

```
using System ;
using System.ComponentModel ;
using System.Collections ;
using System.Diagnostics ;
using System.Windows.Forms ;
using System.Drawing ;
using System.Drawing.Drawing2D ;

namespace sb54_CSAI.MiscFormComponents
{
    #region MenuIcons CLASS

    /// <summary>
    /// A custom extender class that adds a MenuIcons
    /// attribute to MenuItem objects, and custom draws the menu
    /// with an icon stored in a referenced ImageList control.
    /// <br></br>
    /// <br></br>
    /// This extension was written to provide an simple way to link
    /// icons in an ImageList with a menu, and owner draw the menu.
    /// This could be accomplished using owner drawn menus, but that
    /// would require each menu to provide a DrawMenu() method.
    /// Using an extender, no custom coding is required. An extra
    /// property simply appears on each of the menu items which will
    /// run this code
    /// <br></br>
    /// <br></br>
    /// This class was written and using an article contained at the following URL
    /// http://www.codeproject.com/cs/menu/menuimage.asp
    /// <br></br>
    /// I have had to modify the source to carry out what I would condier to be correctly
    /// drawn XP style menus
    /// <br></br>
    /// GUI components are an area that interest me so I have included the source code in
    this namespace so that
    /// I could understand what was being done by the Author (Chris Beckett ) of the
    original classes.
    /// I could have simply used the Authors original C# Dll, but I would not have learned
    much that way
    /// so I chose to examine and modify (where necessary) the source code. Some of the
    names of classes are
    /// different here, as I actually wrote the code line by line, using the original
    article so that I could
    /// fully comprehend the original implementation. As such all the comments that are
    found within the code are by and large my own
    /// comments
    /// </summary>
    [ProvideProperty( "MenuIcons", typeof(Component)) ]
    [DefaultProperty("ImageList")]
    public class MenuIcons : Component, IExtenderProvider
    {
        #region Instance fields
        //Instance fields
        private const int IMAGE_BUFFER = 25 ;
        private const int SHORTCUT_RIGHTBUFFER = 10 ;
        private const int SHORTCUT_BUFFER = 25 ;
        private int IMAGE_WIDTH = SystemInformation.SmallIconSize.Width ;
        private int IMAGE_HEIGHT = SystemInformation.SmallIconSize.Height ;
        private Hashtable oHashTable = new Hashtable( ) ;
        private ImageList oImageList = null ;
        #endregion
        #region Public Constructors

        /// <summary>
        /// Constructor for instance that supports .NET designer.
        /// </summary>
        /// <param name="container">Reference to container hosting this instance.</param>
        public MenuIcons(System.ComponentModel.IContainer container)
        {
            //add this to the container (.NET designer)
            container.Add(this) ;
        }
    }
}
```

```
/// <summary>
/// Constructor for instance that doesn't support .NET designer.
/// </summary>
public MenuIcons()
{
}

#endregion
#region Public Methods

/// <summary>
/// Used to set a MenuIcons property value for
/// a specific MenuItem component instance.
/// </summary>
/// <param name="component">the MenuItem object to store</param>
/// <param name="indexValue">the image index value to associate with the menu item</
param>
public void SetMenuIcons( Component component, string indexValue )
{
    // check if the string value is null, or not numeric
    // automatically throws and error if not during the convert
    if (indexValue != null)
        if ( indexValue.Length > 0 )
        {
            uint imageIndex = Convert.ToUInt16( indexValue ) ;
        }

    // store the menuItem and related index in the local hashtable
    if ( oHashTable.Contains( component ) != true)
    {
        oHashTable.Add( component, indexValue ) ;
        MenuItem menuItem = (MenuItem) component ;

        // set the menu to owner drawn
        menuItem.OwnerDraw = true ;

        // hook up the menu owner drawn events
        menuItem.MeasureItem += new MeasureItemEventHandler( OnMeasureItem ) ;
        menuItem.DrawItem += new DrawItemEventHandler( OnDrawItem ) ;
    }
    else
    {
        oHashTable [ component ] = indexValue ;
    }
}

/// <summary>
/// Used to retrieve the MenuIcons extender property value
/// for a given MenuItem component instance.
/// </summary>
/// <param name="component">the menu item instance associated with the value</param>
/// <returns>Returns the MenuIcons index property value for the specified MenuItem
component instance.
/// </returns>
public string GetMenuIcons( Component component )
{
    if( oHashTable.Contains( component ) )
        return (string) oHashTable[ component ] ;

    return null;
}

/// <summary>
/// Used to determine if the given component is supported by
/// the extender.
/// </summary>
/// <param name="component">component to evaluate for compatability</param>
/// <returns>Returns True/False if the component supports the extender.</returns>
public bool CanExtend( object component )
{
    // only support MenuItem objects that are not
    // top-level menus (default rendering for top-level
    // menus is fine - does not need extension

```

```

    if ( component is MenuItem )
    {
        MenuItem menuItem = (MenuItem) component ;
        return ! ( menuItem.Parent is MainMenu ) ;
    }
    //default false, can not extend
    return false ;
}

/// <summary>
/// Gets or Sets the ImageList control that holds menu images.
/// </summary>
public ImageList ImageList
{
    get{ return oImageList ; }
    set{ oImageList = value; }
}

#endregion
#region Private Methods

/// <summary>
/// Performs a set of checks related to a menu image such as
/// a ImageList has been assigned, the image index is a valid
/// number and is within the ImageList images collection boundaries, etc.
/// </summary>
/// <param name="sender">the client object to retrieve the menuindex for</param></
param>
private int GetMenuIconsIndex( Object sender )
{
    string menuImageValue = this.GetMenuIcons( sender as Component ) ;

    // first check that the ImageList reference has been assigned
    // then verify the specified MenuIcons index is valid for the
    // imagelist. Then convert and return the index as an integer
    if ( oImageList != null )
        if ( menuImageValue != null )
            if ( menuImageValue.Length >= 0 )
            {
                int imageIndex = Convert.ToInt32( menuImageValue ) ;
                if ( imageIndex >= 0 && imageIndex < oImageList.Images.Count )
                    return imageIndex ;
            }

    return -1 ;
}

/// <summary>
/// Event triggered to measure the size of a owner drawn MenuItem.
/// <param name="sender">the menu item client object</param>
/// <param name="e">the event arguments</param>
private void OnMeasureItem( Object sender, MeasureItemEventArgs e )
{
    // retrieve the image list index from hash table
    MenuItem menuItem = (MenuItem) sender ;
    MenuDrawer MenuDrawer = new MenuDrawer( menuItem, e.Graphics, oImageList ) ;

    // calculate the menu height
    e.ItemHeight = MenuDrawer.CalcHeight() ;
    e.ItemWidth = MenuDrawer.CalcWidth() ;
}

/// <summary>
/// Event triggered to owner draw the provide MenuItem.
/// </summary>
/// <param name="sender">the menu item client object</param>
/// <param name="e">the event arguments</param>
private void OnDrawItem( Object sender, DrawItemEventArgs e )
{
    // derive the MenuItem object, and create the MenuDrawer
    MenuItem menuItem = (MenuItem) sender ;
    MenuDrawer MenuDrawer = new MenuDrawer( menuItem, e.Graphics, oImageList ) ;
}

```

```

    // draw the menu background
    bool menuSelected = (e.State & DrawItemState.Selected) > 0 ;
    MenuDrawer.DrawBackground( e.Bounds, menuSelected ) ;

    MenuDrawer.DrawBox(e.Bounds, menuSelected);

    if ( MenuDrawer.IsSeperator() == true )
        MenuDrawer.DrawSeperator( e.Bounds ) ;
    else
    {
        int imageIndex = this.GetMenuIconsIndex( sender ) ;
        MenuDrawer.DrawMenu( e.Bounds, menuSelected, imageIndex ) ;
    }
}

#endregion
#region MenuDrawer INNER Class

/// <summary>
/// Provides extra methods to allow all owner drawn menu items to be drawn correctly
///
/// Contains methods for drawing and measuring the menu item
/// <br></br>
/// <br></br>
/// This class was written and using an article contained at the following URL
/// http://www.codeproject.com/cs/menu/menuimage.asp
/// <br></br>
/// I have had to modify the source to carry out what I would condier to be
correctly
/// drawn XP style menus
/// <br></br>
/// GUI components are an area that interest me so I have included the source code
in this namespace so that
/// I could understand what was being done by the Author (Chris Beckett ) of the
original classes.
/// I could have simply used the Authors original C# Dll, but I would not have
learned much that way
/// so I chose to examine and modify (where necessary) the source code. Some of the
names of classes are
/// different here, as I actually wrote the code line by line, using the original
article so that I could
/// fully comprehend the original implementation. As such all the comments that are
found within the code are by and large my own
/// comments
/// </summary>
private class MenuDrawer
{
    #region Instance fields
    //Instance fields

    // some pre-defined buffer values for putting space between
    // icon, menutext, seperator text, and submenu arrow indicators
    private const int SEPERATOR_HEIGHT = 8 ;
    private const int SBORDER_WIDTH = 1 ;
    private const int BORDER_SIZE = SBORDER_WIDTH * 4 ;
    private const int SBUFFER_WIDTH = 5 ;
    private const int LBUFFER_WIDTH = 15 ;
    private const int SHORTCUT_BUFFER_SIZE = 20 ;
    private const int ARROW_WIDTH = 15 ;
    private int IMAGE_WIDTH = SystemInformation.SmallIconSize.Width ;
    private int IMAGE_HEIGHT = SystemInformation.SmallIconSize.Height ;
    private int IMAGE_BUFFER_SIZE = SystemInformation.SmallIconSize.Width + 10 ;
    private MenuItem oMenuItem = null ;
    private Graphics oGraphics = null ;
    private ImageList oImageList = null ;

    #endregion
    #region Public Constructors

    /// <summary>
    /// MenuDrawer constructor to assist in owner drawn menus.
    /// </summary>
    /// <param name="menuItem">a MenuItem object to custom draw</param>

```

```
    /// <param name="graphics">a Graphics object provided by the MeasureItem and DrawItem events</param>
    public MenuDrawer( MenuItem menuItem, Graphics graphics, ImageList imageList )
    {
        //set local fields
        oMenuItem = menuItem ;
        oGraphics = graphics ;
        oImageList = imageList ;
    }

    #endregion
    #region Public Members

    /// <summary>
    /// Based on the menu item text, and the SystemInformation.SmallIconSize,
    /// performs a calculation to determine the correct MenuItem height.
    /// </summary>
    /// <returns>Returns an int value that contains the calculated height of the
    menu item.</returns>
    public int CalcHeight()
    {
        // if the menu is a separator, then return a fixed height
        // otherwise calculate the menu size based on the system font
        // and smalliconsize calculations (with some added buffer values)
        if ( oMenuItem.Text == "-" )
            return SEPERATOR_HEIGHT ;
        else
        {
            // depending on which is longer, set the menu height to either
            // the icon, or the system menu font
            if ( SystemInformation.MenuFont.Height > SystemInformation.SmallIconSize
.Height )
                return SystemInformation.MenuFont.Height + BORDER_SIZE ;
            else
                return SystemInformation.SmallIconSize.Height + BORDER_SIZE ;
        }
    }

    /// <summary>
    /// Based on the menu item text, and the SystemInformation.SmallIconSize,
    /// performs a calculation to determine the correct MenuItem width.
    /// </summary>
    /// <returns>Returns an int value that contains the calculated width of the menu
    item.</returns>
    public int CalcWidth()
    {
        // prepare string formatting used for rendering menu caption
        StringFormat sf = new StringFormat() ;
        sf.HotkeyPrefix = System.Drawing.Text.HotkeyPrefix.Show ;

        // set the menu width by measuring the string, icon and buffer spaces
        int menuWidth = (int) oGraphics.MeasureString( oMenuItem.Text,
SystemInformation.MenuFont, 1000, sf).Width ;
        int shortcutWidth = (int) oGraphics.MeasureString( this.ShortcutText,
SystemInformation.MenuFont, 1000, sf).Width ;

        // if a top-level menu, no image support
        if ( this.IsTopLevel() == true )
            return menuWidth ;
        else
            return IMAGE_BUFFER_SIZE + menuWidth + SHORTCUT_BUFFER_SIZE +
shortcutWidth ;
    }

    /// <summary>
    /// A method to evaluate if the MenuItem has a shortcut selected, and the
    shortcut
    /// has been selected for show.
    /// </summary>
    /// <returns>Returns True/False whether the menu has a shortcut to be displayed.</returns>
    public bool HasShortcut()
    {
```

```

        return ( oMenuItem.ShowShortcut == true && oMenuItem.Shortcut != Shortcut.
None ) ;
    }

    /// <summary>
    /// Evaluates whether the MenuItem is a separator by evaluating the text.
    /// </summary>
    /// <returns>Returns True/False whether the menu is a separator.</returns>
    public bool IsSeperator()
    {
        return ( oMenuItem.Text == "-" ) ;
    }

    /// <summary>
    /// Evaluates whether the MenuItem is a top-level menu that is sited directly
    /// on a MainMenu control.
    /// </summary>
    /// <returns>Returns True/False if the menu item is a top-level menu.</returns>
    public bool IsTopLevel()
    {
        return ( oMenuItem.Parent is MainMenu ) ;
    }

    /// <summary>
    /// Formats the MenuItem and returns the shortcut selection
    /// as a displayable text string.
    /// </summary>
    public string ShortcutText
    {
        get
        {
            if ( oMenuItem.ShowShortcut == true && oMenuItem.Shortcut != Shortcut.
None )
            {
                Keys keys = (Keys) oMenuItem.Shortcut ;
                return Convert.ToChar(Keys.Tab) + System.ComponentModel.
TypeDescriptor.GetConverter(keys.GetType()).ConvertToString(keys) ;
            }
            return null ;
        }
    }

    /// <summary>
    /// Draws a small box under Image
    /// </summary>
    /// <param name="bounds">a Rectangle that holds the drawing canvas boundaries</
param>
    /// <param name="selected">True/False if the menu item is currently selected</
param>
    public void DrawBox (Rectangle bounds, bool selected )
    {
        // if selected then paint the menu as highlighted,
        // otherwise use the default menu brush
        if ( selected == true )
        {
            SolidBrush b = new SolidBrush(Color.LightBlue);
            oGraphics.FillRectangle( b ,1,bounds.Top+2, 18,bounds.Height-4);
        }
        else
        {
            SolidBrush b = new SolidBrush(Color.Gainsboro);
            oGraphics.FillRectangle( b ,bounds.X,bounds.Y, 20,22);
        }
    }

    /// <summary>
    /// Draws a normal menu item including any related icons, checkboxes,
    /// menu text, shortcuts text, and parent/submenu arrows.
    /// </summary>
    /// <param name="bounds">a Rectangle that holds the drawing canvas boundaries</
param>
    /// <param name="selected">True/False if the menu item is currently selected</

```

```

param>
    /// <param name="indexValue">the image index of the menu icon to draw, defaults
to -1</param>
    public void DrawMenu ( Rectangle bounds, bool selected, int indexValue )
    {
        // draw the menu text
        DrawMenuText( bounds, selected ) ;

        // since icons make the menu height longer,
        // paint a custom arrow if the menu is a parent
        // to augment the one painted by the control
        if ( oMenuItem.IsParent == true )
        {
            Image menuImage = null ;
            System.IO.Stream stream = this.GetType().Assembly.
GetManifestResourceStream("MiscFormComponents.SubItem16.ico") ;
            menuImage = Image.FromStream(stream) ;
            this.DrawArrow( menuImage, bounds ) ;
        }

        // if the menu item is checked, ignore any menuimage index
        // and draw the checkbox, otherwise draw the custom image
        if ( oMenuItem.Checked )
            DrawCheckBox ( bounds ) ;
        else
        {
            // see if the menu item has an icon associated and draw image
            if ( indexValue > -1 )
            {
                Image menuImage = null ;
                menuImage = oImageList.Images[indexValue] ;
                DrawImage( menuImage, bounds ) ;
            }
        }
    }

    /// <summary>
    /// Draws the MenuItem background.
    /// </summary>
    /// <param name="bounds">a Rectangle that holds the painting canvas boundaries</
param>
    /// <param name="selected">True/False if the menu item is currently selected</
param>
    public void DrawBackground( Rectangle bounds, bool selected )
    {
        // if selected then paint the menu as highlighted,
        // otherwise use the default menu brush
        if ( selected == true )
        {
            oGraphics.FillRectangle(new SolidBrush(Color.LightBlue),1,bounds.Top+1,
bounds.Width-2,bounds.Height-2);
            Pen p = new Pen(Color.CornflowerBlue);
            oGraphics.DrawRectangle(p,1,bounds.Top+1,bounds.Width-2,bounds.Height-2)
;
        }
        else
        {
            oGraphics.FillRectangle( SystemBrushes.Menu, bounds ) ;
            Pen p = new Pen(SystemBrushes.Menu);
            oGraphics.DrawRectangle(p,0,bounds.Top,bounds.Width,bounds.Height);
        }
    }

    /// <summary>
    /// Draws a menu seperator.
    /// </summary>
    /// <param name="bounds">a Rectangle that holds the drawing canvas boundaries</
param>
    public void DrawSeperator( Rectangle bounds )
    {
        SolidBrush b = new SolidBrush(Color.Gainsboro);
        oGraphics.FillRectangle( b ,bounds.Left,bounds.Top,20,bounds.Height);
    }

```

```

    // create the seperator line pen
    Pen pen = new Pen(SystemColors.ControlDark) ;

    // calculate seperator boundaries
    int xLeft = bounds.Left + IMAGE_BUFFER_SIZE ;
    int xRight = xLeft + bounds.Width ;
    int yCenter = bounds.Top + (bounds.Height / 2) ;

    // draw a seperator line and return
    oGraphics.DrawLine(pen, xLeft, yCenter, xRight, yCenter) ;
}

#endregion
#region Private Methods

    /// <summary>
    /// Draws the text for an ownerdrawn MenuItem.
    /// </summary>
    /// <param name="bounds">a Rectangle that holds the drawing area boundaries</
param>
    /// <param name="selected">True/False whether the menu item is currently
selected</param>
    private void DrawMenuText ( Rectangle bounds, bool selected )
    {
        // use system fonts and colors to select the menu brush so the menu
        // will appear correctly for any desktop theme
        Font menuFont = new Font("tahoma",8,FontStyle.Regular);

        SolidBrush menuBrush = null ;
        if ( oMenuItem.Enabled == false )
            menuBrush = new SolidBrush( SystemColors.GrayText ) ;
        else
        {
            if ( selected == true )
                menuBrush = new SolidBrush( SystemColors.HighlightText ) ;
            else
                menuBrush = new SolidBrush( SystemColors.MenuText ) ;
        }

        // draw the menu text
        StringFormat sfMenu = new StringFormat() ;
        sfMenu.HotkeyPrefix = System.Drawing.Text.HotkeyPrefix.Show ;
        oGraphics.DrawString( oMenuItem.Text, menuFont, menuBrush, bounds.Left +
IMAGE_BUFFER_SIZE, bounds.Top + ((bounds.Height - menuFont.Height) / 2), sfMenu ) ;

        // if the menu has a shortcut, then also
        // draw the shortcut right aligned
        if ( this.IsTopLevel() != true || this.HasShortcut() == false )
        {
            StringFormat sfShortcut = new StringFormat() ;
            sfShortcut.HotkeyPrefix = System.Drawing.Text.HotkeyPrefix.Show ;
            sfShortcut.FormatFlags |= StringFormatFlags.DirectionRightToLeft;
            int shortcutWidth = (int) oGraphics.MeasureString( this.ShortcutText,
menuFont, 1000, sfShortcut).Width ;
            oGraphics.DrawString(this.ShortcutText, menuFont, menuBrush, (bounds.
Width) - LBUFFER_WIDTH , bounds.Top + ((bounds.Height - menuFont.Height) / 2),
sfShortcut);
        }
    }

    /// <summary>
    /// Draws a checked item next to a MenuItem.
    /// </summary>
    /// <param name="bounds">a Rectangle that identifies the drawing area boundaries
</param>
    private void DrawCheckBox( Rectangle bounds )
    {
        // use the very handy ControlPaint object to paint
        // a checkbox.
        ButtonState btnState = ButtonState.Flat ;

        if ( oMenuItem.Checked == true )
            btnState = btnState | ButtonState.Checked ;
    }

```

```

        if ( oMenuItem.Enabled == false )
            btnState = btnState | ButtonState.Inactive ;

        // draw the checkbox
        ControlPaint.DrawCheckBox(oGraphics, bounds.Left + SBORDER_WIDTH, bounds.Top +
+ ((bounds.Height - IMAGE_HEIGHT) / 2), IMAGE_WIDTH, IMAGE_HEIGHT, btnState ) ;
    }

    /// <summary>
    /// Draws a provided image onto the MenuItem.
    /// </summary>
    /// <param name="menuImage">an Image to paint on the menu</param>
    /// <param name="bounds">a Rectangle that holds the drawing space boundaries</
param>
    private void DrawImage( Image menuImage, Rectangle bounds )
    {
        // if the menu item is enabled, then draw the image normally
        // otherwise draw it as disabled
        if ( oMenuItem.Enabled == true )
        {
            oGraphics.DrawImage(menuImage, bounds.Left + SBORDER_WIDTH, bounds.Top +
((bounds.Height - IMAGE_HEIGHT) / 2), IMAGE_WIDTH, IMAGE_HEIGHT ) ;
        }
        else
            ControlPaint.DrawImageDisabled(oGraphics, menuImage, bounds.Left +
SBORDER_WIDTH, bounds.Top + ((bounds.Height - IMAGE_HEIGHT) / 2), SystemColors.Menu ) ;
    }

    /// <summary>
    /// Draws a custom arrow on the right-side edge of the menu to indicate
    /// the menu has submenu items. Used to supplement a base contorl arrow
    /// that is painted incorrectly (seems to be a bug), and make the arrow
    /// appear correctly for longer menu items.
    /// </summary>
    /// <param name="menuImage">an Image to paint on the menu</param>
    /// <param name="bounds">a Rectangle that holds the drawing space boundaries</
param>
    private void DrawArrow( Image menuImage, Rectangle bounds )
    {
        //draw the arrow
        oGraphics.DrawImage(menuImage, bounds.Left + bounds.Width - ARROW_WIDTH,
bounds.Top + ((bounds.Height - IMAGE_HEIGHT) / 2), IMAGE_WIDTH, IMAGE_HEIGHT ) ;
    }

    #endregion
}
#endregion
}
#endregion
}

```