ i ] = new _Day( i );
}

//Create Dates

for( int i = 0; i < NumDates; ++i )
{
    Dates[ i ] = new _Date( i );
}

}

private TimeFrameObject GetTimeFrameObject( MouseEventArgs e )
{
    TimeFrameObject tfo = null;

    if( Grid.Contains( e.X, e.Y ) )
    {
        tfo = CellMatrix[ (int)( ( e.Y - Header ) / y_grid_inc), (int)( ( e.X -
        LeftMargin ) / x_grid_inc) ];
    }

    else if( ( e.X > Grid.X ) && ( e.X < ( Grid.X + Grid.Width ) ) && ( e.Y <
    Grid.Y ) )
    {
        tfo = (TimeFrameObject)TimeSlots[ (int)( ( e.X - LeftMargin ) /
        x_grid_inc) ];
    }

    else if( ( e.X < Grid.X ) && ( e.Y < ( Grid.Y + Grid.Height ) ) && ( e.Y >
    Grid.Y ) )
    {
        tfo = (_Day)Days[ (int)( ( e.Y - Header ) / y_grid_inc) ];
    }
}
```

```
}  
else if( TopLeft.Contains( e.X, e.Y ) )  
{  
tfo = TopLeft;  
}  
return tfo;  
}  
private void SelectCellBlock( Cell cellLast, Cell cellNew )  
{  
if( cellNew.X < cellLast.X )  
{  
if( cellNew.Y < cellLast.Y )  
{  
for( int i = cellNew.X; i <= cellLast.X; ++i )  
{  
for( int j = cellNew.Y; j <= cellLast.Y; ++j )  
{  
CellMatrix[ i, j ].Selected = cellLast.Selected;  
}  
}  
}  
else  
{  
for( int i = cellNew.X; i <= cellLast.X; ++i )  
{  
for( int j = cellLast.Y; j <= cellNew.Y; ++j )
```

```
{  
CellMatrix[ i, j ].Selected = cellLast.Selected;  
}  
}  
}  
}  
else  
{  
if( cellNew.Y < cellLast.Y )  
{  
for( int i = cellLast.X; i <= cellNew.X; ++i )  
{  
for( int j = cellNew.Y; j <= cellLast.Y; ++j )  
{  
CellMatrix[ i, j ].Selected = cellLast.Selected;  
}  
}  
}  
else  
{  
for( int i = cellLast.X; i <= cellNew.X; ++i )  
{  
for( int j = cellLast.Y; j <= cellNew.Y; ++j )  
{  
CellMatrix[ i, j ].Selected = cellLast.Selected;  
}  
}  
}
```

```
}  
  
}  
  
}  
  
}  
  
private void Draw( Graphics g )  
  
{  
  
g.FillRectangle( m_sbBack, 0,0,this.Width,this.Height );  
  
g.FillRectangle( m_sbGridBack, Grid );  
  
DrawTopLeft( g );  
  
DrawDays( g );  
  
DrawDates( g );  
  
DrawTimes( g );  
  
DrawTimeSlots( g );  
  
DrawGrid( g );  
  
DrawToolTip( g );  
  
}  
  
private void DrawToolTip( Graphics g )  
  
{  
  
if( ( ttInfo != null ) && ( LastMouseHover != null ) )  
  
{  
  
Font font = new Font( Control.DefaultFont.FontFamily, 8 );  
  
SizeF sf = g.MeasureString( ttInfo.Text, font );  
  
RectangleF lBounds = new RectangleF( LastMouseHover.X + 1,  
LastMouseHover.Y - ( 2 + sf.Height ), sf.Width, sf.Height + 1 );  
  
if( ( lBounds.Y < 0 ) && ( ( lBounds.X + lBounds.Width + c_ToolTipXPad ) >  
this.Width ) )  
  
{  
  
Adding week to user control.
```

```
lBounds.Y = LastMouseHover.Y;

lBounds.X = lBounds.X - ( lBounds.Width + 3 );

}

else if( ( lBounds.Y > 0 ) && ( ( lBounds.X + lBounds.Width +
c_ToolTipXPad ) > this.Width ) )

{

lBounds.X = lBounds.X - ( lBounds.Width + 3 );

}

else if( ( lBounds.Y < 0 ) && ( ( lBounds.X + lBounds.Width +
c_ToolTipXPad ) < this.Width ) )

{

lBounds.Y = LastMouseHover.Y;

lBounds.X += c_ToolTipXPad;

}

g.FillRectangle( m_sbInfoBack, lBounds );

g.DrawRectangle( Pens.Black, Rectangle.Round( lBounds ) );

g.DrawString( ttInfo.Text, font, m_sbInfoText, lBounds.X, lBounds.Y );

}

}

private void DrawTopLeft( Graphics g )

{

if( tfoLastHighlighted != null )

{

if( tfoLastHighlighted is TopLeftCell )

{

g.FillRectangle( m_sbHotTrack, tfoLastHighlighted.Bounds );

}

}

}

}
```

```
}  
}  
private void DrawGrid( Graphics g )  
{  
    for( int i = 0; i < NumDays; ++i )  
    {  
        for( int j = 0; j < NumTimeIntervals; ++j )  
        {  
            if( CellMatrix[ i, j ].Highlighted )  
            {  
                g.FillRectangle( m_sbHotTrack, CellMatrix[ i,j ].Bounds );  
            }  
            else if( CellMatrix[ i, j ].Selected )  
            {  
                g.FillRectangle( m_sbSelected, CellMatrix[ i,j ].Bounds );  
            }  
        }  
    }  
}  
  
//Draw Vertical Lines  
for( float i = 0; i <= NumTimeIntervals; ++i )  
{  
    if( ( i % TimeGridVerticalInterval ) == 0 )  
    {  
        if( ( i % TimeDisplayInterval ) == 0 )  
        {  
            {
```

```
g.DrawLine( Pens.Black, Grid.X + (i * x_grid_inc), 0, Grid.X + (i *
x_grid_inc), Grid.Y + Grid.Height );

}

else

{

g.DrawLine( Pens.Gray, Grid.X + (i * x_grid_inc), 0, Grid.X + (i *
x_grid_inc), Grid.Y + Grid.Height );

}

}

}

//Draw Horizontal Lines

for( float i = 0; i <= NumDays; ++i )

{

g.DrawLine( Pens.Black, 0, Grid.Y + (i * y_grid_inc), Grid.X + Grid.Width,
Grid.Y + (i * y_grid_inc) );

}

}

private void DrawDays( Graphics g )

{

Font font = new Font( Control.DefaultFont.FontFamily, FindEmSize( g, Days[
0 ].Bounds ) );

for( int i = 0; i < NumDays; ++i )

{

_Day d = Days[ i ];

SizeF sf = g.MeasureString( d.DOW.ToString(), font );

if( d.Highlighted )

{

g.FillRectangle( m_sbHotTrack, d.Bounds );
```

```
}

g.DrawString( d.DOW.ToString(), font, m_sbForeColor, LeftRectangle(
d.Bounds, sf ).Location );

}

}

private void DrawDates( Graphics g )

{

Font font = new Font( Control.DefaultFont.FontFamily, FindEmSize( g,
Dates[ 0 ].Bounds ) );

for( int i = 0; i < NumDates; ++i )

{

_Date d = Dates[ i ];

SizeF sf = g.MeasureString( d.DT.ToString(), font );

if( d.Highlighted )

{

g.FillRectangle( m_sbHotTrack, d.Bounds );

}

g.DrawString( d.DT.ToString(), font, m_sbForeColor, DateRectangle( d.Bounds,
sf ).Location );

}

}

private void DrawTimes( Graphics g )

{

Font font = new Font( Control.DefaultFont.FontFamily,
Control.DefaultFont.Size );

DateTime dt;

long t_inc = c_ts.Ticks / (long)NumTimeIntervals;

for( int i = 0; i <= NumTimeIntervals; ++i )

Adding week to user control.
```

```
{
if( ( i % TimeDisplayInterval ) == 0 )
{
dt = new DateTime( i * t_inc );
string strDate = dt.ToString( "htt" );
SizeF sf = g.MeasureString( strDate, Control.DefaultFont );
g.DrawString( strDate,
Control.DefaultFont, m_sbForeColor,
( Grid.X + ( i * x_grid_inc ) ) - ( sf.Width / 2.0F ), Grid.Y + Grid.Height +
1 );
}
}
}

private void DrawTimeSlots( Graphics g )
{
if( tfoLastHighlighted != null )
{
if( tfoLastHighlighted is TimeSlot )
{
g.FillRectangle( m_sbHotTrack, tfoLastHighlighted.Bounds.X, 0,
tfoLastHighlighted.Bounds.Width, tfoLastHighlighted.Bounds.Height );
}
}
}

private RectangleF CenterRectangle( RectangleF recLarge, SizeF sizeSmaller )
{
RectangleF recNew = new RectangleF( recLarge.Location, recLarge.Size );

```

```
recNew.X = recLarge.X + ( ( recLarge.Width / 2.0F ) - ( sizeSmaller.Width /  
2.0F ) );
```

```
recNew.Y = recLarge.Y + ( ( recLarge.Height / 2.0F ) - ( sizeSmaller.Height  
/ 2.0F ) );
```

```
return recNew;
```

```
}
```

```
private RectangleF DateRectangle( RectangleF recLarge, SizeF sizeSmaller )
```

```
{
```

```
RectangleF recNew = new RectangleF( recLarge.Location, recLarge.Size );
```

```
recNew.X = recLarge.X + ( ( recLarge.Width / 2.0F ) - ( sizeSmaller.Width /  
2.0F ) ) - 60.0F;
```

```
recNew.Y = recLarge.Y + ( ( recLarge.Height / 2.0F ) - ( sizeSmaller.Height  
/ 2.0F ) );
```

```
return recNew;
```

```
}
```

```
private RectangleF LeftRectangle( RectangleF recLarge, SizeF sizeSmaller )
```

```
{
```

```
RectangleF recNew = new RectangleF( recLarge.Location, recLarge.Size );
```

```
recNew.X = recLarge.X + 2.0F;
```

```
recNew.Y = recLarge.Y + 2.0F;
```

```
return recNew;
```

```
}
```

```
private float FindEmSize( Graphics g, RectangleF rec )
```

```
{
```

```
float i = 1.0F;
```

```
while( true )
```

```
{
```

```
Font f = new Font( Control.DefaultFont.FontFamily, i );
```

```
Adding week to user control.
```

microsoft.public.dotnet.languages.csharp: Adding week to user control.

```
SizeF sf = g.MeasureString( DayOfWeek.Wednesday.ToString(), f );  
  
if( ( sf.Width > rec.Width ) || ( sf.Height > rec.Height ) )  
{  
    break;  
}  
  
++i;  
}  
  
return i > 2 ? i - 1 : 1;  
}  
  
#endregion  
  
private void numericUpDown1_ValueChanged(object sender, System.EventArgs e)  
{  
    //this is where I want to handle the date changing.  
}  
}  
}
```