

How to read barcodes from live video cam (wpf) in C# with ByteScout BarCode Reader SDK

Tutorial on how to read barcodes from live video cam (wpf) in C#

On this page you will learn from code samples for programming in C#. Writing of the code to read barcodes from live video cam (wpf) in C# can be done by developers of any level using ByteScout BarCode Reader SDK. ByteScout BarCode Reader SDK is the SDK for barcode decoding. Can read all popular types from Code 128, GS1, UPC and Code 39 to QR Code, Datamatrix, PDF417. Images, pdf, TIF images and live web camera are supported as input. Designed to handle documents with noise and defects. Includes optional splitter and merger for pdf and tiff based on barcodes. Batch mode is optimized for high performance with multiple threads. Decoded values can be exported to XML, JSON, CSV or into custom data format. It can read barcodes from live video cam (wpf) in C#.

This rich sample source code in C# for ByteScout BarCode Reader SDK includes the number of functions and options you should do calling the API to read barcodes from live video cam (wpf). This C# sample code is all you need for your app. Just copy and paste the code, add references (if needs to) and you are all set! Implementing C# application typically includes multiple stages of the software development so even if the functionality works please test it with your data and the production environment.

Download free trial version of ByteScout BarCode Reader SDK from our website with this and other source code samples for C#.

FOR MORE INFORMATION AND FREE TRIAL:

[Download Free Trial SDK \(on-premise version\)](#)

[Read more about ByteScout BarCode Reader SDK](#)

[Explore API Documentation](#)

[Get Free Training for ByteScout BarCode Reader SDK](#)

[Get Free API key for Web API](#)

[visit www.Bytescout.com](http://www.Bytescout.com)

Source Code Files:

App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Windows;

namespace ReadFromVideoCamera.VS2010.WPF
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {

    }
}
```

VideoScanForm.xaml.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Reflection;
using System.Threading;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Threading;
using Bytescout.BarCodeReader;
using TouchlessLib;

namespace ReadFromVideoCamera.VS2010.WPF
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
```

```

{
    // Touchless lib manager object (to use it you should have TouchlessLib.dll and
    private TouchlessMgr _touchlessMgr;

    // USED IN POPUP MODE ONLY (see ShowScanPopup() method)
    // Close or not on the first barcode found
    // (results are saved in _foundBarcodes)
    public bool CloseOnFirstBarcodeFound { get; set; }

    // Indicates if the form is closed
    public bool IsClosed { get; set; }

    // Background processing object
    BackgroundWorker _backgroundWorker = new BackgroundWorker();

    // Barcode type to scan
    private BarcodeTypeSelector _barcodeTypeToFind = new BarcodeTypeSelector();

    // Array with decoded barcodes from the last scanning session
    public FoundBarcode[] FoundBarcodes { get; set; }

    // Scanning delay (ms); default is to scan every 800 ms.
    const int ScanDelay = 800;

    // Internal variable to indicate the status.
    public static bool Status = true;

    public delegate void SimpleDelegate();

    /// <summary>
    /// Creates the form.
    /// </summary>
    public MainWindow()
    {
        InitializeComponent();

        lblScanning.Visibility = Visibility.Collapsed;

        _backgroundWorker.WorkerSupportsCancellation = true;
        _backgroundWorker.DoWork += BackgroundWorker_DoWork;
        _backgroundWorker.RunWorkerCompleted += BackgroundWorker_RunWo
    }

    // Searches for barcodes in bitmap object
    private FoundBarcode[] FindBarcodes(Bitmap bitmap)
    {
        Reader reader = new Reader();

        try
        {
            reader.RegistrationName = "demo";
            reader.RegistrationKey = "demo";

            this.Dispatcher.Invoke(DispatcherPriority.Normal, (SimpleDelegate) Upd

            reader.BarcodeTypesToFind = _barcodeTypeToFind;

            //reader.MaxNumberOfBarcodesPerPage = 1;

```

```

/* -----
NOTE: We can read barcodes from specific page to increase performance.
For sample please refer to "Decoding barcodes from PDF by pages" program
-----

FoundBarcode[] result = reader.ReadFrom(bitmap);
String timeNow = string.Format("{0:HH:mm:ss:tt}", DateTime.Now);

        this.Dispatcher.Invoke(DispatcherPriority.Normal, (SimpleAction)
    {
        if (result != null && result.Length > 0)
        {

            textAreaBarcodes.SelectAll();
            textAreaBarcodes.Selection.Text = "\nTime: " + timeNow +

                // insert barcodes into text area
            foreach (FoundBarcode barcode in result)
            {

                // make a sound that indicates barcode found
                Console.Beep();
                // form the string with barcode value
                String barcodeValue = String.Format("Found: {0} {1}\n", barcode.Type, barcode.Value);
                // add barcode to the text area output
                textAreaBarcodes.AppendText(barcodeValue + "\n");
                // add barcode to the list of saved barcodes
                lblFoundBarcodes.Content = String.Format("Found {0} barcodes", result.Length);
            }
        }

        // make "Scanning.." label flicker
        lblScanning.Visibility = lblScanning.Visibility == Visibility.Visible
            ? Visibility.Visible
            : Visibility.Collapsed;

        lblScanning.UpdateLayout();
    });

        // return found barcodes
    return result;
}
finally
{
    reader.Dispose();
}
}

private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Populate barcode types into the combobox
    PopulateBarcodeTypesCombobox();

    InitCamera();

    StartDecoding();
}

private void InitCamera()
{
    try

```

```

    {
        // Create Touchless lib manager to work with video camera
        _touchlessMgr = new TouchlessMgr();

        // Iterate through available video camera devices
        foreach (Camera camera in _touchlessMgr.Cameras)
        {
            // Add to list of available camera devices
            cbCamera.Items.Add(camera);
        }

        // Select first available camera
        cbCamera.SelectedItem = _touchlessMgr.Cameras[0];

        // Setting default image dimensions; see also camera selection event.
        _touchlessMgr.Cameras[0].CaptureWidth = int.Parse(tbCameraWidth.Text);
        _touchlessMgr.Cameras[0].CaptureHeight = int.Parse(tbCameraHeight.Text);

    }
    catch (Exception ex)
    {
        MessageBox.Show("No video camera available. Please connect camera.\n" -
    }
}

public void StartDecoding()
{
    UpdateCameraSelection();

    // Clear the text box output
    TextRange txt = new TextRange(textAreaBarcodes.Document.ContentStart, textAreaBarcodes.Document.ContentEnd);
    txt.Text = "";

    // Clean list of barcodes
    FoundBarcodes = null;

    // Check camera selected
    if (cbCamera.SelectedIndex != -1)
    {
        // Set status
        Status = true;

        // Update UI buttons
        btnStart.IsEnabled = false;
        btnStop.IsEnabled = true;
        cbBarcodeType.IsEnabled = false;
        cbCamera.IsEnabled = false;
        tbCameraHeight.IsEnabled = false;
        tbCameraWidth.IsEnabled = false;
        lblScanning.Content = "Scanning...";

        // Start the decoding thread
        _backgroundWorker.RunWorkerAsync(CloseOnFirstBarcodeFound);
    }
    else
    {
        MessageBox.Show("Please select camera");
    }
}
}

```

```

// Update picture box with the latest frame from video camera
void CurrentCamera_OnImageCaptured(object sender, CameraEventArgs e)
{
    // You can change image dimensions if needed
    //_touchlessMgr.CurrentCamera.CaptureWidth = 320;
    //_touchlessMgr.CurrentCamera.CaptureHeight = 240;
    Dispatcher.Invoke(DispatcherPriority.Normal, (SimpleDelegate) ()
    {
        if (_touchlessMgr != null)
        {
            pictureVideoPreview.BeginInit();
            BitmapImage imageSource = BitmapToImage(bitmap);

            ScaleTransform st = new ScaleTransform();
            st.ScaleX = (double)320 / (double)imageSource.Width;
            st.ScaleY = (double)240 / (double)imageSource.Height;
            TransformedBitmap tb = new TransformedBitmap(imageSource, st);

            pictureVideoPreview.Source = tb;
            pictureVideoPreview.EndInit();
            pictureVideoPreview.UpdateLayout();
        }
    });
}

// Convert System.Drawing.Bitmap to System.Windows.Media.Imaging.BitmapImage
BitmapImage BitmapToImageSource(Bitmap bitmap, ImageFormat imageFormat)
{
    using (MemoryStream memoryStream = new MemoryStream())
    {
        bitmap.Save(memoryStream, imageFormat);
        memoryStream.Position = 0;
        BitmapImage bitmapImage = new BitmapImage();
        bitmapImage.BeginInit();
        bitmapImage.StreamSource = memoryStream;
        bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
        bitmapImage.EndInit();

        return bitmapImage;
    }
}

private void btnStart_Click(object sender, RoutedEventArgs e)
{
    StartDecoding();
}

private void btnStop_Click(object sender, RoutedEventArgs e)
{
    StopDecoding();
}

private void StopDecoding()
{
    _backgroundWorker.CancelAsync();

    // Update UI elements
    lblScanning.Visibility = Visibility.Collapsed;
}

```

```

// Change working status
Status = false;

btnStart.IsEnabled = true;
btnStop.IsEnabled = false;

cbBarcodeType.IsEnabled = true;
cbCamera.IsEnabled = true;

tbCameraHeight.IsEnabled = true;
tbCameraWidth.IsEnabled = true;

        if (CloseOnFirstBarcodeFound && FoundBarcodes != null && FoundBarcodes.Count > 0)
            Close();
}

public void BackgroundWorker_DoWork(object sender, DoWorkEventArgs e)
{
    BackgroundWorker worker = (BackgroundWorker) sender;
    bool closeOnFirstBarcode = (bool) e.Argument;

    while (true)
    {
        // Work till user canceled the scan
        if (worker.CancellationPending)
        {
            e.Cancel = true;
            return;
        }

        // Get current frame bitmap from camera using Touchless
        Bitmap bitmap = _touchlessMgr.CurrentCamera.GetCurrentImage();

        // Search barcodes
        FoundBarcode[] result = null;

        if (bitmap != null)
            result = FindBarcodes(bitmap);

        // Check if we need to stop on first barcode found
        if (closeOnFirstBarcode && result != null && result.Length > 0)
        {
            e.Result = result;
            return; // end processing
        }

        // Wait a little to lower CPU load
        Thread.Sleep(ScanDelay);
    }
}

private void BackgroundWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    // Clear last results
    FoundBarcodes = null;

    if (e.Cancelled)
    {
        lblScanning.Content = "Canceled";
    }
}

```

```

    }
    else if (e.Error != null)
    {
        lblScanning.Content = "Error: " + e.Error.Message;
    }
    else
    {
        lblScanning.Content = "Done.";
        FoundBarcodes = (FoundBarcode[]) e.Result;
    }

    StopDecoding();
}

private void cbCamera_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    UpdateCameraSelection();
}

private void UpdateCameraSelection()
{
    if (cbCamera.Items.Count > 0 && cbCamera.SelectedIndex > -1)
    {
        if (_touchlessMgr.CurrentCamera != null)
            _touchlessMgr.CurrentCamera.OnImageCaptured -= CurrentCamera_OnImageCaptured;

        _touchlessMgr.CurrentCamera = null;

        Camera currentCamera = _touchlessMgr.Cameras[cbCamera.SelectedIndex];

        // Setting camera output image dimensions
        currentCamera.CaptureWidth = int.Parse(tbCameraWidth.Text);
        currentCamera.CaptureHeight = int.Parse(tbCameraHeight.Text);

        _touchlessMgr.CurrentCamera = currentCamera;
        currentCamera.OnImageCaptured += CurrentCamera_OnImageCaptured;
    }
}

// Updates barcode type filter according with combobox selection
private void UpdateBarcodeTypeToFindFromCombobox()
{
    string selectedItemText = cbBarCodeType.Text;

    if (string.IsNullOrEmpty(selectedItemText))
        throw new Exception("Empty barcode type selection.");

    _barcodeTypeToFind.Reset();

    // Iterate through BarcodeTypeSelector bool properties
    // and enable property by barcode name selected in the combobox
    foreach (PropertyInfo propertyInfo in typeof(BarcodeTypeSelector).GetProperties())
    {
        // Skip readonly properties
        if (!propertyInfo.CanWrite)
            continue;

        if (propertyInfo.Name == selectedItemText)
            propertyInfo.SetValue(_barcodeTypeToFind, true, null);
    }
}

```

```

}

protected void PopulateBarcodeTypesCombobox()
{
    cbBarcodeType.Items.Clear();
    List <string> items = new List<string>();

    foreach (PropertyInfo propinfo in typeof(BarcodeTypeSelector).
    {
        // Skip readonly properties
        if (!propinfo.CanWrite)
            continue;

        items.Add(propinfo.Name);
    }

    items.Sort();
    cbBarcodeType.ItemsSource = items;

    // Select first item in combobox (first is "Find All")
    cbBarcodeType.SelectedItem = cbBarcodeType.Items[0];
}

private void Window_Closing(object sender, CancelEventArgs e)
{
    Deinitialize();
}

private void Deinitialize()
{
    // cancel decoding thread
    _backgroundWorker.CancelAsync();

    // Deinit camera
    DeinitCamera();

    // Mark as closed
    IsClosed = true;
}

private void btnExit_Click(object sender, RoutedEventArgs e)
{
    Close();
}

private void DeinitCamera()
{
    if (_touchlessMgr != null)
    {
        _touchlessMgr.CurrentCamera.OnImageCaptured -= CurrentCamera_OnImageCap
        _touchlessMgr.CurrentCamera = null;
    }

    if (cbCamera.SelectedItem != null)
        cbCamera.SelectedItem = null;

    cbCamera.Items.Clear();
    _touchlessMgr = null;

    Thread.Sleep(500);
}

```

```

}

private void btnTryPopup_Click(object sender, RoutedEventArgs e)
{
    // Stop scan if any
    StopDecoding();

    // Deinit the current camera
    DeinitCamera();

    ShowScanPopup();

    // Reinit current camera
    InitCamera();
}

private void ShowScanPopup()
{
    // Create another MainWindow instance to scan barcodes
    MainWindow popup = new MainWindow();
    // Set new popup position shifted by 20 pixels
    popup.Left = Left + 20;
    popup.Top = Top + 20;

    // Set the new popup window to close on first found barcode
    popup.CloseOnFirstBarcodeFound = true;

    // Hide btnTryPopup button
    popup.btnTryPopup.Visibility = Visibility.Hidden;
    popup.btnStop.Visibility = Visibility.Hidden;
    popup.btnStart.Visibility = Visibility.Hidden;

    // Set the popup title
    popup.Title = "POPUP DIALOG - ONE-TIME SCAN";

    // Show the dialog
    popup.Show();

    // Now wait while the popup is closed (it will be closed on barcode found)
    while (!popup.IsClosed)
    {
        // HACK: Simulate "DoEvents"
        Dispatcher.Invoke(DispatcherPriority.Background, new ThreadStart(delegate
        {
            Thread.Sleep(20);
        }));
    }

    // Checking if one-time scan dialog found barcodes
    if (popup.FoundBarcodes != null && popup.FoundBarcodes.Length > 0)
        MessageBox.Show("Popup scan found the barcode: \n" + popup.FoundBarcode);
    else
        MessageBox.Show("Popup canceled. Returning to the main window");

    // Close the dialog
    popup.Close();
}
}
}

```

VIDEO

<https://www.youtube.com/watch?v=EARSPJFIJMU>

ON-PREMISE OFFLINE SDK

[60 Day Free Trial](#) or [Visit ByteScout BarCode Reader SDK Home Page](#)

[Explore ByteScout BarCode Reader SDK Documentation](#)

[Explore Samples](#)

[Sign Up for ByteScout BarCode Reader SDK Online Training](#)

ON-DEMAND REST WEB API

[Get Your API Key](#)

[Explore Web API Docs](#)

[Explore Web API Samples](#)

[visit www.ByteScout.com](http://www.ByteScout.com)

[visit www.PDF.co](http://www.PDF.co)

www.bytescout.com