

13,007,145 members (61,337 online)

[Sign in](#)[home](#) [articles](#) [quick answers](#) [discussions](#) [features](#) [community](#) [help](#)[Articles](#) » [Languages](#) » [C#](#) » [Windows Forms](#)

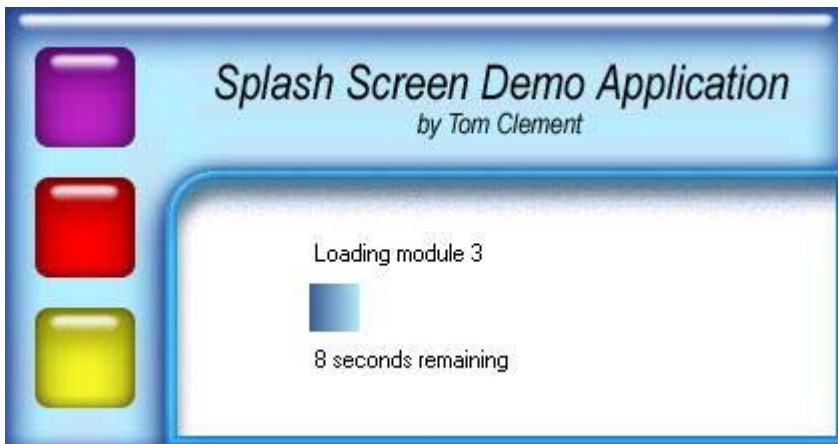
A Pretty Good Splash Screen in C#

**Tom Clement**, 28 Jun 2014

4.90 (305 votes)

Rate this:

A splash screen with some neat predictive progress bar features

[Download source and demo - 34.5 KB](#)

Introduction

Every time a customer loads your application, you have the opportunity to impress or disappoint with your splash screen. A good splash screen will:

- Run on a separate thread
- Fade in as it appears, and fade out as it disappears
- Display a running status message that is updated using a static method
- Display and update a predictive self-calibrating owner-drawn smooth-gradient progress bar
- Display the number of seconds remaining before load is complete

In this tutorial, we'll explore how to create a splash screen and add these features one at a time. We start with the creation of a simple splash screen, followed by the code changes required for the addition of each feature. You can skip to the bottom of the article to see the complete source code. I've also included a small test project in the download that demonstrates the splash screen.

Background

It was a lot of fun writing the code for this article and, while it's not perfect for all needs, I hope it saves you some coding time. The most fun for me is seeing a completely accurate and smoothly progressing progress bar as my application loads up. Please feel free to post any enhancement suggestions, bugs or other comments you may have.

Create the Simple Splash Screen Project

Start out by creating a Windows Forms project. Name it `SplashScreen`. Rename `Form1.cs` to `SplashScreen.cs`.

Now obtain a product bitmap with a light background suitable for putting text over. If you're lucky, a really talented person (like [dzCepheus](#) - see the thread below) will provide one for you. Set the Background Image property to it. Set the following properties on the form:

[Hide](#) [Copy Code](#)

```
FormBorderStyle = None
StartPosition = CenterScreen
```

In the form constructor, add the line:

[Hide](#) [Copy Code](#)

```
this.ClientSize = this.BackgroundImage.Size;
```

Go to the project properties and change the Output type to Class Library. At this point, if you are not adding this project to an existing solution, you may want to add another WinForms project to your solution for testing the splash screen. You can call the public methods from that class as you build the splash screen.

Make it Available from Static Methods

Because the splash screen will only need a single instance, you can simplify your code by using static methods to access it. By just referencing the `SplashScreen` project, a component can launch, update or close the splash screen without needing an object reference. Add the following code to `SplashScreen.cs`:

[Hide](#) [Copy Code](#)

```
static SplashScreen ms_frmSplash = null;
// A static entry point to launch SplashScreen.
static public void ShowForm()
{
    ms_frmSplash = new SplashScreen();
    Application.Run(ms_frmSplash);
}
// A static method to close the SplashScreen
static public void CloseForm()
{
    ms_frmSplash.Close();
}
```

Put it on its Own Thread

A splash screen displays information about your application while it is loading and initializing its components. If you are going to display any dynamic information during that time, you should put it on a separate thread to prevent it from freezing when initialization is hogging the main thread.

Start by using the Threading namespace:

[Hide](#) [Copy Code](#)

```
using System.Threading;
```

Declare a static variable to hold the thread:

[Hide](#) [Copy Code](#)

```
static Thread ms_oThread = null;
```

Now add a method to create and launch the splash screen on its own thread. Wait before returning to ensure that the static methods aren't called before the form exists:

[Hide](#) [Copy Code](#)

```
static public void ShowSplashScreen()
{
    // Make sure it is only launched once.
    if( ms_frmSplash != null )
        return;
    ms_oThread = new Thread( new ThreadStart(SplashScreen.ShowForm));
    ms_oThread.IsBackground = true;
    ms_oThread.SetApartmentState(ApartmentState.STA);
    ms_oThread.Start();
    while (ms_frmSplash == null || ms_frmSplash.IsHandleCreated == false)
    {
        System.Threading.Thread.Sleep(TIMER_INTERVAL);
    }
}
```

Now **ShowForm()** can be made private, since the form will now be shown using **ShowSplashScreen()**.

[Hide](#) [Copy Code](#)

```
// A static entry point to launch SplashScreen.
static private void ShowForm()
```

Add Code to Fade In and Fade Out

It can add real flair to your splash screen by having it fade in when it first appears, and fade out just as your application appears. The form's **Opacity** property makes this easy.

Declare variables defining increment and decrement rate. These define how quickly the form appears and disappears. They are directly related to the timer interval, since they represent how much the **Opacity** increases or decreases per timer tick, so if you modify the timer interval, you will want to change these proportionally.

[Hide](#) [Copy Code](#)

```
private double m_dblOpacityIncrement = .05;
private double m_dblOpacityDecrement = .08;
private const int TIMER_INTERVAL = 50;
```

Add a timer to the form, rename it to **UpdateTimer** then modify the constructor to start the timer and initialize the opacity to zero.

[Hide](#) [Copy Code](#)

```
this.Opacity = .0;
UpdateTimer.Interval = TIMER_INTERVAL;
UpdateTimer.Start();
```

Modify the **CloseForm()** method to initiate the fade away process instead of closing the form.

[Hide](#) [Copy Code](#)

```
static public void CloseForm()
{
    if( ms_frmSplash != null )
    {
```

```

    // Make it start going away.
    ms_frmSplash.m_dblOpacityIncrement = -ms_frmSplash.m_dblOpacityDecrement;
}
ms_oThread = null; // we do not need these any more.
ms_frmSplash = null;
}

```

Add a Tick event handler to change the opacity as the form is fading in or fading out, and to close the splash screen form when the opacity reaches 0.

[Hide](#) [Copy Code](#)

```

private void UpdateTimer_Tick(object sender, System.EventArgs e)
{
    if( m_dblOpacityIncrement > 0 )
    {
        if( this.Opacity < 1 )
            this.Opacity += m_dblOpacityIncrement;
        }
    else
    {
        if( this.Opacity > 0 )
            this.Opacity += m_dblOpacityIncrement;
        else
            this.Close();
        }
    }
}

```

At this point, you have a splash screen that fades into view when you call the **ShowSplashScreen()** method and starts fading away when you call the **CloseForm()** method.

Add Code to Display a Status String

Now that the basic splash screen is complete, we can add status information to the form, so the user can tell that something's going on. To do this, we add the member variable **m_sStatus** to the form to store the status and a label **lblStatus** to display it. We then add an accessor method to set the variable and modify the timer tick method to update the label. In general, it is illegal to update a UI element across threads, so, for example, an attempt to update a label on the splash screen from the main application thread will cause an exception (as of .NET 2.0). This code avoids that problem by updating data only in the status and progress updates. The timer event (on the splash screen thread) is used to do the UI updates, based on the member variables that were changed during the updates.

[Hide](#) [Copy Code](#)

```

private string m_sStatus;
...
// A static method to set the status.
static public void SetStatus(string newStatus)
{
    if( ms_frmSplash == null )
        return;
    ms_frmSplash.m_sStatus = newStatus;
}

```

Now we modify the **UpdateTimer_Tick** method to update the label.

[Hide](#) [Copy Code](#)

```

lblStatus.Text = m_sStatus;

```

Now Add a Progress Bar

There's no reason to use the standard WinForms progress bar here unless you really want that look. We'll make a gradient progress bar by painting our own Panel control. To do this, add a panel named **pnlStatus** to the form and set its

BackColor to **Transparent**. In practice, you might want to derive your own control from the Panel if you expect to use it in more than one place. Here, we'll just paint it in response to the timer event.

Declare a variable to hold the percent completion value. It is a double with a value that will vary between 0 and 1 as the progress bar progresses. Also declare a rectangle to hold the current progress rectangle.

[Hide](#) [Copy Code](#)

```
private double m_dblCompletionFraction = 0;
private Rectangle m_rProgress;
```

For now, add a public property for setting the current percent complete. Later, when we add the self-calibration feature, we'll eliminate the need for it.

[Hide](#) [Copy Code](#)

```
// Static method for updating the progress percentage.
static public double Progress
{
    get
    {
        if( ms_frmSplash != null )
            return ms_frmSplash.m_dblCompletionFraction;
        return 1.0;
    }
    set
    {
        if( ms_frmSplash != null )
            ms_frmSplash.m_dblCompletionFraction = value;
    }
}
```

Now we modify the timer's Tick event handler to calculate the portion of the **Panel** we want to paint.

[Hide](#) [Copy Code](#)

```
...
int width = (int)Math.Floor(pnlStatus.ClientRectangle.Width
    * m_dblCompletionFraction);
int height = pnlStatus.ClientRectangle.Height;
int x = pnlStatus.ClientRectangle.X;
int y = pnlStatus.ClientRectangle.Y;
if( width > 0 && height > 0 )
{
    m_rProgress = new Rectangle( x, y, width, height);
}
...
```

Now, still in the **UpdateTimer_Tick** event handler, paint the gradient progress bar. Start by adding the **System.Drawing.Drawing2D** namespace. You will probably want to fiddle with the RGB values to get a color scheme that works with your graphic.

[Hide](#) [Copy Code](#)

```
using System.Drawing.Drawing2D;
// Draw the progress bar.
if( e.ClipRectangle.Width > 0 && m_iActualTicks > 1 )
{
    LinearGradientBrush brBackground =
        new LinearGradientBrush(m_rProgress,
            Color.FromArgb(50, 50, 200),
            Color.FromArgb(150, 150, 255),
            LinearGradientMode.Horizontal);
    e.Graphics.FillRectangle(brBackground, m_rProgress);
}
}
```

Smooth the Progress by Extrapolating Between Progress Updates

I don't know about you, but I've always been annoyed by the way progress bars progress. They're jumpy, stop during long operations, and always cause me vague anxiety that maybe they've *stopped responding*.

Well, this next bit of code tries to alleviate that anxiety by making the progress bar move even during lengthy operations. We do this by changing the meaning of the **Progress** updates. Instead of indicating current percent complete, they now indicate the percentage of time we expect the current activity to take before the next Progress update. For example, the first update might indicate that 25% of the total will pass before the second update. This allows us to use the timer to paint more and more of the status bar, up to and including 25% (but not beyond) while we are waiting for the next update. For now, we'll guess at how much to progress per timer tick. Later, we'll calculate this based on experience.

Add member variables to represent the previous progress and the amount to increment the progress bar per timer tick.

[Hide](#) [Copy Code](#)

```
private double m_dbllastCompletionFraction = 0.0;
private double m_dblPBIncrementPerTimerInterval = .015;
```

Modify the **Progress** property to save the previous value before setting the new Progress value.

[Hide](#) [Copy Code](#)

```
ms_frmSplash.m_dbllastCompletionFraction =
    ms_frmSplash.m_dblCompletionFraction;
```

Modify the **Timer.Tick** event handler to do the progressive update:

[Hide](#) [Copy Code](#)

```
if( m_dbllastCompletionFraction < m_dblCompletionFraction )
{
    m_dbllastCompletionFraction += m_dblPBIncrementPerTimerInterval;
    int width = (int)Math.Floor(pnlStatus.ClientRectangle.Width
        * m_dbllastCompletionFraction);
    int height = pnlStatus.ClientRectangle.Height;
    int x = pnlStatus.ClientRectangle.X;
    int y = pnlStatus.ClientRectangle.Y;
    if (width > 0 && height > 0) // Paint progress bar
    {
        ...
    }
}
```

Now Make the Progress Bar Calibrate Itself

We can now eliminate the need to specify the progress percentages by calculating the values and remembering them between splash screen invocations. Notice that this will work only if you make a fixed sequence of calls to **SetStatus()** and **SetReferencePoint()** during startup.

XML Storage

You can use any persistent storage mechanism for remembering these values (there are only 2 strings to store) between invocations. Here, we'll use an XML file stored in a user-specific location that's writable even when a user is logged in with non-administrative privileges. (In an earlier version of this article, we used the windows registry.)

[Hide](#) [Shrink](#) [▲](#) [Copy Code](#)

```
using System.Xml;
...
```

```
internal class SplashScreenXMLStorage
{
    private static string ms_StoredValues = "SplashScreen.xml";
    private static string ms_DefaultPercents = "";
    private static string ms_DefaultIncrement = ".015";
    // Get or set the string storing the percentage complete at each checkpoint.
    static public string Percents
    {
        get { return GetValue("Percents", ms_DefaultPercents); }
        set { SetValue("Percents", value); }
    }
    // Get or set how much time passes between updates.
    static public string Interval
    {
        get { return GetValue("Interval", ms_DefaultIncrement); }
        set { SetValue("Interval", value); }
    }

    // Don't use the installation directory for the XML file - it's not always writable
    static private string StoragePath
    {
        get {return Path.Combine(Application.UserAppDataPath, ms_StoredValues);}
    }

    // Helper method for getting inner text of named element.
    static private string GetValue(string name, string defaultValue)
    {
        if (!File.Exists(StoragePath))
            return defaultValue;

        try
        {
            XmlDocument docXML = new XmlDocument();
            docXML.Load(StoragePath);
            XmlElement elValue = docXML.DocumentElement.SelectSingleNode(name) as XmlElement;
            return (elValue == null) ? defaultValue : elValue.InnerText;
        }
        catch
        {
            return defaultValue;
        }
    }

    // Helper method to set inner text of named element. Creates document if it doesn't exist
    static public void SetValue(string name, string stringValue)
    {
        XmlDocument docXML = new XmlDocument();
        XmlElement elRoot = null;
        if (!File.Exists(StoragePath))
        {
            elRoot = docXML.CreateElement("root");
            docXML.AppendChild(elRoot);
        }
        else
        {
            docXML.Load(StoragePath);
            elRoot = docXML.DocumentElement;
        }
        XmlElement value = docXML.DocumentElement.SelectSingleNode(name) as XmlElement;
        if (value == null)
        {
            value = docXML.CreateElement(name);
            elRoot.AppendChild(value);
        }
        value.InnerText = stringValue;
        docXML.Save(StoragePath);
    }
}
```

Member Variables

Now declare variables for keeping track of how long each interval between updates is taking (this time) and what it took per interval last time (from the XML file). Declare some Boolean flags to indicate whether this is the first launch and the timer has been started.

[Hide](#) [Copy Code](#)

```
// Self-calibration support
private int m_iIndex = 1;
private int m_iActualTicks = 0;
private ArrayList m_alPreviousCompletionFraction;
private ArrayList m_alActualTimes = new ArrayList();
private DateTime m_dtStart;
private bool m_bFirstLaunch = false;
private bool m_bDTSet = false;
```

Reference Points

We need to declare methods for recording various reference points during application startup. Reference points are critical to making a self-calibrating progress bar since they replace progress bar percent-complete updates. (The percent complete for each reference point will be calculated from the prior invocation of the progress bar.) To make the best use of this capability, you should sprinkle reference points inside of the initialization code that runs during application startup. The more you place, the smoother and more accurate your progress bar will be. This is when static access really pays off, because you don't need a reference to **SplashScreen** to call them.

First, we'll need a simple utility function to return elapsed Milliseconds since the Splash Screen first appeared. This is used for calculating the percentage of overall time allocated to each interval between ReferencePoint calls.

[Hide](#) [Copy Code](#)

```
// Utility function to return elapsed Milliseconds since the
// SplashScreen was launched.
private double ElapsedMilliseconds()
{
    TimeSpan ts = DateTime.Now - m_dtStart;
    return ts.TotalMilliseconds;
}
```

Now we'll be modifying **SetStatus()** and adding a new **SetReferencePoint()** method. Both call **SetReferenceInternal()** which records the time of the first call and adds the elapsed time of each subsequent call to an array for later processing. It sets the progress bar values by referencing previous recorded values for the progress bar. For example, if we're processing the third **SetReferencePoint()** call, we use the actual percentage of the overall load time that occurred between the third and fourth calls during the previous invocation. First, add the **SetReferencePoint()** method and **SetReferenceInternal()** which does the work of recording the time taken since the splash screen was started.

[Hide](#) [Shrink](#) ▲ [Copy Code](#)

```
// Static method called from the initializing application to
// give the splash screen reference points. Not needed if
// you are using a lot of status strings.
static public void SetReferencePoint()
{
    if( ms_frmSplash == null )
        return;
    ms_frmSplash.SetReferenceInternal();
}

// Internal method for setting reference points.
private void SetReferenceInternal()
{
    if (m_bDTSet == false)
    {
        m_bDTSet = true;
    }
}
```



```

        m_dtStart = DateTime.Now;
        ReadIncrements();
    }
    double dblMilliseconds = ElapsedMilliseconds();
    m_alActualTimes.Add(dblMilliseconds);
    m_dblLastCompletionFraction = m_dblCompletionFraction;
    if (m_alPreviousCompletionFraction != null && m_iIndex < m_alPreviousCompletionFraction.Count)
        m_dblCompletionFraction = (double)m_alPreviousCompletionFraction[m_iIndex++];
    else
        m_dblCompletionFraction = (m_iIndex > 0) ? 1 : 0;
}

```

The next two functions, **ReadIncrements()** and **StoreIncrements()**, read and write the calculated intervals associated with each of the ReferencePoint values.

Hide Shrink ▲ Copy Code

```

// Function to read the checkpoint intervals from the previous invocation of the
// splashscreen from the XML file.
private void ReadIncrements()
{
    string sPBIncrementPerTimerInterval = SplashScreenXMLStorage.Interval;
    double dblResult;

    if (Double.TryParse(sPBIncrementPerTimerInterval,
        System.Globalization.NumberStyles.Float,
        System.Globalization.NumberFormatInfo.InvariantInfo, out dblResult) == true)
        m_dblPBIncrementPerTimerInterval = dblResult;
    else
        m_dblPBIncrementPerTimerInterval = .0015;

    string sPBPreviousPctComplete = SplashScreenXMLStorage.Percents;

    if (sPBPreviousPctComplete != "")
    {
        string[] aTimes = sPBPreviousPctComplete.Split(null);
        m_alPreviousCompletionFraction = new ArrayList();

        for (int i = 0; i < aTimes.Length; i++)
        {
            double dblVal;
            if (Double.TryParse(aTimes[i], System.Globalization.NumberStyles.Float,
                System.Globalization.NumberFormatInfo.InvariantInfo, out dblVal))
                m_alPreviousCompletionFraction.Add(dblVal);
            else
                m_alPreviousCompletionFraction.Add(1.0);
        }
    }
    else
    {
        m_bFirstLaunch = true;
    }
}

// Method to store the intervals (in percent complete) from the current invocation of
// the splash screen to XML storage.
private void StoreIncrements()
{
    string sPercent = "";
    double dblElapsedMilliseconds = ElapsedMilliseconds();
    for (int i = 0; i < m_alActualTimes.Count; i++)
        sPercent += ((double)m_alActualTimes[i] /
            dblElapsedMilliseconds).ToString("0.####",
            System.Globalization.NumberFormatInfo.InvariantInfo) + " ";

    SplashScreenXMLStorage.Percents = sPercent;
    m_dblPBIncrementPerTimerInterval = 1.0 / (double)m_iActualTicks;
    SplashScreenXMLStorage.Interval =
        m_dblPBIncrementPerTimerInterval.ToString("#.000000",

```

```

        System.Globalization.NumberFormatInfo.InvariantInfo);
    }

```

We now can modify the **SetStatus()** method to add a Reference when the Status is updated. We also add an overloaded method to permit a Status update without the **SetReferenceInternal()** call. This is useful if you are in a section of code that has a variable set of status string updates. Note that depending on how often **SetStatus()** is called, you may not need many **SetReference()** calls in your startup code.

[Hide](#) [Copy Code](#)

```

// A static method to set the status and update the reference.
static public void SetStatus(string newStatus)
{
    SetStatus(newStatus, true);
}

// A static method to set the status and optionally update the reference.
// This is useful if you are in a section of code that has a variable
// set of status string updates. In that case, don't set the reference.
static public void SetStatus(string newStatus, bool setReference)
{
    if (ms_frmSplash == null)
        return;
    ms_frmSplash.m_sStatus = newStatus;
    if (setReference)
        ms_frmSplash.SetReferenceInternal();
}

```

We also need to modify the timer tick handler to paint only when **m_bFirstLaunch** is **false**. This prevents the first launch from showing an uncalibrated progress bar.

[Hide](#) [Copy Code](#)

```

...
// Paint progress bar
if(m_bFirstLaunch == false && m_dbfLastCompletionFraction < m_dbfCompletionFraction)
...

```

Add a Time Remaining Counter

Finally, we can fairly accurately estimate the remaining time for initialization by examining what percentage is yet to be done. Add a label called **lblTimeRemaining** to the splash screen form to display it. Add a member variable **m_sTimeRemaining** to hold the corresponding string, then add the following code to the **UpdateTimer_Tick()** event handler to update the **lblTimeRemaining** label on the SplashScreen form.

[Hide](#) [Copy Code](#)

```

private string m_sTimeRemaining;
...
private void UpdateTimer_Tick(object sender, System.EventArgs e)
{
    ...
    int iSecondsLeft = 1 + (int)(TIMER_INTERVAL * ((1.0 - m_dbfLastCompletionFraction) /
m_dbfPBIncrementPerTimerInterval)) / 1000;
    m_sTimeRemaining = (iSecondsLeft == 1) ? string.Format("1 second remaining") :
string.Format("{0} seconds remaining", iSecondsLeft);
    ...
    lblTimeRemaining.Text = m_sTimeRemaining;
}

```

Also modify the **ReadIncrements()** method which clears the time remaining label as follows:

[Hide](#) [Copy Code](#)

```

private void ReadIncrements()
{

```

```
...
    m_sTimeRemaining = "";
...
}
```

Note that the label is set using a member variable. This makes it possible to set the time remaining text across threads without causing a cross-thread exception.

Using the SplashScreen

To use the splash screen, just call **SplashScreen.ShowSplashScreen()** on the first line of your **Main()** entry point. Periodically call either **SetStatus()** (if you have a new status to report) or **SplashScreen.SetReferencePoint()** (if you don't) to calibrate the progress bar. When your initialization is complete, call **SplashScreen.CloseForm()** to start the fade out process. Take a look at the test module provided in the download if you have any questions.

You may want to play around with the various constants to adjust the time of fade in and fade out. If you set the interval to a very short time (like 10 ms), you'll get a beautiful smoothly progressing progress bar but your performance may suffer.

When the application first loads, you will notice that the progress bar and time remaining counter do not display. This is because the splash screen needs one load to calibrate the progress bar. It will appear on subsequent application launches.

SplashScreen.cs Source Code

[Hide](#) [Shrink](#) ▲ [Copy Code](#)

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.IO;
using System.Threading;
using System.Windows.Forms;
using System.Xml;
using Microsoft.Win32;
using System.Runtime.InteropServices;

namespace SplashScreen
{
    // The SplashScreen class definition. AKO Form
    public partial class SplashScreen : Form
    {
        #region Member Variables
        // Threading
        private static SplashScreen ms_frmSplash = null;
        private static Thread ms_oThread = null;

        // Fade in and out.
        private double m_dblOpacityIncrement = .05;
        private double m_dblOpacityDecrement = .08;
        private const int TIMER_INTERVAL = 50;

        // Status and progress bar
        private string m_sStatus;
        private string m_sTimeRemaining;
        private double m_dblCompletionFraction = 0.0;
        private Rectangle m_rProgress;

        // Progress smoothing
        private double m_dblLastCompletionFraction = 0.0;
        private double m_dblPBIncrementPerTimerInterval = .015;

        // Self-calibration support
```

```
private int m_iIndex = 1;
private int m_iActualTicks = 0;
private ArrayList m_alPreviousCompletionFraction;
private ArrayList m_alActualTimes = new ArrayList();
private DateTime m_dtStart;
private bool m_bFirstLaunch = false;
private bool m_bDTSet = false;
```

```
#endregion Member Variables
```

```
/// <summary>
/// Constructor
/// </summary>
public SplashScreen()
{
    InitializeComponent();
    this.Opacity = 0.0;
    UpdateTimer.Interval = TIMER_INTERVAL;
    UpdateTimer.Start();
    this.ClientSize = this.BackgroundImage.Size;
}
```

```
#region Public Static Methods
```

```
// A static method to create the thread and
// launch the SplashScreen.
```

```
static public void ShowSplashScreen()
{
    // Make sure it's only launched once.
    if (ms_frmSplash != null)
        return;
    ms_oThread = new Thread(new ThreadStart(SplashScreen.ShowForm));
    ms_oThread.IsBackground = true;
    ms_oThread.SetApartmentState(ApartmentState.STA);
    ms_oThread.Start();
    while (ms_frmSplash == null || ms_frmSplash.IsHandleCreated == false)
    {
        System.Threading.Thread.Sleep(TIMER_INTERVAL);
    }
}
```

```
// Close the form without setting the parent.
```

```
static public void CloseForm()
{
    if (ms_frmSplash != null && ms_frmSplash.IsDisposed == false)
    {
        // Make it start going away.
        ms_frmSplash.m_dblOpacityIncrement = -ms_frmSplash.m_dblOpacityDecrement;
    }
    ms_oThread = null; // we don't need these any more.
    ms_frmSplash = null;
}
```

```
// A static method to set the status and update the reference.
```

```
static public void SetStatus(string newStatus)
{
    SetStatus(newStatus, true);
}
```

```
// A static method to set the status and optionally update the reference.
// This is useful if you are in a section of code that has a variable
// set of status string updates. In that case, don't set the reference.
```

```
static public void SetStatus(string newStatus, bool setReference)
{
    if (ms_frmSplash == null)
        return;

    ms_frmSplash.m_sStatus = newStatus;

    if (setReference)
```

```
        ms_frmSplash.SetReferenceInternal();
    }

    // Static method called from the initializing application to
    // give the splash screen reference points. Not needed if
    // you are using a lot of status strings.
    static public void SetReferencePoint()
    {
        if (ms_frmSplash == null)
            return;
        ms_frmSplash.SetReferenceInternal();
    }
}
#endregion Public Static Methods

#region Private Methods

    // A private entry point for the thread.
    static private void ShowForm()
    {
        ms_frmSplash = new SplashScreen();
        Application.Run(ms_frmSplash);
    }

    // Internal method for setting reference points.
    private void SetReferenceInternal()
    {
        if (m_bDTSet == false)
        {
            m_bDTSet = true;
            m_dtStart = DateTime.Now;
            ReadIncrements();
        }
        double dblMilliseconds = ElapsedMilliseconds();
        m_alActualTimes.Add(dblMilliseconds);
        m_dblLastCompletionFraction = m_dblCompletionFraction;
        if (m_alPreviousCompletionFraction != null && m_iIndex
            < m_alPreviousCompletionFraction.Count )
            m_dblCompletionFraction = (double)m_alPreviousCompletionFraction[m_iIndex++];
        else
            m_dblCompletionFraction = (m_iIndex > 0) ? 1 : 0;
    }

    // Utility function to return elapsed milliseconds since the
    // SplashScreen was launched.
    private double ElapsedMilliseconds()
    {
        TimeSpan ts = DateTime.Now - m_dtStart;
        return ts.TotalMilliseconds;
    }

    // Function to read the checkpoint intervals from the previous invocation of the
    // splashscreen from the XML file.
    private void ReadIncrements()
    {
        string sPBIncrementPerTimerInterval = SplashScreenXMLStorage.Interval;
        double dblResult;

        if (Double.TryParse(sPBIncrementPerTimerInterval,
            System.Globalization.NumberStyles.Float,
            System.Globalization.NumberFormatInfo.InvariantInfo, out dblResult) == true)
            m_dblPBIncrementPerTimerInterval = dblResult;
        else
            m_dblPBIncrementPerTimerInterval = .0015;

        string sPBPreviousPctComplete = SplashScreenXMLStorage.Percents;

        if (sPBPreviousPctComplete != "")
        {
```

```
string[] aTimes = sPBPreviousPctComplete.Split(null);
m_alPreviousCompletionFraction = new ArrayList();

for (int i = 0; i < aTimes.Length; i++)
{
    double dblVal;
    if (Double.TryParse(aTimes[i],
        System.Globalization.NumberStyles.Float,
        System.Globalization.NumberFormatInfo.InvariantInfo, out dblVal) == true)
        m_alPreviousCompletionFraction.Add(dblVal);
    else
        m_alPreviousCompletionFraction.Add(1.0);
}
}
else
{
    m_bFirstLaunch = true;
    m_sTimeRemaining = "";
}
}

// Method to store the intervals (in percent complete) from the current invocation of
// the splash screen to XML storage.
private void StoreIncrements()
{
    string sPercent = "";
    double dblElapsedMilliseconds = ElapsedMilliseconds();
    for (int i = 0; i < m_alActualTimes.Count; i++)
        sPercent += ((double)m_alActualTimes[i] / dblElapsedMilliseconds)
            .ToString("0.####", System.Globalization.NumberFormatInfo.InvariantInfo) + " ";

    SplashScreenXMLStorage.Percents = sPercent;

    m_dblPBIIncrementPerTimerInterval = 1.0 / (double)m_iActualTicks;

    SplashScreenXMLStorage.Interval = m_dblPBIIncrementPerTimerInterval
        .ToString("#.000000", System.Globalization.NumberFormatInfo.InvariantInfo);
}

public static SplashScreen GetSplashScreen()
{
    return ms_frmSplash;
}

#endregion Private Methods

#region Event Handlers
// Tick Event for the Timer control. Handle fade in and fade out and paint progress bar.
private void UpdateTimer_Tick(object sender, EventArgs e)
{
    lblStatus.Text = m_sStatus;

    // Calculate opacity
    if (m_dblOpacityIncrement > 0)    // Starting up splash screen
    {
        m_iActualTicks++;
        if (this.Opacity < 1)
            this.Opacity += m_dblOpacityIncrement;
    }
    else // Closing down splash screen
    {
        if (this.Opacity > 0)
            this.Opacity -= m_dblOpacityIncrement;
        else
        {
            StoreIncrements();
            UpdateTimer.Stop();
            this.Close();
        }
    }
}
```

```

    }

    // Paint progress bar
    if (m_bFirstLaunch == false && m_dbLastCompletionFraction < m_dbCompletionFraction)
    {
        m_dbLastCompletionFraction += m_dbLPBIncrementPerTimerInterval;
        int width = (int)Math.Floor(pnlStatus.ClientRectangle.Width
                                    * m_dbLastCompletionFraction);
        int height = pnlStatus.ClientRectangle.Height;
        int x = pnlStatus.ClientRectangle.X;
        int y = pnlStatus.ClientRectangle.Y;
        if (width > 0 && height > 0)
        {
            m_rProgress = new Rectangle(x, y, width, height);
            if (!pnlStatus.IsDisposed)
            {
                Graphics g = pnlStatus.CreateGraphics();
                LinearGradientBrush brBackground =
                    new LinearGradientBrush(m_rProgress,
                                            Color.FromArgb(58, 96, 151),
                                            Color.FromArgb(181, 237, 254),
                                            LinearGradientMode.Horizontal);
                g.FillRectangle(brBackground, m_rProgress);
                g.Dispose();
            }
            int iSecondsLeft = 1 + (int)(TIMER_INTERVAL *
                                         ((1.0 - m_dbLastCompletionFraction) / m_dbLPBIncrementPerTimerInterval)) / 1000;
            m_sTimeRemaining = (iSecondsLeft == 1) ?
                               string.Format("1 second remaining") :
                               string.Format("{0} seconds remaining", iSecondsLeft);
        }
    }
    lblTimeRemaining.Text = m_sTimeRemaining;
}

// Close the form if they double click on it.
private void SplashScreen_DoubleClick(object sender, System.EventArgs e)
{
    // Use the overload that doesn't set the parent form to this very window.
    CloseForm();
}

#endregion Event Handlers
}

#region Auxiliary Classes
/// <summary>
/// A specialized class for managing XML storage for the splash screen.
/// </summary>
internal class SplashScreenXMLStorage
{
    private static string ms_StoredValues = "SplashScreen.xml";
    private static string ms_DefaultPercents = "";
    private static string ms_DefaultIncrement = ".015";

    // Get or set the string storing the percentage complete at each checkpoint.
    static public string Percents
    {
        get { return GetValue("Percents", ms_DefaultPercents); }
        set { SetValue("Percents", value); }
    }

    // Get or set how much time passes between updates.
    static public string Interval
    {
        get { return GetValue("Interval", ms_DefaultIncrement); }
        set { SetValue("Interval", value); }
    }
}

// Store the file in a location where it can be written with only User rights.

```

```

// (Don't use install directory).
static private string StoragePath
{
    get {return Path.Combine(Application.UserAppDataPath, ms_StoredValues);}
}

// Helper method for getting inner text of named element.
static private string GetValue(string name, string defaultValue)
{
    if (!File.Exists(StoragePath))
        return defaultValue;

    try
    {
        XmlDocument docXML = new XmlDocument();
        docXML.Load(StoragePath);
        XmlElement elValue = docXML.DocumentElement.SelectSingleNode(name) as XmlElement;
        return (elValue == null) ? defaultValue : elValue.InnerText;
    }
    catch
    {
        return defaultValue;
    }
}

// Helper method for setting inner text element. Creates XML file if it doesn't exist.
static public void SetValue(string name,
    string stringValue)
{
    XmlDocument docXML = new XmlDocument();
    XmlElement elRoot = null;
    if (!File.Exists(StoragePath))
    {
        elRoot = docXML.CreateElement("root");
        docXML.AppendChild(elRoot);
    }
    else
    {
        docXML.Load(StoragePath);
        elRoot = docXML.DocumentElement;
    }
    XmlElement value = docXML.DocumentElement.SelectSingleNode(name) as XmlElement;
    if (value == null)
    {
        value = docXML.CreateElement(name);
        elRoot.AppendChild(value);
    }
    value.InnerText = stringValue;
    docXML.Save(StoragePath);
}
}
#endregion Auxiliary Classes
}

```

Update

This article was originally written over 10 years ago, when the current version of .NET was 1.1. Over time, readers have reported a number of defects and suggestions for workarounds (in the comments below) that haven't, until now, been incorporated into the article. However, a few years back [Mahin Gupta](#) was kind enough to update the source code with fixes and changes based on those comments and had provided a link to that source that many folks were downloading and using. This update takes from that source code and incorporates the changes into the text of the article, as well as updating the attached zip file containing the code.

I'd like to thank Mahin, as well as the many folks who posted bug reports and fixes below. Some of the changes included in this update are:

- Fixed cross threading oversight (`lblTimeRemaining` label was being set directly.)
- Alternative storage to avoid the windows registry (Colin Stansfield, Lagrange)
- Delay updates until the splash screen thread is started (daws)
- `timer.Stop()` before closing the form (kilroytrout)
- Draw the progress bar during the timer tick event (kilroytrout, [cbschuld](#))
- Remaining time accuracy improvements ([Pacman69](#))
- Double click to dismiss splash screen fault (Natural_Demon)

I didn't implement one suggestion. A few of the comments pointed out focus problems when the splash screen closes (in MDI applications). The suggestion was to set the owner of the splash screen to the appropriate parent form as part of closing the window. I tried all the proposed fixes (I think) and couldn't find something that works. The problem appears to be that setting a form's owner invokes code on the owner (`AddOwnedForm`) which is not thread safe. So even if you do it during an invoke, the operation fails with a cross thread exception. The only 'fix' for this I could find online was to turn off the exception (`Control.CheckForIllegalCrossThreadCalls = false;`) during the operation. I decided that might not be wise. So either cross thread form ownership is not possible, or I can't figure it out. Any suggestions for resolving this would be appreciated. In the test project, I found that calling `this.Activate()` before calling `SplashScreen.CloseForm()` was sufficient.

History

- 11-16-2003 First Version.
- 11-18-2003 Corrected some typos and clarified behavior when the application is first called. Changed code to not display the progress bar on the first load.
- 11-20-2003 Added improvements and bug fixes based on Quentin Pouplard's comments (below).
- 12-23-2003 Added the graphic provided for us by dzCepheus (below).
- 7-28-2013 Reviewed suggestions and bug fixes provided in the comments section and updated accordingly.

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

Share

EMAIL

About the Author



Tom Clement

Product Manager

United States 

I've been programming in C, C++, Visual Basic and C# for over 30 years. I've worked at Sierra Systems, ViewStar, Mosaix, Lucent, Avaya, Avinon, Apptero, Serena and now Guidewire Software in various roles over my career.

You may also be interested in...



Splash Screen Control



Generate and add keyword variations using AdWords API



Yet Another Splash Screen in C#



Window Tabs (WndTabs) Add-In for DevStudio



SAPrefs - Netscape-like Preferences Dialog



WTL for MFC Programmers, Part IX - GDI Classes, Common Dialogs, and Utility Classes

Comments and Discussions

You must [Sign In](#) to use this message board.

Search Comments

Go

Spacing

Tight

Layout













































































Thread View

Per page 50

Update

First Prev Next

	Splashtscreen not topmost?	Member 11006677	7-Jun-17 4:20	1
	Solution to the problem of main form jumping to the background after Splash closes	toolsmythe	17-Nov-15 9:18	3
	Change Background Image	Patrik Bindea	24-Jun-15 23:38	2
	Problem: Mainform remains in background after splashtscreen closes	Mario M.	14-May-15 8:45	3
	Can this be fixed?	HighProSoft	12-Jan-15 11:04	4
	Very Deep work.	Bertin @nonodata.com	3-Dec-14 19:21	3
	it's nice but has problem!!!!	mm.ebrahimi	16-Oct-14 7:44	2
	Love it! Question: Putting Splash Screen Top Most (i.e. always in foreground)	Member 8753121	13-Sep-14 7:37	1
	Main form not focused/maximized after closing the splash screen problem.	oatsoda	8-Sep-14 0:23	1
	It's just me or we can't call the form from the other project even in one Solution ?	Tommy Aria Pradana	28-Jul-14 3:42	2
	one project	Ye Htut	26-Jul-14 21:21	2
	Splash screen and Mdi App.	devnet247	18-Jul-14 19:07	1
	Very nice splash	Degryse Kris	14-Jul-14 23:10	2
	Is there any way to minimize or hide the splash window while the main window is minimized?	Michael Chao	30-Jun-14 5:52	5
	My vote of 5	prashita gupta	29-Jun-14 20:30	2

 My vote of 5 	 jayveebishie	28-Jun-14 22:45	2
 Great tool, very impressive!!, Ist there a possibility to play a video instead of showing the image 	 Pfotenhauer	2-Jun-14 4:56	4
 splash screen architecture 	 avisal	26-May-14 9:30	3
 Independently closing Splash Form 	 Member 8082126	22-May-14 8:49	1
 Well begun, half done! 	 Romano Scuri	22-May-14 3:43	2
 My vote of 5 	 Humayun Kabir Mamun	21-May-14 19:13	2
 Splashscreen causes images in imagelist not to be displayed 	 kolmo	23-Apr-14 4:21	2
 My vote of 6! 	 Bharat Mallapur	21-Mar-14 2:49	2
 My vote of 5 	 Bharat Mallapur	21-Mar-14 2:46	1
 Fade in-out issues 	 brennt	5-Nov-13 4:39	3
 First time run is different, and I wonder why 	 Michael Chao	24-Sep-13 8:09	3
 My vote of 5 	 Member 2262881	29-Aug-13 1:40	2
 My vote of 5 	 Rachelle Villar	22-Aug-13 20:51	2
 My vote of 5 	 Jellico2nd	30-Jul-13 23:48	2
 My vote of 5 	 Serge-USA	30-Jul-13 10:42	2
 Thanks to Smitha Vijayan for edits and cleanup of the article 	 Tom Clement	29-Jul-13 9:04	1
 Updated source is now available 	 Tom Clement	28-Jul-13 15:29	3
 Looks very useful, but can't get a working copy... 	 Tomee	24-Jul-13 4:26	5
 My vote of 5 	 stuartmkeymark	3-Jul-13 4:56	2
 Animated GIF dont work 	 arzamm@hotmail.com	3-Apr-13 6:03	2
 cross threading calls 	 santa239	22-Feb-13 8:57	1
 My vote of 5 	 Sonam K	3-Feb-13 19:46	1
 invoked on set status & set time remaning 	 han6man	6-Jan-13 21:02	1
 new Thread(new ThreadStart(SplashScreen.ShowForm)); ore new Thread(new ThreadStart(ShowForm)); 	 han6man	25-Oct-12 4:31	2
 Status / Time Remaining showing on test PC, but not on another PC? 	 MattAlex151	20-Sep-12 4:59	3
 Thanks - an an update... 	 pt1401	25-Jul-12 2:14	3
 Excellent demonstration 	 sami zahid	12-Jul-12 18:06	2
 My vote of 5 	 Md. Marufuzzaman	2-Jun-12 23:46	2
 My vote of 5 	 onurag19	4-May-12 20:16	1
 Is there a way to use this in a WPF application? 	 stevefer	29-Mar-12 18:41	1
 Form minimize issue after splash close (.net 3.5 version) 	 mehmetbulus	1-Mar-12 2:39	3
 My vote of 5 	 Didil-1	24-Dec-11 6:24	3
 [My vote of 1] Unbelievably bad 	 clarkybravo	4-Dec-11 7:30	3
 My vote of 5 	 DuffmanLight	6-Nov-11 11:24	2
 I've noticed this is being downloaded quite often recently (20 times a day or so) 	 Tom Clement	15-Sep-11 15:25	4

Last Visit: 31-Dec-99 18:00 Last Update: 29-Jun-17 1:12

[Refresh](#)[1](#) [2](#) [3](#) [4](#) [Next »](#)
[General](#)
[News](#)
[Suggestion](#)
[Question](#)
[Bug](#)
[Answer](#)
[Joke](#)
[Praise](#)
[Rant](#)
[Admin](#)
[Permalink](#) | [Advertise](#) | [Privacy](#) | [Terms of Use](#) | [Mobile](#)
 Web02 | 2.8.170628.1 | Last Updated 28 Jun 2014

Sélectionner une langue ▼

Layout: [fixed](#) | [fluid](#)

Article Copyright 2003 by Tom Clement
Everything else Copyright © [CodeProject](#), 1999-2017