Dynamic .NET TWAIN 8.1

Developer's Guide

December, 2019

Dynamsoft™

15+ Years' Experience in TWAIN SDKs and Version Control Solutions

Contents

Preface 5

Description 5

Audience 5

Getting Started 6

What Is Dynamic .NET TWAIN 6

Dynamic .NET TWAIN Modules 6

Basic Requirements 7

How to use TWAIN Scan Module? 8

What Is TWAIN 8

Step 1: Add Dynamsoft.Twain.dll to your project 8

Right click on Reference to add Reference 8

Open the local file system if the dll is not listed 9

Step 2: Start building a scanning application 10

Add the namespaces 10

Add the code for the button 10

Customize your scan settings 15

Related settings 16

Custom Capabilities 16

How to use the Camera Module? 18

Step 1: Add Dynamsoft.Camera.dll to your project 18

Right click on Reference to add Reference 18

Open the local file system if the dll is not listed 19

Step 2: Start building an acquisition application 20

Add the namespaces 20

Add code for the button 20

Customize your camera settings 20

How to use the PDF module? 22

Step 1: Add Dynamsoft.PDF.dll and Dynamsoft.ImageCore.dll to your project 22
Right click on References to add reference 22
Browse the local file system if the dll is not listed 22
Step 2: Add annotation to the PDF file and save it 23
Add the namespace and initialize the component like following: 23
Add lines, ellipses, rectangles and text onto an acquired image. 23
If you know where the annotation goes, it would be good to specify the location, color, and other properties. 24
Save the Annotation with the PDF 24
Step 3: Load a non-image PDF to buffer 27
Add the namespace and initialize the component like following: 27
Get the convert result 27
How to use the Barcode Generator? 28
Step 1: Add Dynamsoft.BarcodeGenerator.dll to your project 28
Right click on References to add reference 28
Open the local file system if the dll is not listed 29
Step 2: Add a barcode to the image 29
Add namespace and initialize the component like following: 29
Add the barcode to the existing image using the following line: 30
How to use OCR? 31
Step 1: Add Dynamsoft.OCR.dll to your project 31
Right click on References to add reference 31
Open the local file system if the dll is not listed 32
Step 2: Start executing OCR 32
Add namespace and initialize the component like following: 32
Load local image(s) into the buffer and show it in the viewer 34
Preparation 34
Step 1: Add Dynamsoft.Froms.Viewer.dll to your project 34
Step 2: Control the buffer 36
Manipulate image(s) 39

Related methods 41

Save and Upload Images 43

Save the image(s) to a local system 43

Save a single image 43

Save selected image(s) as TIFF 43

Save the image(s) to a local database 43

Upload the image(s) to a web server 44

Upload image(s) with meta data 45

Download image(s) 45

Useful Resources 47

Sample Download 47

Knowledge Base 47

FAQ 47

All APIs (Property, Method, Capability, and Event) 47

Contact Us 47

Preface

Description

This guide provides instructions on how to use Dynamsoft's Dynamic .NET TWAIN SDK. It provides a general overview of most of the things you can achieve with the SDK.

Audience

This guide is meant for both experienced Dynamic .NET TWAIN SDK developers and those who are still new to our SDK.

For developers who are new to the SDK, this guide will get you started quickly and help you develop your own scanning module within a short time.

For those who have used the SDK before, this guide provides information on advanced APIs which can help polish your scanning module.

This guide is updated with every new release. You can always find information about the new features made available in that new version.

Getting Started

What Is Dynamic .NET TWAIN

Dynamic .NET TWAIN is a document imaging SDK designed by the standards of the TWAIN Protocol. It's built based on Microsoft .NET Framework 2.0/4.0.

The library is optimized for use in C# and VB.NET desktop apps. It allows you to **acquire images** from any TWAIN compliant or DirectShow-enabled devices (scanner, webcams, and capture cards etc.), **edit** the images, and then **save** them to your local machine or **upload** them to an FTP or HTTP server.

The library also supports loading/downloading image formats like BMP, JPG, PNG, TIF, and PDF.

Dynamic .NET TWAIN Modules

Module Name	Description	License
TwainScan	an image acquisition library which gets image data from TWAIN compatible devices.	TWAIN Scan Module
DirectShowCamera	an image acquisition library which gets image data from USB Video Class (UVC) compatible webcams on Windows.	Webcam Module
Barcode Generator 1D & 2D	a barcode generator library which generates multi-format barcode symbols.	Barcode Generator
PDF	a PDF library which rasterizes any PDF document into images，marks up PDF documents by drawing objects such as underlining, circles, notes/comments and saves or uploads the images as PDF document, etc.	PDF Library
OCR	a Tesseract-based OCR library which processes images containing information in different languages and returns searchable PDF files or text.	OCR SDK
ImageCore	an image processing library which includes image I/O and image editing features.	For free

ImageViewer	an image viewer library which displays image including PDF with annotation.	For free

Basic Requirements

Operating System: Microsoft Windows Operating Systems

Microsoft .NET Framework 2.0 or above.

Scanner/Camera: TWAIN compliant.

Webcam: DirectShow Enabled.

How to use TWAIN Scan Module?

What Is TWAIN

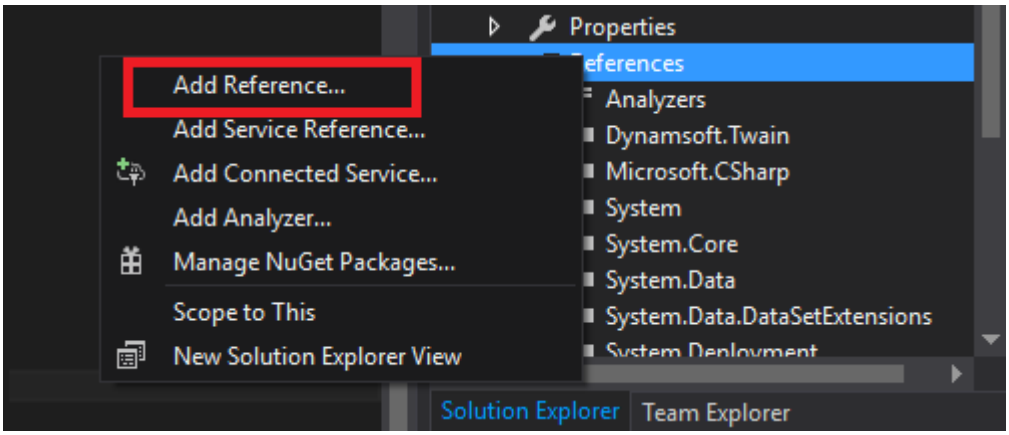
TWAIN is an applications programming interface and communications protocol that regulates communication between software and digital imaging devices, such as image scanners and digital cameras.

Today the TWAIN standard, including the specification, data source manager, and sample code, are maintained by the not-for-profit organization **TWAIN Working Group**.

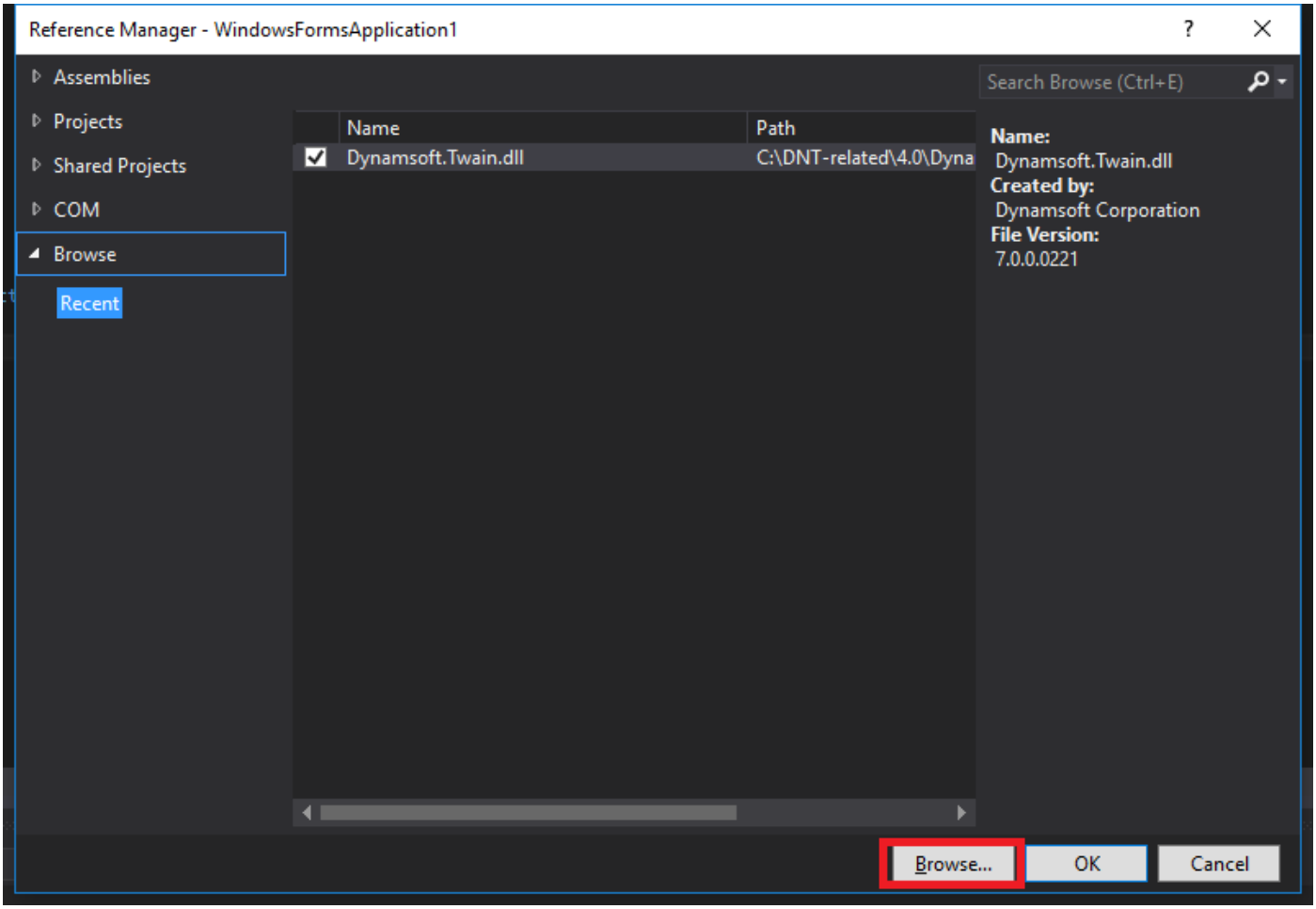
Dynamsoft Corporation is a member of the TWAIN Working Group.

Step 1: Add Dynamsoft.Twain.dll to your project

Right click on Reference to add Reference



Open the local file system if the dll is not listed



Once you install Dynamic .Net TWAIN, you can find the DLLs in the below directories:

{installation directory}\Redistributable\Assembly\For .NETFramework 4.0 ***Dynamsoft.TWAIN.dll***

{installation directory}\Redistributable\Assembly\For .NETFramework 2.0 ***Dynamsoft.TWAIN.dll***

Step 2: Start building a scanning application

Add the namespaces

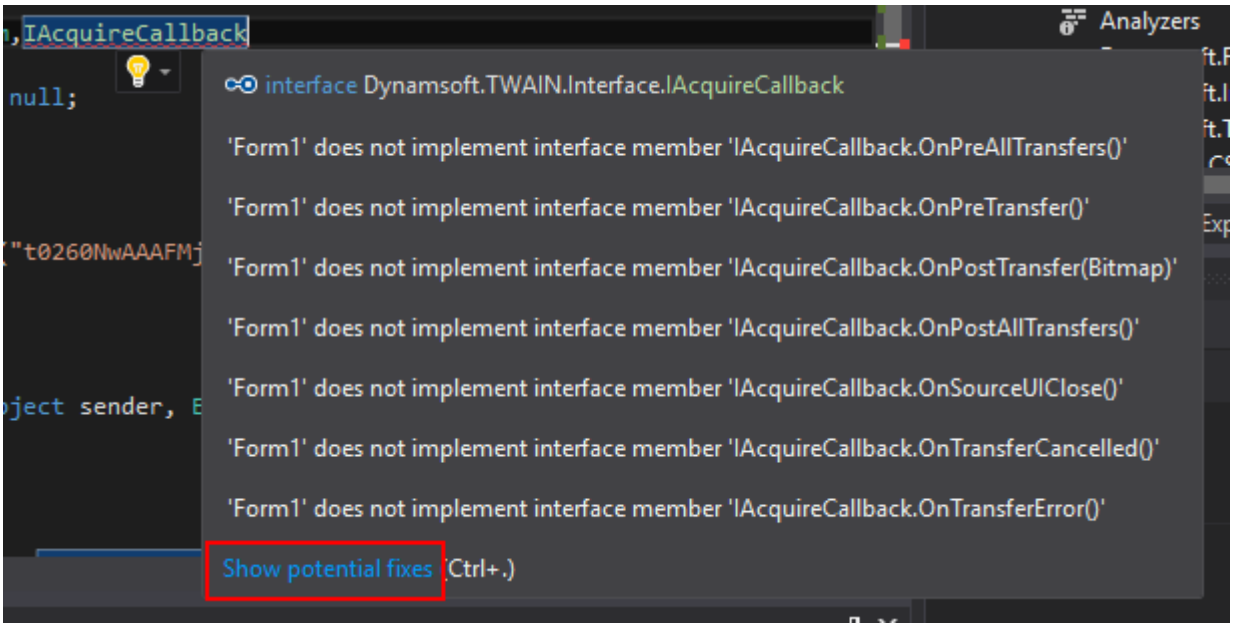
```
using Dynamsoft.TWAIN;
using Dynamsoft;
using Dynamsoft.TWAIN.Common;
using Dynamsoft.TWAIN.Enums;
using Dynamsoft.TWAIN.Interface;
```

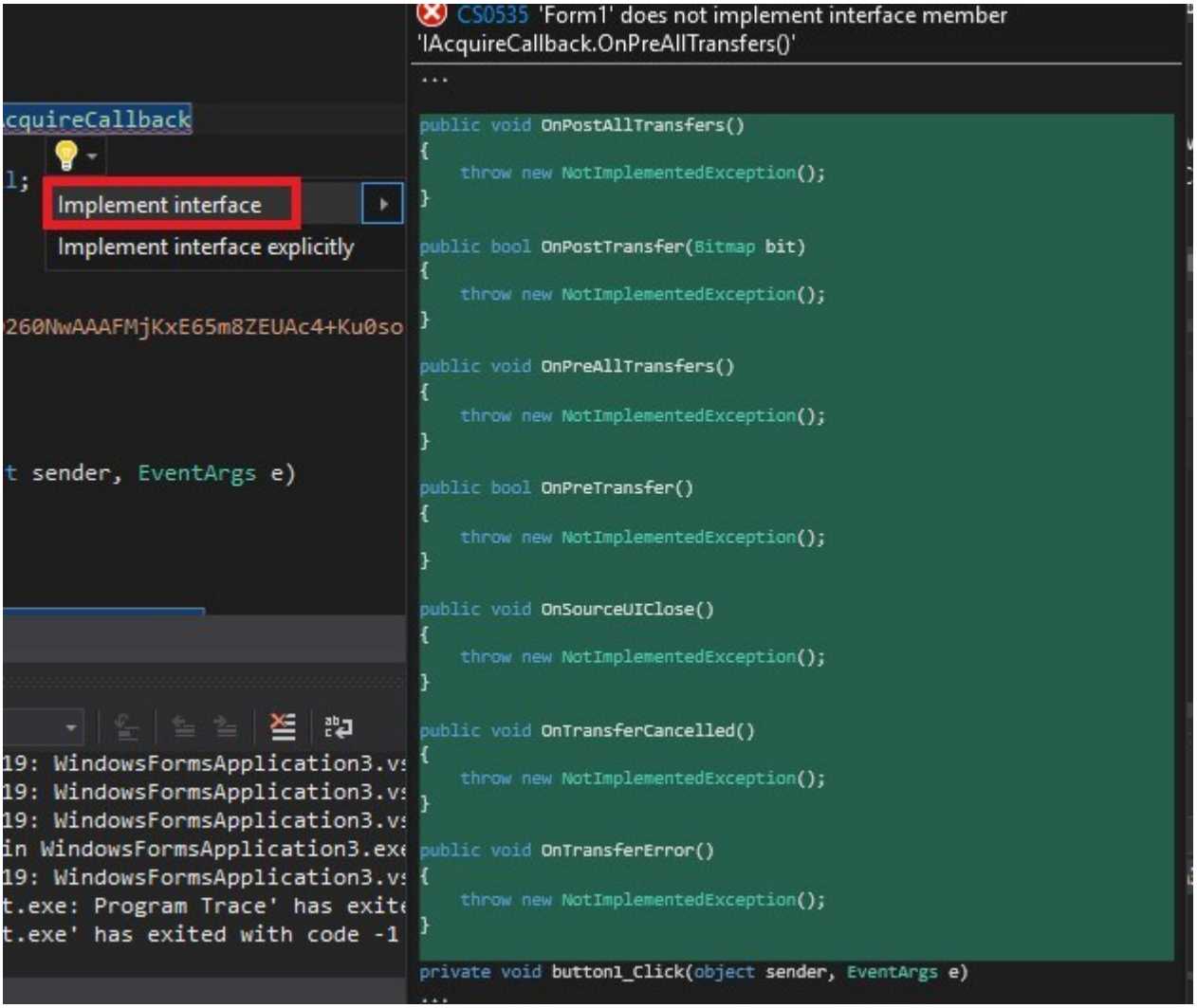
Add the code for the button

```
namespace WindowsFormsApplication3
{
    public partial class Form1 : Form,IAcquireCallback
    {
        private TwainManager twain = null;
        public Form1()
        {
            InitializeComponent();
            twain = new TwainManager("<your license here>");
         }

         private void button1_Click(object sender, EventArgs e)
         {
             twain.SelectSource();
             twain.CloseSource();
             twain.OpenSource();
             twain.AcquireImage(this as IAcquireCallback);
        }
    }
}
```

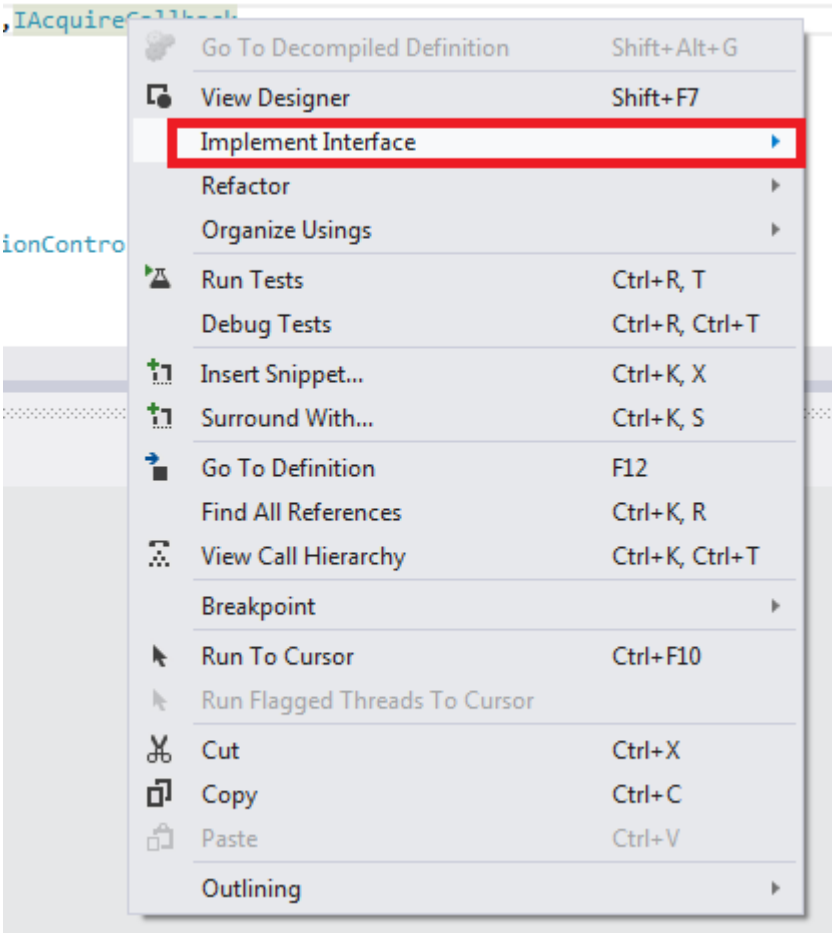
1. Implement interface for IAcquireCallback

   For VS2015:


IAcquireCallback
Analyzers
interface Dynamsoft.TWAIN.Interface.IAcquireCallback
'Form1' does not implement interface member 'IAcquireCallback.OnPreAllTransfers()'
'Form1' does not implement interface member 'IAcquireCallback.OnPreTransfer()'
'Form1' does not implement interface member 'IAcquireCallback.OnPostTransfer(Bitmap)'
'Form1' does not implement interface member 'IAcquireCallback.OnPostAllTransfers()'
'Form1' does not implement interface member 'IAcquireCallback.OnSourceUIClose()'
'Form1' does not implement interface member 'IAcquireCallback.OnTransferCancelled()'
'Form1' does not implement interface member 'IAcquireCallback.OnTransferError()'
Show potential fixes (Ctrl+.)


CS0535 'Form1' does not implement interface member 'IAcquireCallback.OnPreAllTransfers()'
Implement interface
Implement interface explicitly
public void OnPostAllTransfers()
{
throw new NotImplementedException();
}
public bool OnPostTransfer(Bitmap bit)
{
throw new NotImplementedException();
}
public void OnPreAllTransfers()
{
throw new NotImplementedException();
}
public bool OnPreTransfer()
{
throw new NotImplementedException();
}
public void OnSourceUIClose()
{
throw new NotImplementedException();
}
public void OnTransferCancelled()
{
throw new NotImplementedException();
}
public void OnTransferError()
{
throw new NotImplementedException();
}
private void button1_Click(object sender, EventArgs e)

For VS 2012:



2. Change the functions accordingly

```
public void OnPostAllTransfers()
{
}
/*needs to return something in this event*/
public bool OnPostTransfer(Bitmap bit)
{
    bit.Save ("<the path for saving image>");
    return true;
}
public void OnPreAllTransfers()
{
}
/*needs to return something in this event*/
public bool OnPreTransfer()
```

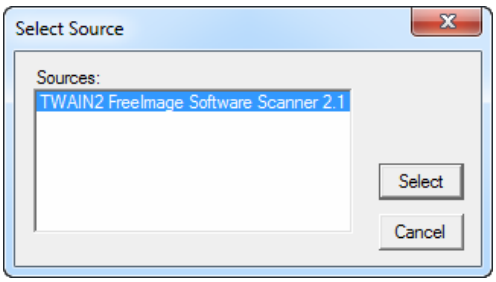
```
{
    return true;
}
public void OnSourceUIClose()
{
}
public void OnTransferCancelled()
{
}
public void OnTransferError()
{
}
```

The whole application will be as following:

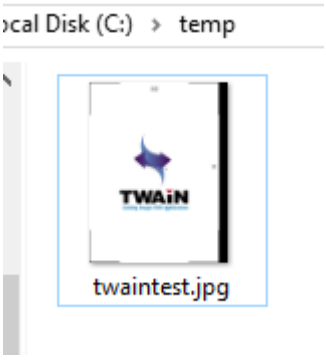
```
namespace WindowsFormsApplication3
{
       public partial class Form1 : Form,IAcquireCallback
      {
            private TwainManager twain = null;
           public Form1()
           {
                InitializeComponent();
                twain = new TwainManager("<your license here>");
           }

           public void OnPostAllTransfers()
          {
          }
          /*needs to return something in this event*/
           public bool OnPostTransfer(Bitmap bit)
           {
               bit.Save(@"c:\\temp\\twaintest.jpg");
                return true;
          }
          public void OnPreAllTransfers()
          {
          }
          /*needs to return something in this event*/
           public bool OnPreTransfer()
          {
             return true;
          }
          public void OnSourceUIClose()
```

```
            {
            }
            public void OnTransferCancelled()
             {
             }
             public void OnTransferError()
            {
            }
            private void button1_Click(object sender, EventArgs e)
            {
               twain.SelectSource();
                  twain.CloseSource();
                  twain.OpenSource();
                 twain.AcquireImage(this as IAcquireCallback);
            }
        }
    }
```

Then the application is ready to run. On the form, click the "Scan" button. A window, as shown below, will ask which scanner you would like to use. If no scanner is available, please make sure you have the scanner driver installed on your machine and the scanner is running and connected to the machine. If you do not have a scanner at hand, you can download the virtual scanner for testing.

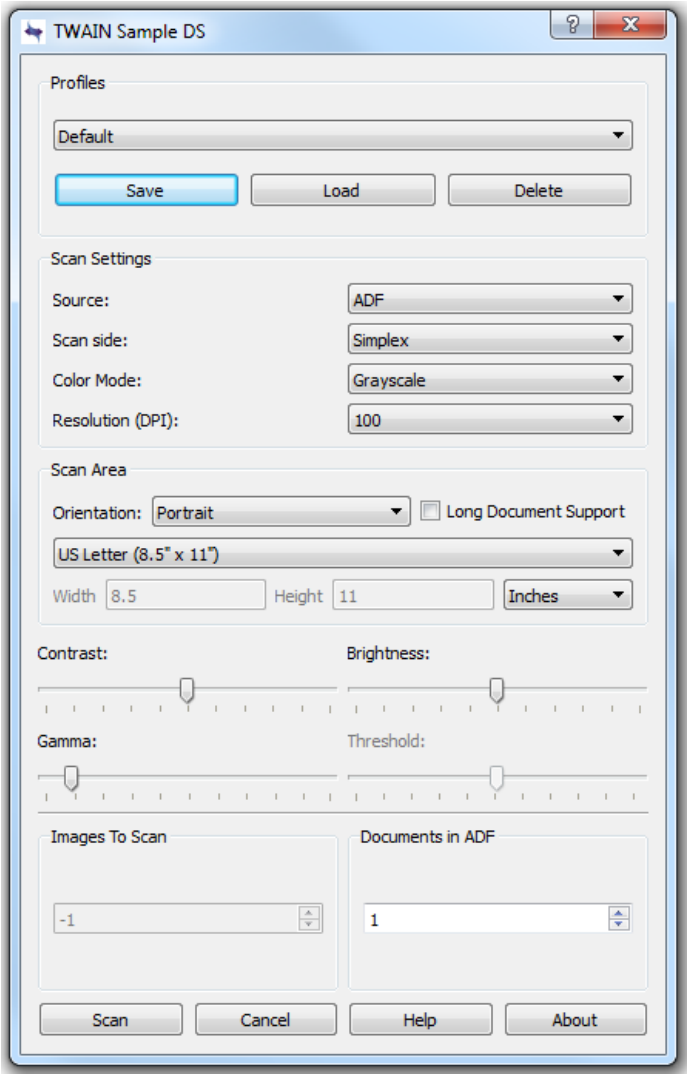


In our case, we simply select "TWAIN2 Free Image Software Scanner 2.1" for the test and you can see an image is scanned in. Good Job!



Customize your scan settings

Before you start an actual scan session, you can configure the way your documents to be scanned. Normally, you can do all of this in the User Interface provided by the device vendor, for example:



However, these settings might be too overwhelming for end users, especially for those without a technical background. With our dll, you can configure all these settings in your code instead. For example:

```
twain.SelectSource();
twain.CloseSource();
twain.OpenSource();      //Open the source before doing any configuration
twain.IfShowUI = false; //Hide the User Interface of the scanner
twain.IfFeederEnabled = true; //Use the document feeder for batch scan
twain.IfDuplexEnabled = false; //Scan in Simplex mode
twain.PixelType = TWICapPixelType.TWPT_GRAY; //Scan pages in Grey
twain.Resolution = 200; //Scan pages in 200 DPI
```

```
twain.AcquireImage(this as IAcquireCallback);//Start scanning
```

Related settings

BitDepth	Returns or sets the pixel bit depth for the current value of PixelType property.
Brightness	Returns or sets the brightness value available within the Source.
Contrast	Returns or sets the contrast values available within the Source.
Duplex	Returns whether the source supports duplex.
IfDuplexEnabled	Returns or sets whether the Source supports duplex.
IfFeederEnabled	Returns or sets whether the Automatic Document Feeder (ADF) is enabled.
IfFeederLoaded	Returns whether or not there are documents loaded in the Source's feeder.
IfPaperDetectable	Returns the value whether the Source has a paper sensor.
JPEGQuality	Returns or sets the quality of the JPEG file.
PageSize	Returns or sets the page size(s) the Source can/should use to acquire image data.
PixelFlavor	Returns or sets the pixel flavor for acquired images.
PixelType	Returns or sets the pixel type of acquired images.
Resolution	Returns or sets the current resolution for image acquisition.
ScanInNewThread	Acquire image in a new separate thread.
TransferMode	Returns or sets the transfer mode.
XferCount	Returns and sets the number of images you are willing to transfer per session.

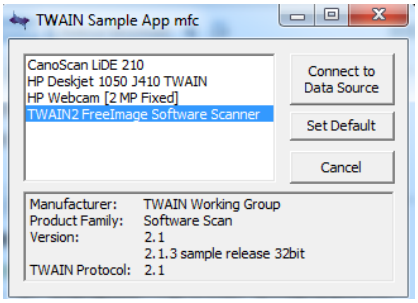
Custom Capabilities

TWAIN devices such as scanners provide Capabilities, which can be used by TWAIN applications like Dynamic .NET TWAIN.

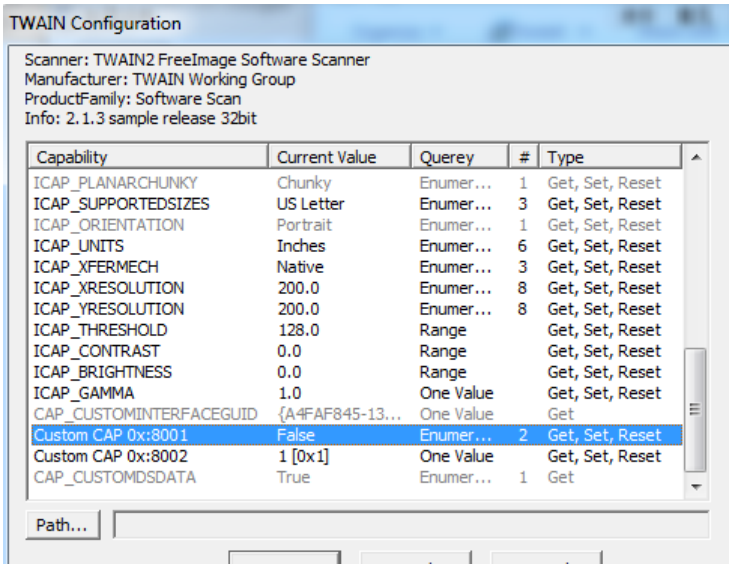
The TWAIN specification defines many standard capabilities and new capabilities are being added gradually. However, there are still some capabilities which are not standardized and are only supported by certain models of devices. These capabilities are called “Custom Capabilities”. Since they are not standard, we need to perform “Capability Negotiation” to make sure of them.

Below are the detailed steps:

1. Install the TWAIN sample application (32-bit, 64-bit).
2. Open the target device's source with the TWAIN sample application.



3. In the window "TWAIN Configuration", find the custom capability you want to use. Since there won't be a recognizable name, you need to consult the device manual or its vendor for more details.



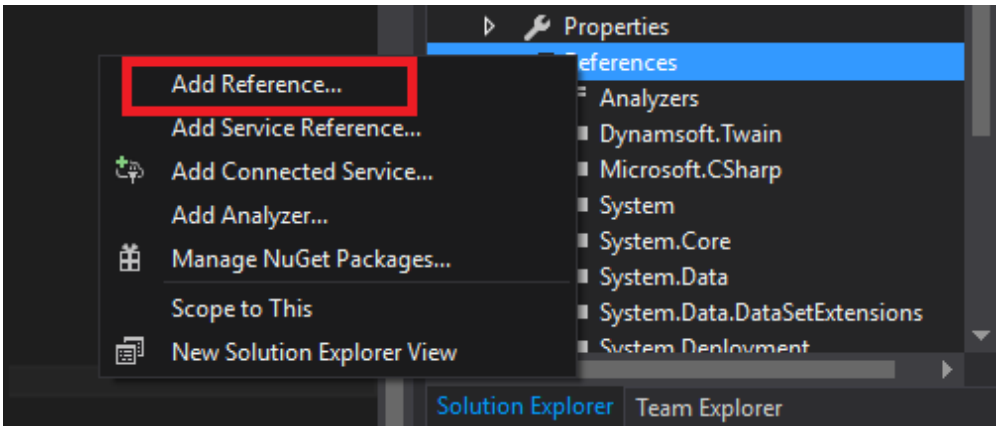
4. From the above window, we know the hexadecimal value for this capability (0x8001 as an example) as well as its type (Enumeration) and available operations (Get, Set, Reset), we can use it like this,

```
twain.SelectSource();
twain.CloseSource();
twain.OpenSource();      //Open the source before doing any configuration
twain.Capability = (TWCapability)0x8001;
bool bRet = twain.CapGet();
double db = twain.CapValue; //Get the current value of the capability

twain.Capability = (TWCapability)0x8001;
twain.CapValue = 1;
twain.CapType = TWCapType.TWON_ONEVALUE;
bRet = twain.CapSet();//Set the value to be 1
```

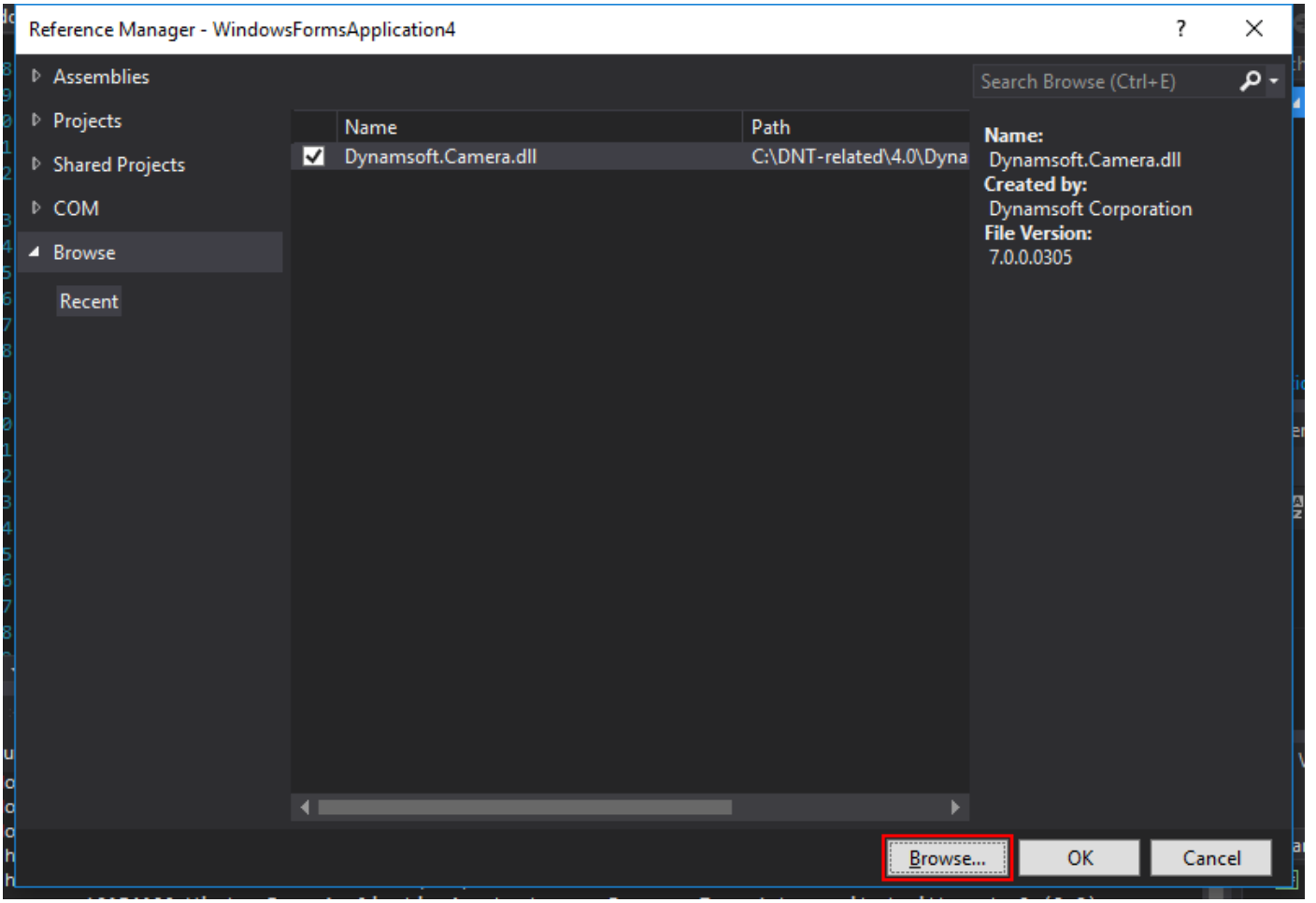
How to use the Camera Module?

Step 1: Add Dynamsoft.Camera.dll to your project

Right click on Reference to add Reference



Open the local file system if the dll is not listed



Once you install Dynamic .Net TWAIN, you can find the DLLs in the below directories:

{installation directory}\Redistributable\Assembly\For .NETFramework 4.0 ***Dynamsoft.Camera.dll***

{installation directory}\Redistributable\Assembly\For .NETFramework 2.0 ***Dynamsoft.Camera.dll***

Step 2: Start building an acquisition application

Add the namespaces

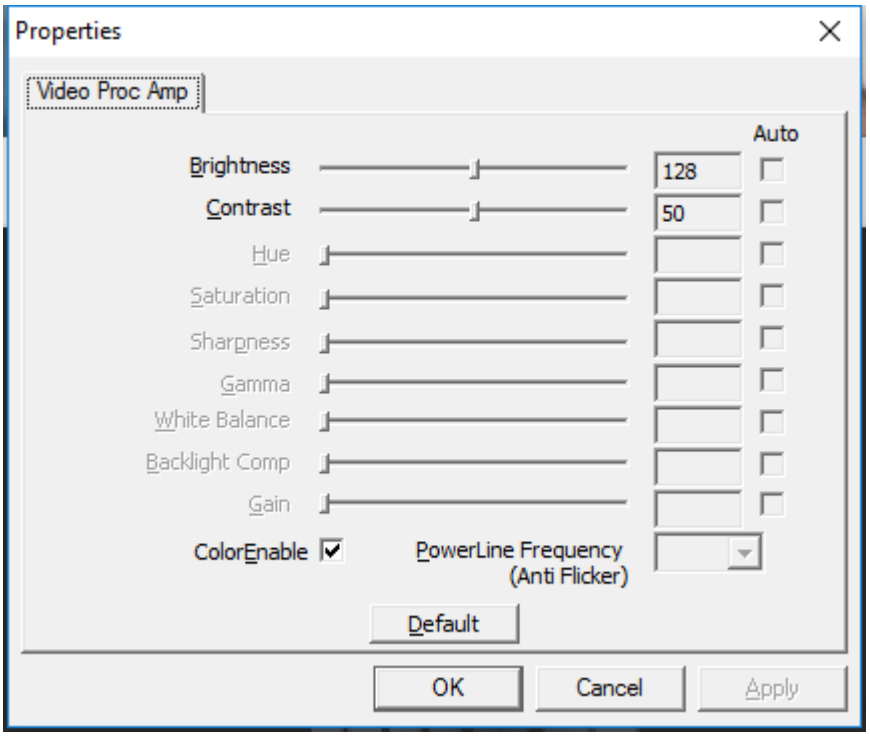
```
using Dynamsoft;
using Dynamsoft.UVC;
using Dynamsoft.UVC.Common;
using Dynamsoft.UVC.Delegate;
using Dynamsoft.UVC.Enums;
```

Add code for the button

```
namespace WindowsFormsApplication3
{
    public partial class Form1 : Form
    {
        private Camera camera = null;
        private CameraManager cameramanager = null;
        public Form1()
        {
            InitializeComponent();
            cameramanager = new CameraManager ("<your license here>");
         }

         private void button1_Click(object sender, EventArgs e)
         {
            camera=cameramanager.SelectCamera(<Your camera index>);
            camera.Open();
            Bitmap grabbedimage=camera.GrabImage();
            grabbedimage.Save("c://temp//cameratestimage.jpg");
            }
    }
}
```

Customize your camera settings

Before you start an actual grab session, you can configure the way your images are to be grabbed. Normally, you can do all of this in the User Interface provided by the device vendor, for example:



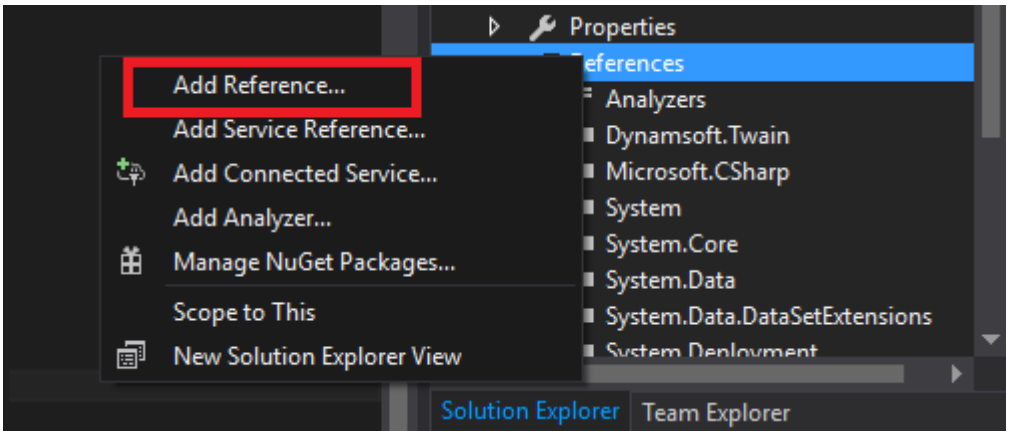
However, these settings might be too overwhelming for end users, especially for those without a technical background. With our dll, you can configure all these settings in your code instead. For example:

```
camera=cameramanager.SelectCamera(<Your camera index>); //The camera index starts with 0
camera.Close();
camera.Open(); //O pen the source before doing any configuration
MessageBox.Show((camera.Contrast.MaxValue).ToString());//Show the maximum value for contrast
MessageBox.Show((camera.Sharpness.MinValue).ToString());//Show the minimum value for sharpness
camera.Zoom.Value =2; //Set the zoom value for the camera
camera.WhiteBalance.IfAuto = true; //Set automatically change the white balance for the camera
Bitmap grabbedimage=camera.GrabImage(); //Start Grabbing image
```

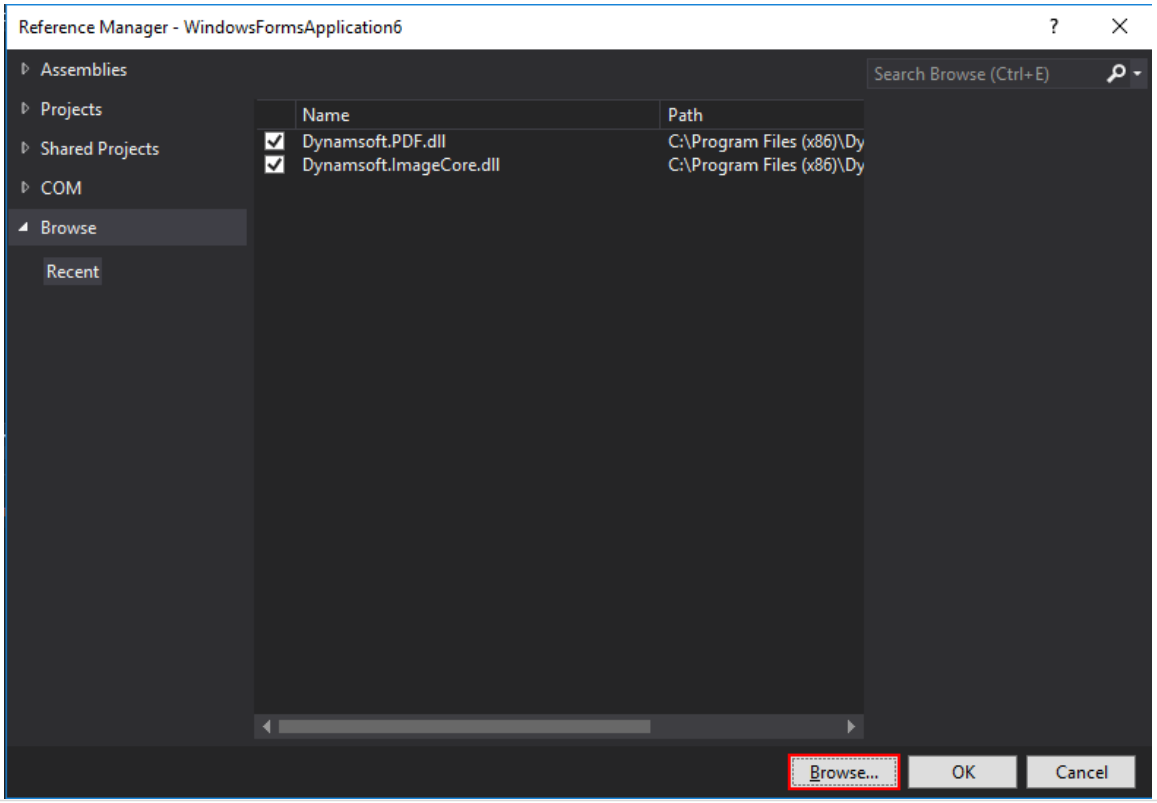
How to use the PDF module?

Step 1: Add Dynamsoft.PDF.dll and Dynamsoft.ImageCore.dll to your project

Right click on References to add reference



Browse the local file system if the dll is not listed



Once you install Dynamic .Net TWAIN, you can find the DLLs in the below directories:

{installation directory}\Redistributable\Assembly\For .NETFramework 4.0 ***Dynamsoft.PDF.dll***

{installation directory}\Redistributable\Assembly\For .NETFramework 2.0 ***Dynamsoft.PDF.dll***

Step 2: Add annotation to the PDF file and save it

Add the namespace and initialize the component like following:

```
using Dynamsoft.PDF;
using Dynamsoft.PDF.Annotation;
using Dynamsoft.PDF.Enums;
using Dynamsoft.Core;
public partial class Form1 : Form
{
    private ImageCore imagecore = null;
    private PDFCreator creator = null;
    public Form1()
    {
        InitializeComponent();
        imagecore = new ImageCore();
        creator = new PDFCreator("<Your license here>");
    }

}
```

Add lines, ellipses, rectangles and text onto an acquired image.

For example, to add a line, you can use the code below:

```
//Draw a line
AnnotationData tempLineAnnotation = new AnnotationData();
tempLineAnnotation.AnnotationType = AnnotationType.enumLine;
tempLineAnnotation.PenColor = Color.Black.ToArgb();
```

Very easy-to-use, isn't it? You will find further samples below for ellipse, rectangle and text, respectively.

```
//Draw an ellipse
AnnotationData tempEllipseAnnotation = new AnnotationData();
tempEllipseAnnotation.AnnotationType = AnnotationType.enumEllipse;
tempEllipseAnnotation.FillColor = Color.Blue.ToArgb();
tempEllipseAnnotation.PenColor = Color.Black.ToArgb();
```

```
//Draw a rectangle
AnnotationData tempRectAnnotation = new AnnotationData();
tempRectAnnotation.AnnotationType = AnnotationType.enumRectangle;
tempRectAnnotation.FillColor = Color.Green.ToArgb();
tempRectAnnotation.PenColor = Color.Black.ToArgb();
//Add some text
AnnotationData tempTextAnnotation = new AnnotationData();
tempTextAnnotation.AnnotationType = AnnotationType.enumText;
tempTextAnnotation.FontType = new AnnoTextFont();
tempTextAnnotation.FontType.Name = "Microsoft Sans Serif";
tempTextAnnotation.FontType.Size = 30;
tempTextAnnotation.FontType.TextColor = Color.Black.ToArgb();
```

If you know where the annotation goes, it would be good to specify the location, color, and other properties. For example, we can add a rectangle by only clicking on a single button, and all information is written in advance. Here is the code:

```
AnnotationData tempRectAnnotation = new AnnotationData();
tempRectAnnotation.AnnotationType = AnnotationType.enumRectangle;
tempRectAnnotation.StartPoint = new Point(400, 400);
tempRectAnnotation.EndPoint= new Point(490, 520);
tempRectAnnotation.FillColor = Color.Green.ToArgb();
tempRectAnnotation.PenColor = Color.Black.ToArgb();
tempRectAnnotation.PenWidth = 2;
tempRectAnnotation.Description = "Create a rectangle annotation.";
AddAnnotation(tempRectAnnotation);
```

After an annotation has been added, it can be re-shaped, re-located and edited.

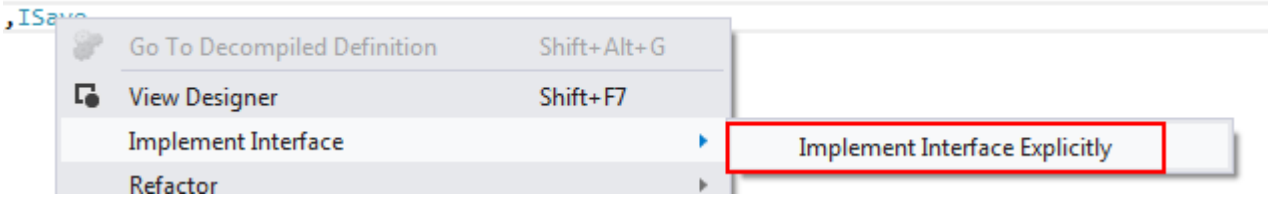
Save the Annotation with the PDF

To use the save function, the ISave interface is needed. Please add it and implement it like the following:

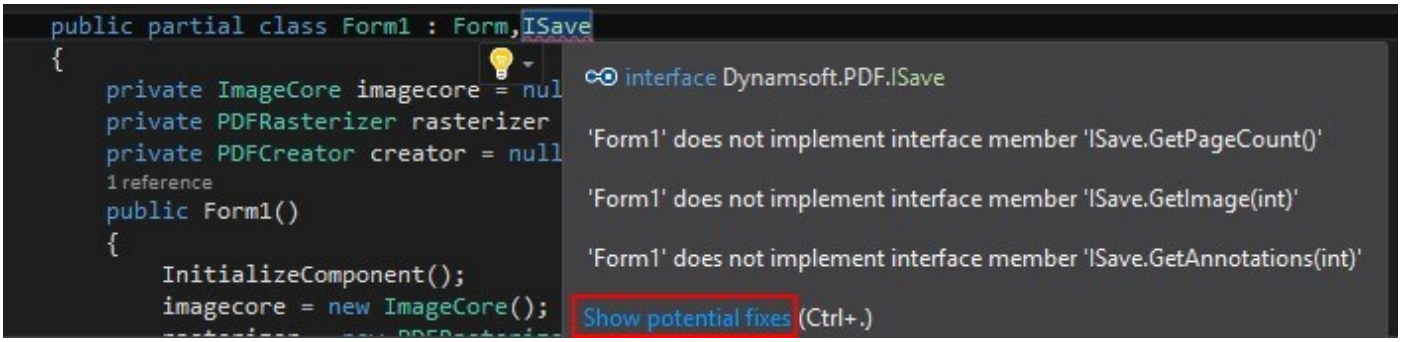
```
using Dynamsoft.PDF;
using Dynamsoft.PDF.Annotation;
using Dynamsoft.PDF.Enums;
using Dynamsoft.Core;
public partial class Form1 : Form, ISave
{
    private ImageCore imagecore = null;
    private PDFCreator creator = null;
    public Form1()
    {
        InitializeComponent();
```

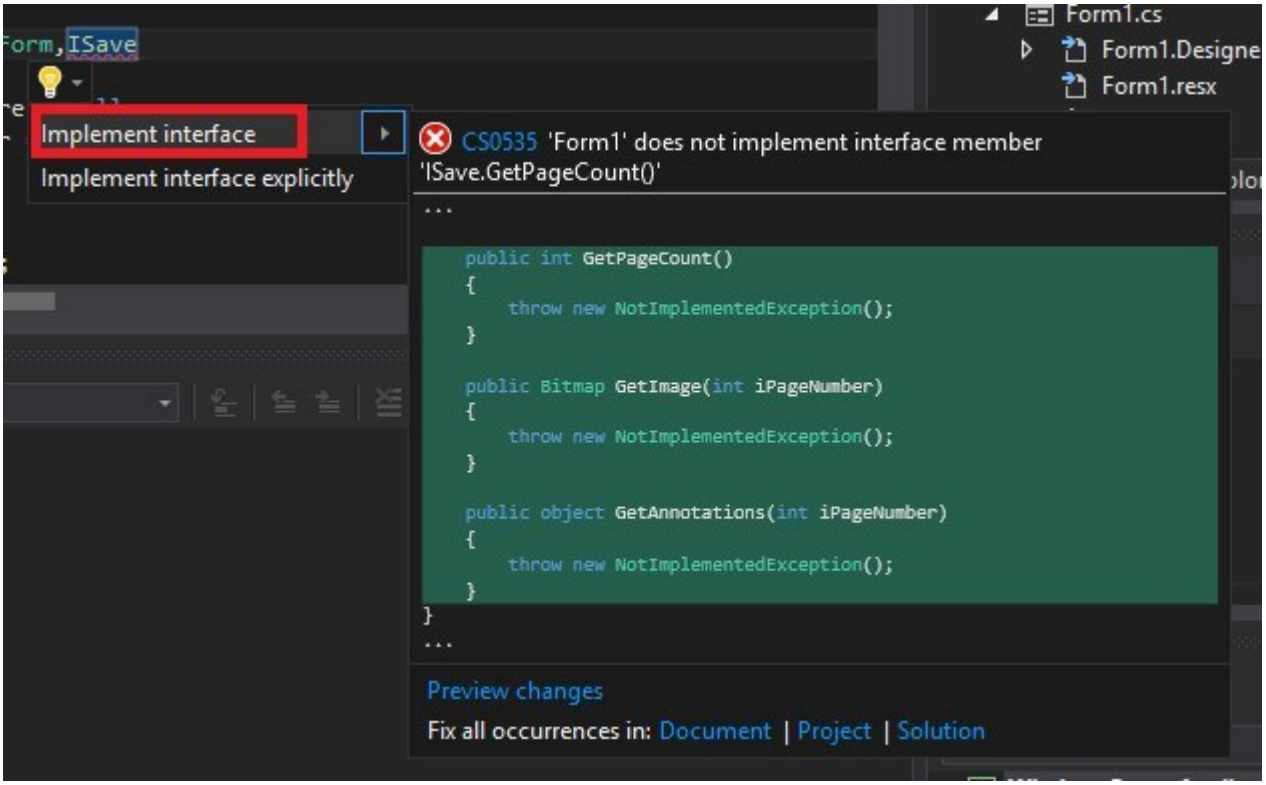
```
            imagecore = new ImageCore();
            creator = new PDFCreator("<Your license here>");
        }

    }
```

For VS2012:



For VS 2015:





Please change the implementations accordingly. Below is an example:

```
public int GetPageCount()
{
    return imagecore.ImageBuffer.HowManyImagesInBuffer;
}

public Bitmap GetImage(int iPageNumber)
{
    return imagecore.ImageBuffer.GetBitmap((short)iPageNumber);
}

public object GetAnnotations(int iPageNumber)
{
    return imagecore.ImageBuffer.GetMetaData((short)iPageNumber,
EnumMetaDataType.enumAnnotation);
}
```

Below is the function for saving:

```
creator.Save(this as ISave, "<the path for saving image>");
```

Please explore the PDF Annotation Sample in the installation directory to learn more.

{installation directory}\Samples\C# Samples\AnnotationSample

Step 3: Load a non-image PDF to buffer

Add the namespace and initialize the component like following:

```
using Dynamsoft.PDF;

public partial class Form1 : Form ,IConvertCallback
{
    private PDFRasterizer rasterizer = null;
    public Form1()
    {
        InitializeComponent();
        rasterizer = new PDFRasterizer("<your license here>");
    }

}
```

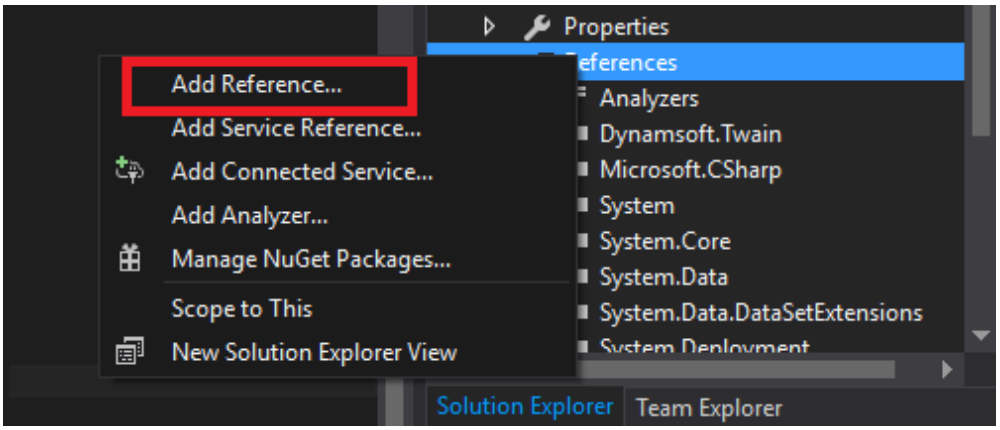
Get the convert result

```
private void button1_Click(object sender, EventArgs e)
{
    //Set up the add-on
    rasterizer.ConvertMode = Dynamsoft.PDF.Enums.EnumConvertMode.enumCM_RENDERALL;
    //Invoke the methods to get the result image
    rasterizer.ConvertToImage("<Your file name here>", "", 300, this as IConvertCallback);
}
//The result will be stored in the callback function
public void LoadConvertResult(ConvertResult result)
{
    result.Image.Save("<the path for saving image>");
}
```

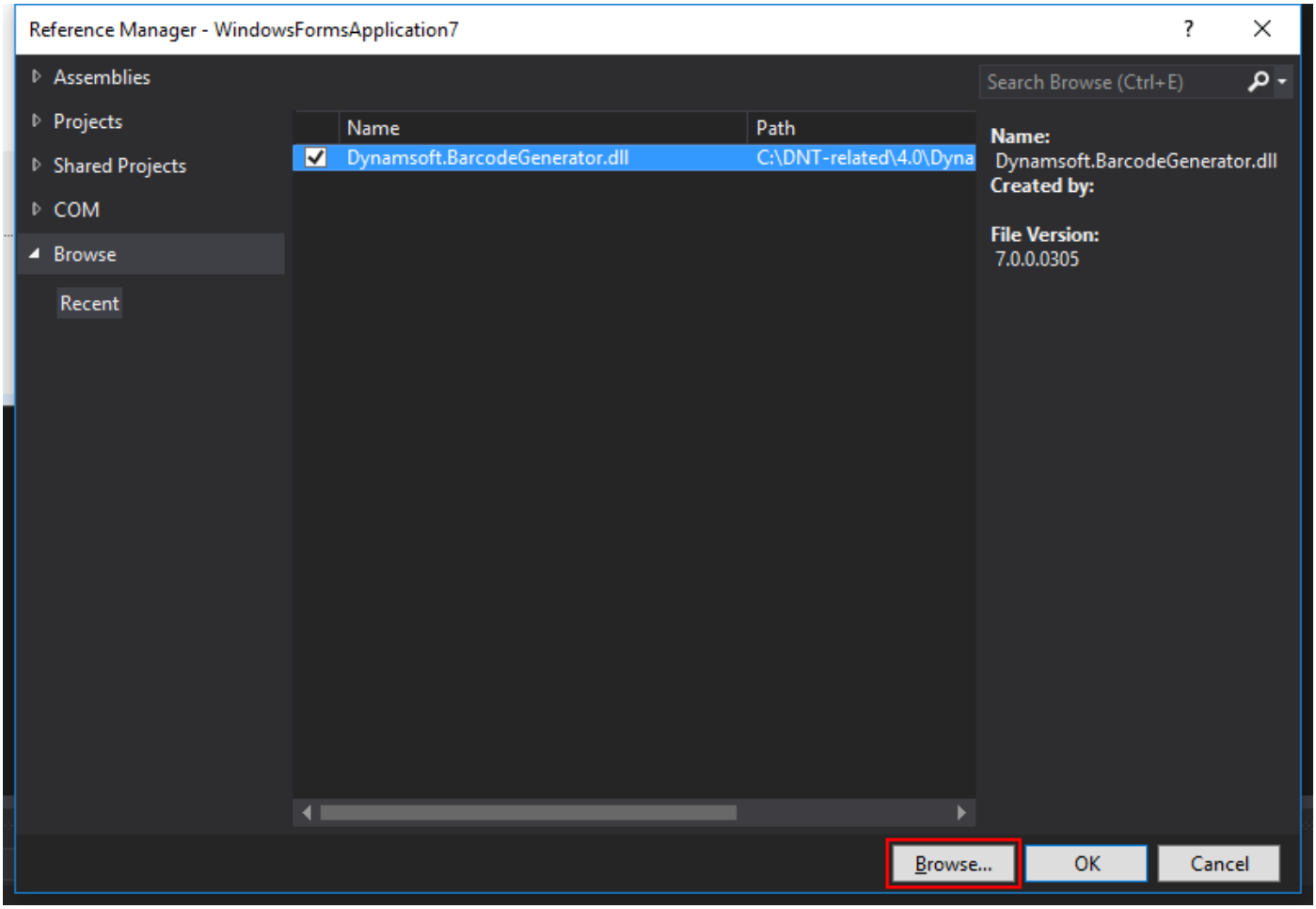
How to use the Barcode Generator?

Step 1: Add Dynamsoft.BarcodeGenerator.dll to your project

Right click on References to add reference



Open the local file system if the dll is not listed



Once you install Dynamic .Net TWAIN, you can find the DLLs in the below directories:

{installation directory}\Redistributable\Assembly\For .NETFramework 4.0 ***Dynamsoft.BarcodeGenerator.dll***

{installation directory}\Redistributable\Assembly\For .NETFramework 2.0 ***Dynamsoft.BarcodeGenerator.dll***

Step 2: Add a barcode to the image

Add namespace and initialize the component like following:

```
using Dynamsoft.Barcode.Generator;

public partial class Form1 : Form
{
```

```
        private BarcodeGenerator generator = null;
        public Form1()
        {
            InitializeComponent();
            generator = new BarcodeGenerator("<your license here>");
        }
    }
```

Add the barcode to the existing image using the following line:

```
generator.AddBarcode(imagebitmap,     Dynamsoft.Barcode.Enums.EnumBarcodeFormat.QR_CODE,
"txtBarcodeContent","txtHumanReadableTxt", BarcodeLocationX, BarocdeLocationY, BarcodeScale);
```

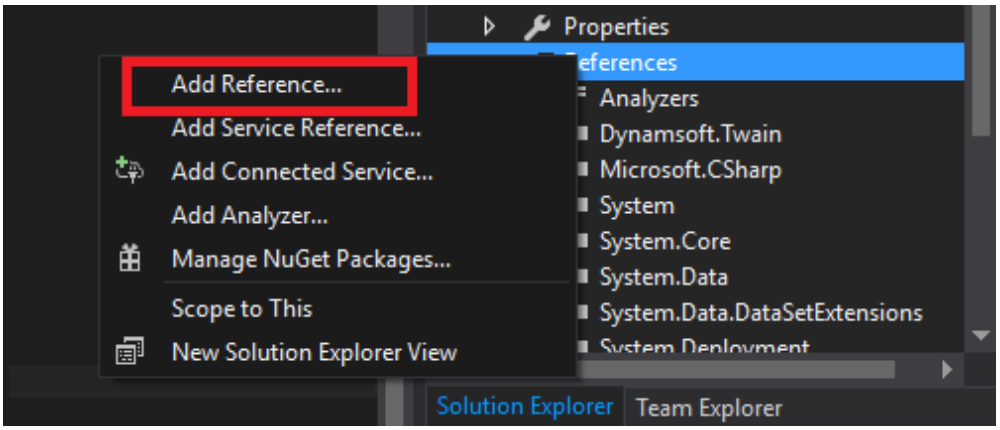
The barcode content is the content to be put into the barcode, and the human readable text is the text to be displayed.

We can also change the size of the barcode image using the parameter BarcodeScale.

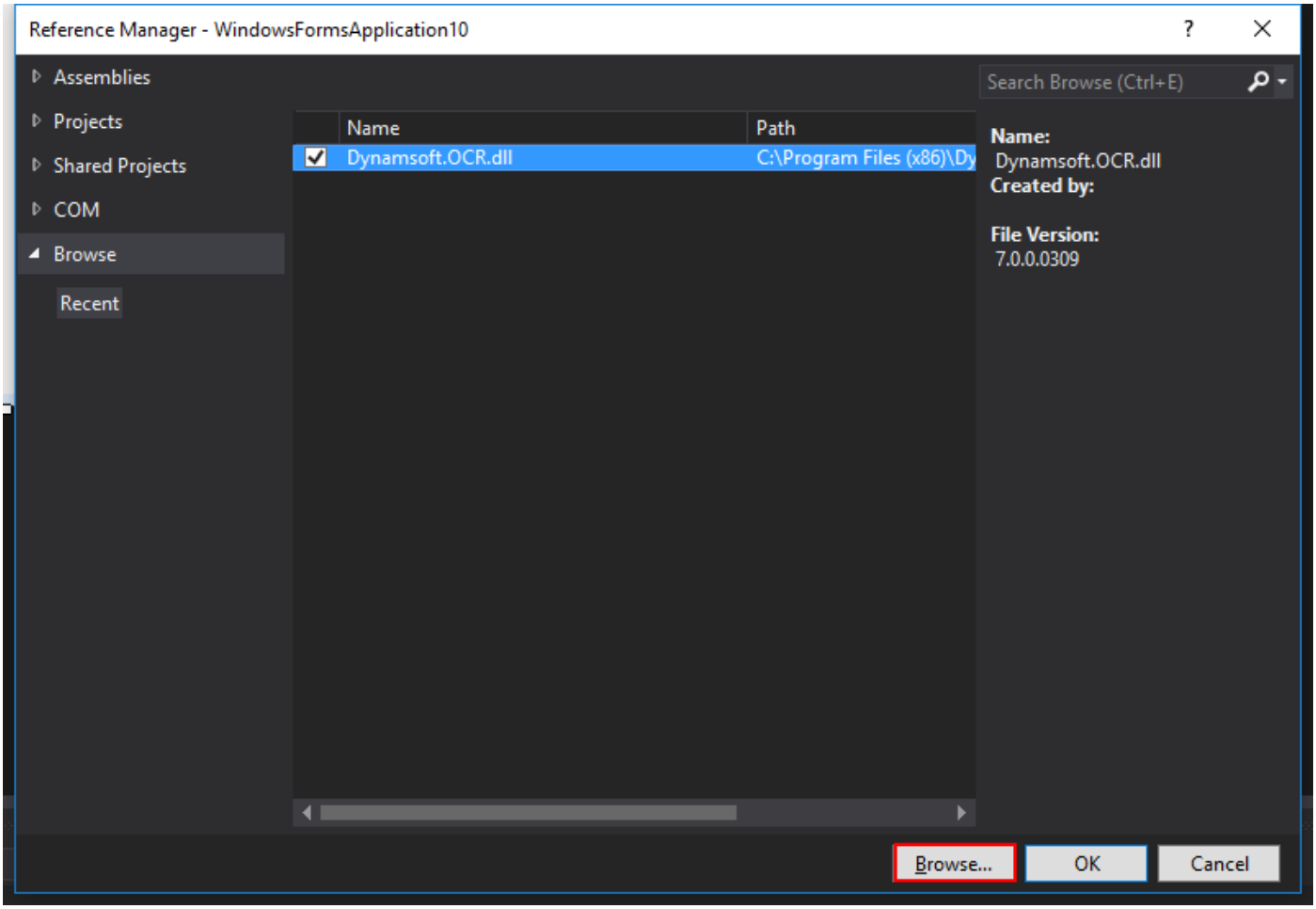
How to use OCR?

Step 1: Add Dynamsoft.OCR.dll to your project

Right click on References to add reference



Open the local file system if the dll is not listed



Once you install Dynamic .Net TWAIN, you can find the DLLs in the below directories:

{installation directory}\Redistributable\Assembly\For .NETFramework 4.0 ***Dynamsoft.OCR.dll***

{installation directory}\Redistributable\Assembly\For .NETFramework 2.0 ***Dynamsoft.OCR.dll***

Step 2: Start executing OCR

Add namespace and initialize the component like following:

```
using Dynamsoft.OCR;

public partial class Form1 : Form
{
    private Tesseract ocr = null;
    public Form1()
```

```
        {
            InitializeComponent();
            ocr = new Tesseract("<Your product key here>");
        }

    }
```

Before executing OCR, you need to let your app refer to the language package we provided. The files should be deployed along with your app. In the example here, we use English language data which can be downloaded from Dynamsoft's website. Dynamsoft OCR is based on Tesseract and it supports more than 40 languages. Download Other Language Packages>>

Please specify the data path, language type, and OCR format. It will be more straightforward when you see the sample code below:

```
// Specify language folder
ocr.TessDataPath = "Path";
// OCR in English
ocr.Language = "eng";
// Get the result
ocr.ResultFormat = 0; // Save as text
string[] Filepath = { "<Your file path1>", "<Your file path2>" };
byte[] resultbyte = ocr.Recognize(Filepath);
```

At this point, an OCR result has been kept in the array **resultbyte**. The most important thing, for now, is how to save it as the required format. The result can either be text or searchable PDF. In the sample code above, we use the text format. You can use the PDF format if you like.

To explore more, please feel free to check out the OCR Sample
{installation directory}\Samples\C# Samples\OCRDemo

Load local image(s) into the buffer and show it in the viewer

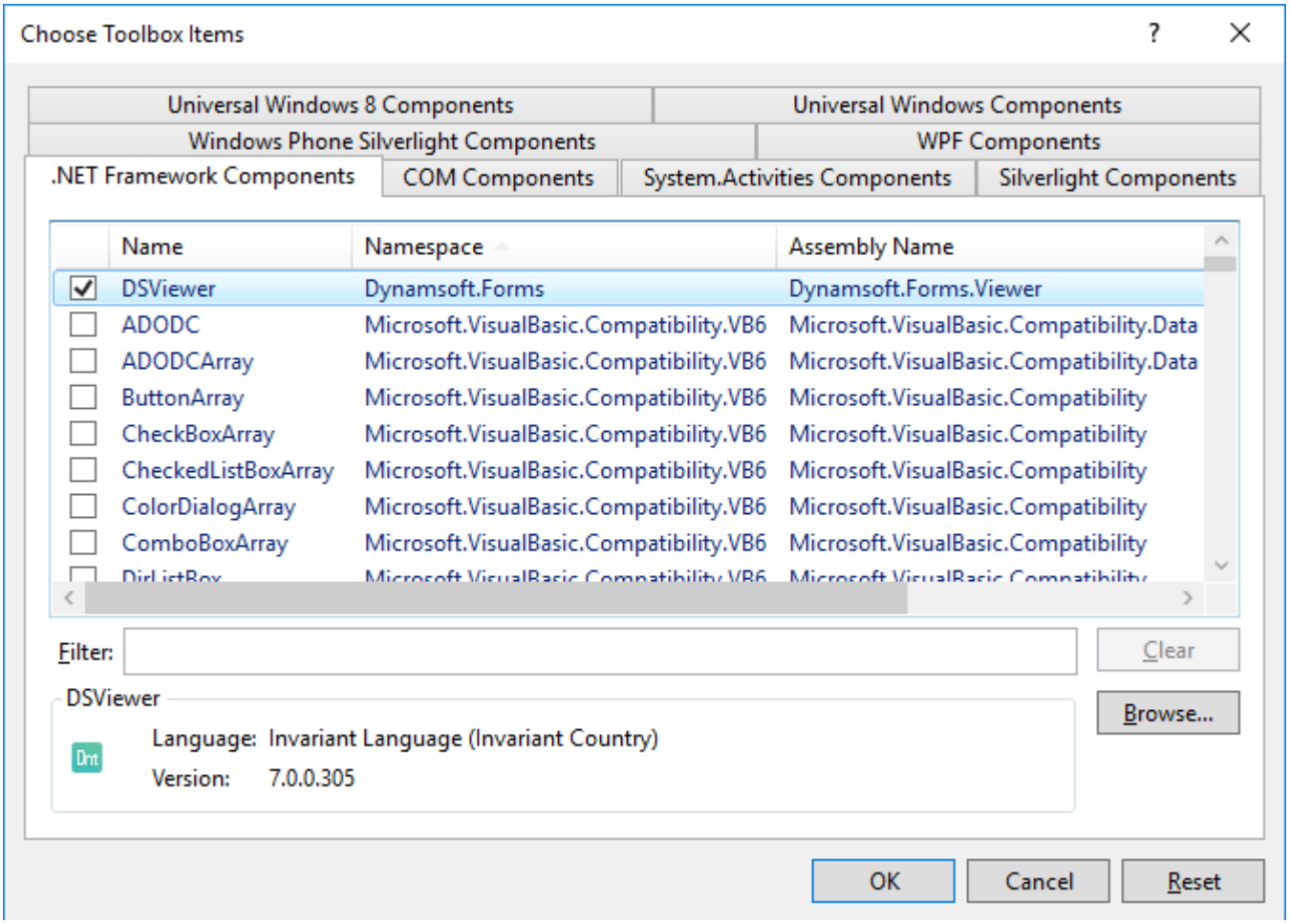
Preparation

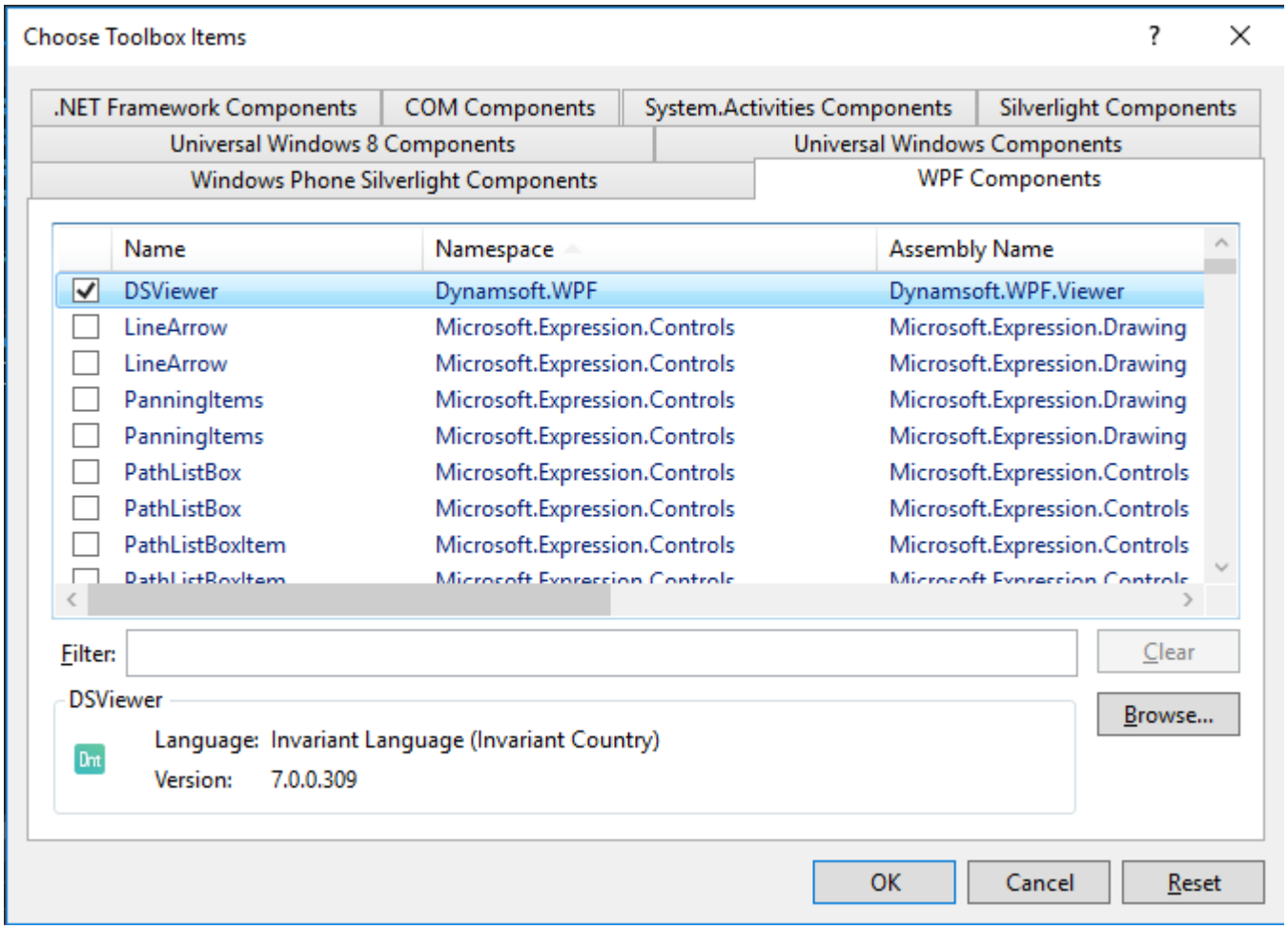
Dynamic .NET TWAIN supports loading and viewing image formats including BMP, JPEG, PNG, TIFF, and PDF. Please bear in mind the following points when trying to load images:

a. Almost all images in BMP, JPEG are supported
b. Currently, JPEG 2000 is not supported
c. Most images in PNG & TIFF are supported
d. Both image-only PDF and text-based PDF (via PDF Rasterizer module) are supported

Step 1: Add Dynamsoft.Froms.Viewer.dll to your project

1. Start Microsoft Visual Studio.
2. From the **Tools** menu, select **Choose Toolbox Items**.
3. Click the **.NET Framework Components** tab for Windows Forms Application or the **WPF Components** tab for WPF Application.
4. In the list, locate **DSViewer.**
5. If the control does not appear in the list, click the Browse button. In the Open dialog box, navigate to the installation folder (such as {installation directory}\Redistributable\For .NET 4.0) to find the file Dynamsoft.Forms.Viewer.dll (for Windows Forms Application) or Dynamsoft.WPF.Viewer.dll (for WPF Application), and click OK.

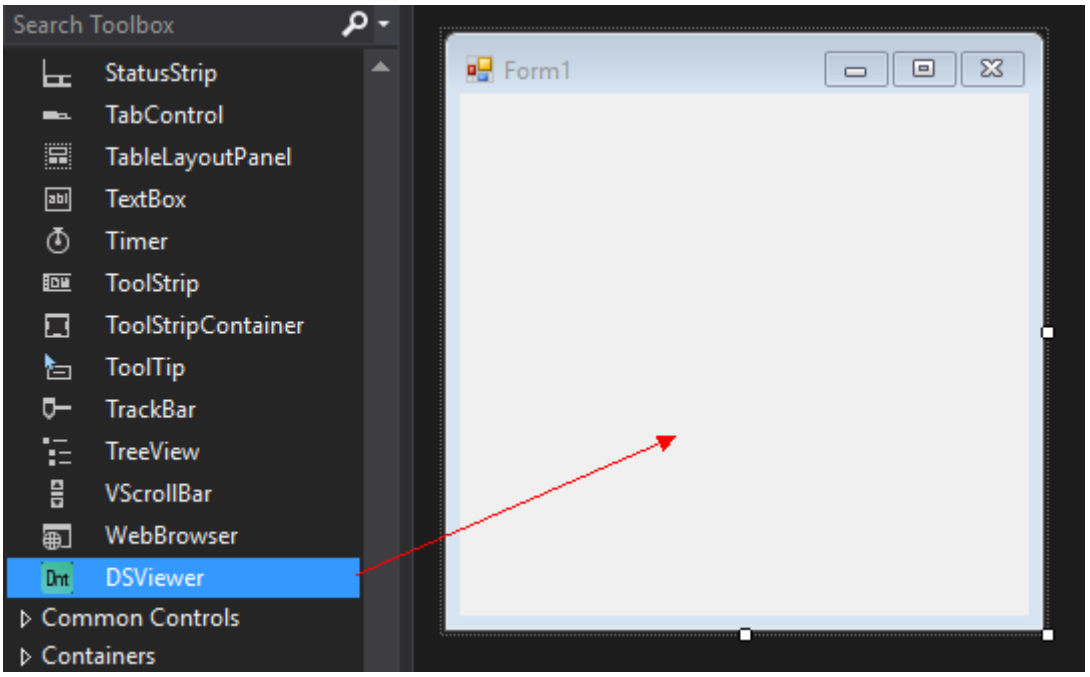

Choose Toolbox Items
Universal Windows 8 Components
Universal Windows Components
Windows Phone Silverlight Components
WPF Components
.NET Framework Components
COM Components
System.Activities Components
Silverlight Components
Name
Namespace
Assembly Name
DSViewer
Dynamsoft.Forms
Dynamsoft.Forms.Viewer
ADODC
Microsoft.VisualBasic.Compatibility.VB6
Microsoft.VisualBasic.Compatibility.Data
ADODCArray
Microsoft.VisualBasic.Compatibility.VB6
Microsoft.VisualBasic.Compatibility.Data
ButtonArray
Microsoft.VisualBasic.Compatibility.VB6
Microsoft.VisualBasic.Compatibility
CheckBoxArray
Microsoft.VisualBasic.Compatibility.VB6
Microsoft.VisualBasic.Compatibility
CheckedListBoxArray
Microsoft.VisualBasic.Compatibility.VB6
Microsoft.VisualBasic.Compatibility
ColorDialogArray
Microsoft.VisualBasic.Compatibility.VB6
Microsoft.VisualBasic.Compatibility
ComboBoxArray
Microsoft.VisualBasic.Compatibility.VB6
Microsoft.VisualBasic.Compatibility
Filter:
Clear
DSViewer
Language: Invariant Language (Invariant Country)
Version: 7.0.0.305
Browse...
OK
Cancel
Reset



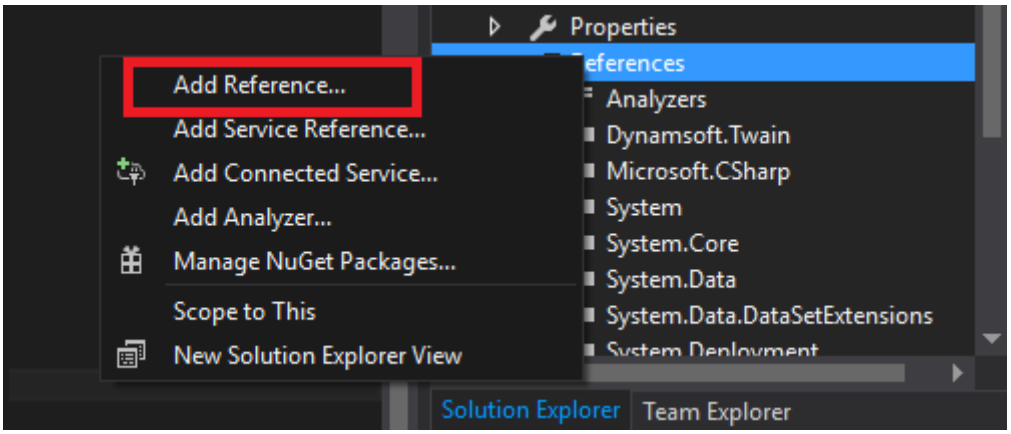
Step 2: Control the buffer

After creating a new project (take a Windows Forms Application as an example), the selected control will appear at the bottom of the selected tab in the Toolbox window.

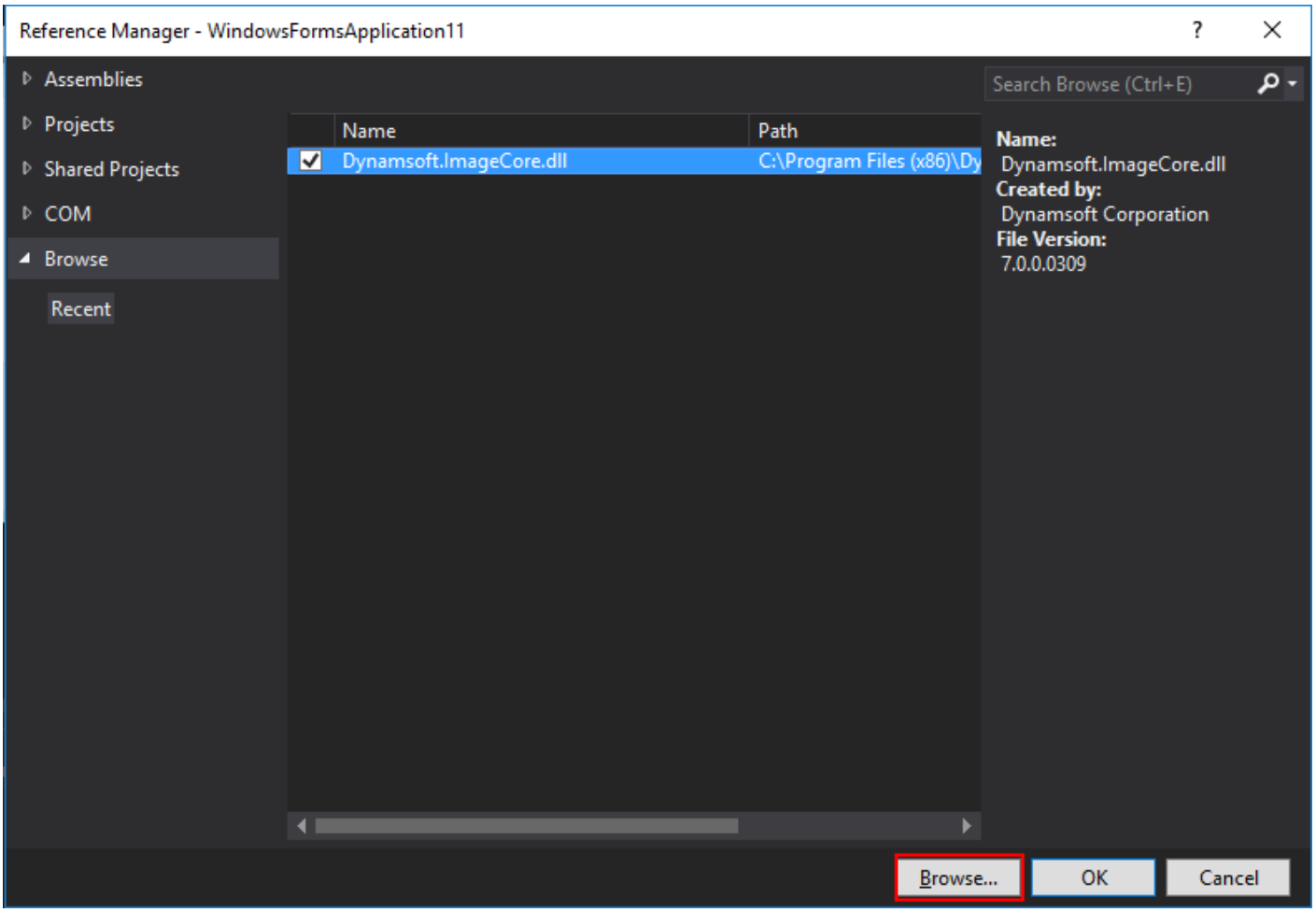
You can double click on the control or drag & drop it onto the form from the Toolbox.



Add the reference for ImageCore module. Right click on **Reference** to add Reference



Open the local file system if the dll is not listed



Add namespace and initialize the component like following:

```
using Dynamsoft.Core;

public partial class Form1 : Form

{
    private ImageCore imagecore = null;
    public Form1()
    {
        InitializeComponent();
        imagecore = new ImageCore();
        dsViewer1.Bind(imagecore);
    }

}
```

After binding the viewer with the core, the operation related to the buffer will be shown in the viewer. Now, we can load an image in buffer using the LoadImage method to show the image in the viewer:

```
imagecore.IO.LoadImage(@"G:/DotNetTWAIN/Images/ImageData.jpg");
```

If you need to load more than one image, you can make use of the OpenFileDialog function provided by the.NET Framework to show a file dialog and choose files form the file system. Below is a code snippet:

```
OpenFileDialog openfileDlg = new OpenFileDialog();
openfileDlg.Filter = "JPEG|*.jpg;*.jpeg";
openfileDlg.Multiselect = true;
if (openfileDlg.ShowDialog() == System.Windows.Forms.DialogResult.OK)
{
    for (int i = 0; i < openfileDlg.FileNames.Length; i++)
    {
        string fileName = openfileDlg.FileNames[i];
        imagecore.IO.LoadImage(fileName);
    }
}
```

Let's do some manipulations with the viewer.

Manipulate image(s)

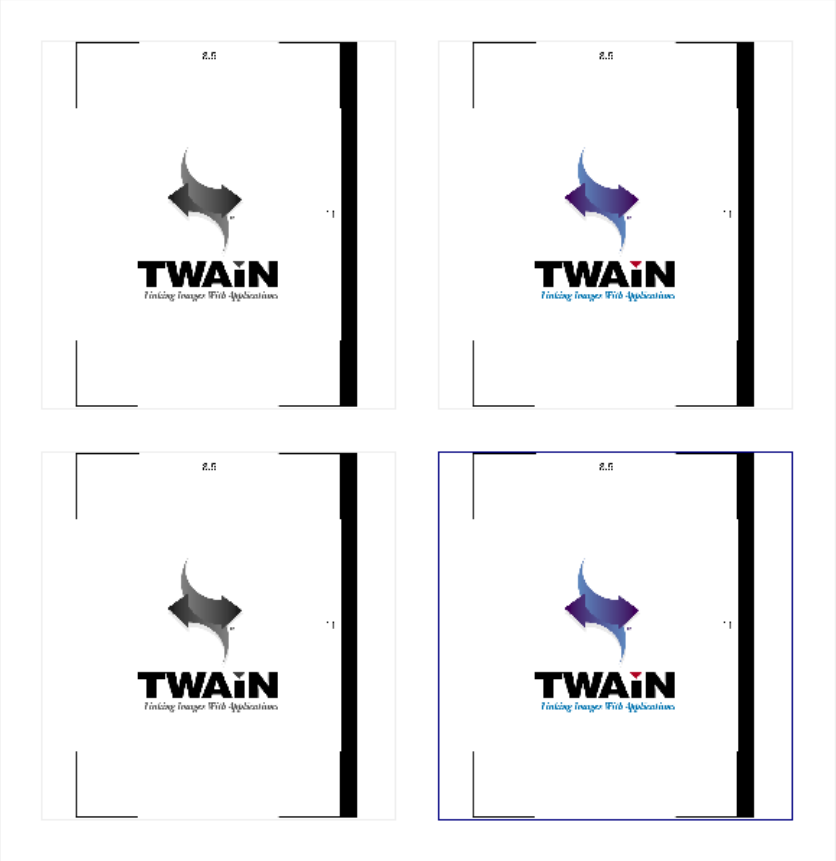
You can manipulate the images that you have scanned or loaded in the buffer.

1. Go through each image by changing the currently displayed image.

```
imagecore.ImageBuffer.CurrentImageIndexInBuffer = 2; //Show the 3rd image in buffer
```

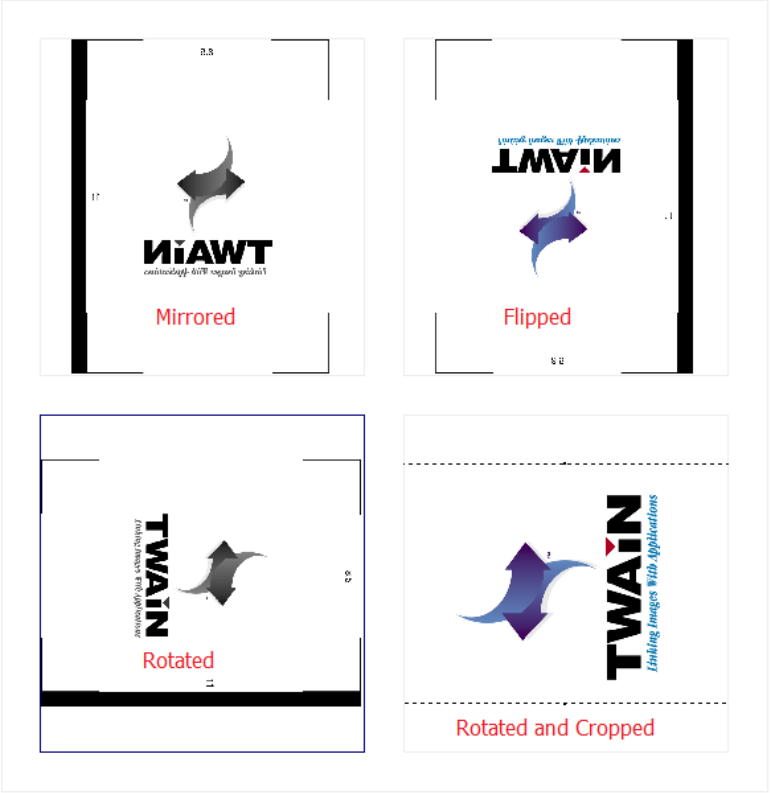
2. Show multiple images by changing the view mode to larger than 1x1.

```
dsViewer1.SetViewMode(2, 2);
```



3. Rotate, flip, mirror or crop, etc. an image.

```
imagecore.ImageProcesser.Mirror(0);
imagecore.ImageProcesser.Flip(1);
imagecore.ImageProcesser.RotateRight(2);
imagecore.ImageProcesser.Crop(3, 101, 243, 680, 831);
imagecore.ImageProcesser.RotateLeft(3);
```



Related methods

ChangeImageSize	Changes the display size of the image of a specified index in buffer.
CopyFrameToClipboard	Copies the image data in the specified area to the system clipboard in DIB format.
CopyToClipboard	Copies the image of a specified index in buffer to clipboard in DIB format.
Crop	Crops the image of a specified index in buffer.
CropToClipboard	Crops the image of a specified index in buffer to clipboard in DIB format.
CutFrameToClipboard	Cuts the image data in the specified area to the system clipboard in DIB format.
CutToClipboard	Cuts the image of a specified index in buffer to clipboard in DIB format.
Erase	Clears the specified area of a specified image, and fill the area with the fill color.
Flip	Flips the image of a specified index in buffer.
GetImageSize	Returns the file size of the new image resized from the image of a specified index in buffer.
GetImageSizeWithSpecifiedType	Pre-calculates the file size of the local image file that is saved from an image of a specified index in buffer.
GetViewMode	Gets the view mode that images are displayed in Dynamic .NET TWAIN.
GrayScale	Convert a specific image to a grayscale image.

Invert	Invert the color of a specific image.
IsBlankImage	Detects whether an image is blank.
LoadDibFromClipboard	Loads a DIB format image from Clipboard into Dynamic .NET TWAIN.
LoadImageFromBytes	Loads image from a byte array with the specified file format.
Mirror	Mirrors the image of a specified index in buffer.
MoveImage	Moves a specified image.
Picture	Returns the Picture object of the image of a specified index in buffer.
Print	Shows the GUI of Image Printer or print all images with the default printer.
RemoveAllImages	Removes all images in buffer.
RemoveImage	Removes the image of a specified index in buffer.
RemoveImages	Deletes the images of specified indices in the array.
Rotate	Rotates the image of a specified index in buffer by specified angle.
RotateLeft	Rotates the image of a specified index in buffer by 90 degrees counter-clockwise.
RotateRight	Rotates the image of a specified index in buffer by 90 degrees clockwise.
SaveImageToBytes	Saves the image of a specified index in buffer to a byte array in the specified file format.
SetViewMode	Sets the view mode that images are displayed in Dynamic .NET TWAIN.
SwitchImage	Switch two images of specified indices in buffer.

Save and Upload Images

Save the image(s) to a local system

You can use Dynamic .NET TWAIN to save the scanned image(s) to a local file system. Currently we support BMP, JPEG, PNG, TIFF (single and multi-page) and PDF (single and multi-page) formats. Below are the functions for saving:

SaveAsTIFF, SaveAsPNG, SaveAsJPEG, SaveAsBMP

Save a single image

For these methods, they normally require two parameters: the full path for the local file system (including the file name) and the image index in the buffer. For example:

```
//Save the first image in buffer as C:\temp\test.jpg
imagecore.IO.SaveAsJPEG("C:\\temp\\test.jpg", 0);
```

Save selected image(s) as TIFF

These methods have two parameters: full path for the local file system (including the file name) and the indices of images that need to be saved.

```
//Save selected images in buffer as C:\temp\test\test.tiff
imagecore.IO.SaveAsTIFF("C:\\temp\\test.tiff", [1,2]);
```

Save the image(s) to a local database

Normally we can save the image data as bytes in a database. We have one method to save image(s) to bytes, so later you can store them in a database.

SaveImageToBytes method.

Here is one example to save the first image as a jpg in the table 'testTable':

```
// Connect to SQL
System.Data.SqlClient.SqlConnection conn = null;
conn = new
System.Data.SqlClient.SqlConnection(ConfigurationManager.ConnectionStrings["ConnStr"].ConnectionStri
ng);
```

```
            conn.Open();

            string sqlcmd = "Insert into [testTable] (id, myImage) Values (1, @Pic)";
            System.Data.SqlClient.SqlCommand insertCommand = new
System.Data.SqlClient.SqlCommand(sqlcmd, conn);
            // For image data, we save the bytes into the database. We save the image to the JPG
format bytes.
            insertCommand.Parameters.Add("Pic", SqlDbType.Image, 0).Value =
imagecore.IO.SaveImageToBytes(0,     Dynamsoft.Core.Enums.EnumImageFileFormat.WEBTW_JPG);
            int queryResult = insertCommand.ExecuteNonQuery();
            if (queryResult == 1)
                MessageBox.Show("Success");
```

Upload the image(s) to a web server

You can use HTTP Post or Put to upload image(s) to a web server. For HTTP Post, the web server should provide an action page/handler to receive the HTTP Post request.

There are two methods to upload the image(s).

To upload to the server, we need to provide the web server information to the HTTP Upload methods. Here is an example:

```
            // If we want to post the image data to http://www.dynamsoft.com/Actions/SaveImages.aspx:
            string webserverAddress = "www.dynamsoft.com";
            string actionPageAddress = "/Actions/SaveImages.aspx";
            string fileName = "test.pdf";
            byte[] byte1 =
imagecore.IO.SaveImageToBytes(imagecore.ImageBuffer.CurrentImageIndexInBuffer,
EnumImageFileFormat.WEBTW_JPG);

            // Different web server may have a different HTTP port. Normally it is 80 for http://
            imagecore.Net.HTTPPort = 80;

            // If the server URL is in SSL, then you should change the port and SSL settings
            // imagecore.Net.HTTPPort = 443; //443 is the default port for SSL
            // imagecore.Net.IfSSL = true;
        imagecore.Net.HTTPUploadThroughPost(webserverAddress, actionPageAddress, fileName,byte1);
```

After the post, you can use HTTPPostResponseString properties to check the result of the post. If the upload succeeded, the action page should return an empty response. Otherwise, you will get an exception such as “HTTP process error”.

If your web server requires authentication (basic or windows authentication), you need to make sure the login information is sent along with the HTTP post. You can use HTTPUserName and HTTPPassword to set the login user name and password.

For the action page/handler, you just need to capture the posted data and save it the way your application needs it. By changing this action page, you can also save the images into the database on the server, etc.

```
HttpFileCollection files = HttpContext.Current.Request.Files;
HttpPostedFile uploadfile = files["RemoteFile"];
//Save to ImageScanned under the current folder
uploadfile.SaveAs(System.Web.HttpContext.Current.Request.MapPath(".") + "/ImageScanned/" +
uploadfile.FileName);
```

#The action page can be written in any server-side language which is capable of processing a HTTP post request.

Upload image(s) with meta data

If you want to send extra data with the HTTP Post methods, you can use the method SetHTTPFormField.

```
imagecore.Net.SetHTTPFormField("documentType", "Passport");
imagecore.Net.SetHTTPFormField("uploadBy", "Robby");
```

On the action page/handler, you can retrieve the extra data just like normal http post parameters:

```
string documentType = HttpContext.Current.Request["documentType"];
string uploadBy = HttpContext.Current.Request["uploadBy"];
```

Download image(s)

If the image is accessible with a URL, you can use the following methods to load it into Dynamic .NET TWAIN:

HTTPDownload and HTTPDownloadEx methods.

```
// If we want to download and load the image from http://www.dynamsoft.com/images/Cache_Server.jpg:
string webserverAddress = "www.dynamsoft.com";
string remoteFile = "/images/Cache_Server.jpg";

// Different web server may have a different HTTP port. Normally it is 80 for http://
imagecore.Net.HTTPPort = 80;

// If the server URL is in SSL, then you should change the port and SSL settings
// imagecore.Net.HTTPPort = 443; //443 is the default port for SSL
// imagecore.Net.IfSSL = true;
```

```
imagecore.Net.HTTPDownload(webserverAddress, remoteFile);
```

Useful Resources

Sample Download

http://www.dynamsoft.com/Downloads/.Net-TWAIN-Sample-Download.aspx

Knowledge Base

http://developer.dynamsoft.com/dnt/kb

FAQ

http://www.dynamsoft.com/Products/.Net-TWAIN-Scanner-FAQ.aspx

All APIs (Property, Method, Capability, and Event)

Plain HTML Help

CHM Help

Contact Us

Email: mailto:support@dynamsoft.com

Online Chat: http://www.dynamsoft.com/Support/LiveHelp.aspx

Phone Number: 1-877-605-5491 or 1-604-605-5491

Telephone: 1-877-605-5491 (Toll Free); 1-604-605-5491

FAX: 1-866-410-8856