TwainScanning

About

TwainScanning is a .NET library that enables you to acquire images from any device that has TWAIN drivers and convert acquired images to various formats including bitmap, jpeg, tiff and pdf.

Installation

Download file from the following link http://twainscanning.net/free-download and extract files from the archive.

Setting up the project

- Add reference TwainScanning.dll into your project.
- Set "Solution Platform" to "x86".

Providing the license key

For evaluation purposes, one can skip this step. If it is skipped all functionality will remain but acquired images will have watermark written on them.

When you buy the license, you will receive serial key valid only for your company. To provide the license key to the library call the `GlobalConfig.SetLicenseKey` function.

Using the library

Initializing data source manager (TWAIN)

Instantiate an instance of the `DataSourceManager` providing them your project's main form and additional info about your application.

By instantiating `DataSourceManager` TWAIN interface (source manager) is opened and initialized, the drivers are loaded and ready to use.

```
var dsm = new DataSourceManager(this); // opens TWAIN inside form
```

If the TWAIN is not supported by the operating system or source manager cannot be opened, instantiation will throw exception and object will not be created.

By disposing of this object TWAIN interface will close, and resources will be freed, so don't forget to dispose it.

Choosing the data source (scanner)

There are two ways to select scanner: from code or by displaying TWAIN built in GUI to choose the source.

From code

List of all available scanners, as a list of the `TwIdentity` objects, can be obtained through the `DataSourceManager.AvailableSources()` method. Use one of the objects from the list to open specific data source (scanner).

```
//listing and writing scanner names
List<TwIdentity> scannerIds = dsm.AvailableSources();
foreach (TwIdentity id in scannerIds)
       Console.WriteLine(id.ProductName);
```

By built-in GUI

Call the `DataSourceManager.SelectDefaultSourceDlg()` method to show dialog on which one can select default data source (scanner).

```
dsm.SelectDefaultSourceDlg();//Setting the default scanner by default twain
```

Opening the data source (scanner)

Data source (scanner) is opened by calling `DataSourceManager.Open()`. If as an argument is passed device name or an instance of the `TwIdentity` object, specific data source will be opened. Otherwise the default data source will be opened.

```
var ds = dsm.OpenSource(); // opens default source
var ds = dsm.OpenSource("ScannerXYZ"); // opens default source ScannerXZY
```

This function returns the `DataSource` object which represents opened, initialized and useful data source (scanner). If it fails to open (for example if a scanner is turned off or it is already opened), the function will throw.

By disposing `DataSource` object the data source (scanner) will be closed and resources freed. If one forgets to dispose this object's data source, it may remain in opened state. That may prevent opening this scanner again.

Basic settings

The `DataSource` object exposes some of the most commonly used setting for easy access through these fields: `PageSize`, `ColorMode`, `PixelDepth`, `Resolution`, `UseFeeder`, `UseDuplex` and `TransferMechanism`.

```
ds.Resolution.Value = 200f;                //setting resolution
ds.TransferMechanism.Value = TwSX.Memory;  //setting transfer mechanism
float resolution = ds.Resolution.Value;    //getting resolution
TwSX mech = ds.TransferMechanism.Value;    //getting transfer mechanism
```

Setting and getting values on these fields is accomplished through the `Value` property.

```
float[] resolutions = ds.Resolution.AvailableValues; //supported resolutions
TwSX[] mechs = ds.TransferMechanism.AvailableValues; //supported mechanism
```

Finding all available values on these fields is accomplished through the `AvailableValues` property.

All settings (capabilities)

All capabilities of the data source (scanner) can be accessed through the `Settings` object on the `DataSource`. Capabilities are arranged into the categories. For a full list of categories and containing capabilities see (Appendix: List of capabilities).

The value of specific capability can be obtained through the Value property on the capability object. If a capability is read-only Value setter is not implemented. If a capability can have multiple values, the Value property is an array.

All supported values can be retrieved through `SupportedValues` property.

`IsSupportedOnThisDevice()` method returns if this opened data source supports this capability.

```
var capOrientation = ds.Settings.ImageAcquire.Orientation; // info about orientation
if (capOrientation.IsSupportedOnThisDevice()) // can scanner support orientation?
{
    TwOR orientation = capOrientation.Value;
    Console.Write(" Current orientation is " + orientation);

    Console.Write(" All possible orientations are:");
    foreach(TwOR orient in capOrientation.SupportedValues)
        Console.Write(" " + orient);

    capOrientation.Value = TwOR.Landscape; //set orientation to Landscape
}
```

Acquiring

There are two ways of scanning available: synchronous and asynchronous.

Synchronous scanning

Synchronous scanning starts by calling `DataSource.Acquire(…)` method. This method doesn't return until scanning is finished.

It is the recommended way for scanning in the console applications.

```
TwImgCollector coll1 = ds.Acquire(false);// scanning has started (no scanner gui)
coll1.SaveAllToBitmaps(...) ;//scanning has finished and collector is ready

TwImgCollectorPdf coll2 = new TwImgCollectorPdf(@"C:\someDir\someFile.pdf");
ds.Acquire(coll2, false);// scanning has started (no scanner gui)
coll2.Dispose(); // scanning has finished ()
```

Asynchronous scanning

Asynchronous scanning starts by calling `DataSource.AcquireAsync(…)` method. This method returns immediately, and information about scanning status is provided using events only (see Events chapter).

It is the recommended way for scanning in the non-console applications.

```
private void Scan(){
    ds.AcquireAsync(onFinishedScanning, true); );// scanning has started
}

private void onFinishedScanning(TwImgCollector collector){ // scanning has finished
    collector.SaveAllToMultipageTiff(@"C:\someFolder\someFile.tiff");
}

private void Scan(){
    var collector = new TwImgCollectorPdf(@"C:\someFolder\someFile.tiff");
    ds.OnScanningFinished += ds_OnScanningFinished;
    ds.AcquireAsync(collector, true);  // scanning has started
}

private void OnScannFinished(object sender, DataSource.ScanningFinishedEventArgs e){
    e.Colletor.Dispose();  // scanning has finished
}
```

Collectors

Collectors can be used to collect scanned images (see chapter: Collectors). They are either provided by the user to the acquire method, or the method will create the `TwImgCollector` and return it to the user (depending on which overloaded function is called).

Transfer Mechanism

For the acquiring the user can specify one of the four transfer mechanisms:

- Native Transfer: the data source delivers a whole image at once in the form of the single bitmap.
- Buffered Memory Transfer: the data source delivers an image in a series of chunks of a bitmap. This mode will trigger the progress event.
- File Transfer: the data source doesn't deliver an image to the library but saves an image to a file specified by the user. Collectors are not used in this scenario.
- Buffered Memory File Transfer: the data source delivers an image in a series of chunks of an image file. This mode will trigger the progress event, and collectors are not used in this scenario.

Other parameters

When acquiring user can specify some additional things:

- should scanner's GUI be displayed
- if the GUI is displayed should it automatically close when scanning is finished
- number of images to scan in the current batch

Events

During the acquiring process several events can be monitored:

- `OnSingleImageAcquired` is triggered when one image is acquired. The user can access this image in the form of `System.Drawing.Bitmap`.
- `OnBatchFinished` is triggered when all images from a single job are acquired. For example, all documents from feeder are scanned, or a single document from the flatbed is scanned.
- `OnScanningFinished` is triggered when scanning ends. For example, the user closed scanner's GUI.
- `OnMemoryTransferProgressUpdate` is triggered when a chunk of the image is delivered to this library. It is useful only for when the `TwSX.Memory` transfer is used.
- `OnErrorEvent` triggered when something went wrong during acquiring process. For example: a paper jam, the device went offline.

Collectors

Collectors are objects which are used to collect acquired images. Collectors are typically instantiated by the user and passed to the data source through some of the acquiring methods. While it is the preferred way to deal with image one is not obligated to use collectors. Collectors can be used for multiple scans; new images will be appended to the old ones.

There are several types of collectors:

TwImgCollector

Most commonly used collector and most powerful. Can save collected images to various formats: bitmap, jpeg, tiff, multipage tiff, pdf and multipage pdf. It uses the hard drive to temporary store scanned images. This feature can be disabled and images will be stored in memory, but then there is a risk of low memory exception.

TwImgCollectorPdf
Stores all acquired images to the single PDF. It doesn't use a hard drive to store temporary images, and it can process the image while the next one is being scanned. The file will be saved on disposing.

TwImgCollectorTiff
Stores all acquired images to the single tiff. It doesn't use a hard drive to store temporary images, and it can process the image while the next one is being scanned. The file will be saved on disposing.

TwImgCollectorPdfSinglepage
Stores all acquired images to the separate PDFs. It also doesn't use a hard drive to store temporarily.

TwImgMultiCollector
Used when the functionality of the several different collectors should be combined; for example: when one must save images to pdf and tiff at the same time. Constructed with multiple collectors. Appends scanned images to every collector that it owns.

One can also create its own collector by implementing IImgCollector interface.

IMPORTANT: All collectors must be disposed after use.

Appendix: List of capabilities

- AsyncDeviceEvents
  - DeviceEvent
- AudibleAlarms
  - Alarms
  - Volume
- AutomaticAdjustments
  - AutomaticSenseMedium
  - AutoDiscardBlankPages
  - AutomaticBorderDetection
  - AutomaticColorEnabled
  - AutomaticColorNonColorPixelType
  - AutomaticCropUsesFrame
  - AutomaticDeskew
  - AutomaticLengthDetection
  - AutomaticRotate
  - AutoSize
  - FlipRotation
  - ImageMerge
  - ImageMergeHeightThreshold
- AutomaticCapture
  - NumberOfImages
  - TimeBeforeFirstCapture
  - TimeBetweenCaptures
- AutomaticScanning
  - AutoScan
  - CameraEnabled
  - CameraOrder
  - CameraSide
  - ClearBuffers
  - MaxBatchBuffers
- BarCodeDetection
  - Enabled
  - SupportedTypes
  - MaxRetries
  - MaxSearchPriorities
  - SearchMode
  - SearchPriorities
  - Timeout
- Caps
  - ExtendedCaps
  - SupportedCaps
  - SupportedDats
- Color
  - ColorManagementEnabled
  - Filter
  - Gamma
  - IccProfile
  - PlanarChunky
- Compression
  - BitOrderCodes
  - CcittKFactor
  - Method
  - JpegPixelType
  - JpegQuality
  - JpegSubSampling
  - PixelFlavorCodes
  - TimeFill
- Device
  - Online
  - TimeDate
  - SerialNumber
  - MinimumHeight
  - MinimumWidth
  - ExposureTime
  - FlashUsed2
  - ImageFilter
  - LampState
  - LightPath
  - LightSource
  - NoiseFilter
  - OverScan
  - PhysicalHeight
  - PhysicalWidth
  - Unit
  - ZoomFactor
- DoublefeedDetection
  - Mode
  - DoubleFeedDetectionLength
  - DoubleFeedDetectionSensitivity
  - DoubleFeedDetectionResponse
- ImprinterEndorser

    - Endorser
    - Imprinter
    - ImprinterEnabled
    - ImprinterIndex
    - ImprinterMode
    - PrinterString
    - PrinterSuffix
    - PrinterVerticalOffset
    - PrinterCharRotation
    - PrinterFontStyle
    - PrinterIndexLeadChar
    - PrinterIndexMaxValue
    - PrinterIndexNumDigits
    - PrinterIndexStep
    - PrinterIndexTrigger
    - PrinterStringPreview
- ImageInformation
    - Author
    - Caption
    - TimeDate
    - ExtImageInfo
    - SupportedExtImageInfo
- ImageAcquire
    - ThumbnailsEnabled
    - AutoBright
    - Brightness
    - Contrast
    - Highlight
    - ImageDataSet
    - Mirror
    - Orientation
    - Rotation
    - Shadow
    - XScaling
    - YScaling
- ImageType
    - BitDepth
    - BitDepthReduction
    - BitOrder
    - CustHalftone
    - Halftones
    - PixelFlawor
    - PixelType
    - Threshold
- Language
    - TheLanguage
- MICR
    - Enabled
- Page
    - Segmented
    - Frames
    - MaxFrames
    - Sizes
- Duplex
    - Mode
    - Enabled
- Feeder
    - Autofeed
    - ClearPage
    - Alignment
    - Enabled
    - Loaded
    - Order
    - Pocket
    - Prepare
    - PaperDetectable
    - PaperHandling
    - ReacquireAllowed
    - RewindPage
    - Type
- PatchCodeDetection
    - Enabled
    - SupportedTypes
    - MaxSearchPriorities
    - SearchPriorities
    - SearchMode
    - MaxRetries
    - Timeout
- PowerMonitoring
    - BatteryMinutes
    - BatteryPercentage
    - PowerSaveTime
    - PowerSupply
- Resolution
    - X
    - Y
    - XNativeRes
    - YNativeRes

- Transfer
    - JobControl
    - ImageCount
    - Compression
    - ImageFileFormat
    - Tiles
    - UndefinedImageSize
    - Mechanism
- UserInterface
    - CameraPreviewUI
    - CustomDSData
    - CustomInterfaceGuid
    - EnableDSUIOnly
    - Indicators
    - IndicatorsMode
    - UIControllable

Appendix: Examples

Minimal example

Minimal but useful example.

```
// Minimal example using scanners UI
using (var dsm = new DataSourceManager(this))                   // opens TWAIN
using (var ds = dsm.OpenSource(dsm.SelectDefaultSourceDlg())) // opens default source
using (var collector = ds.Acquire(true))                        // acquires images
{
    collector.SaveAllToJpegs(@"C:\SomeFolder\Some\name.jpeg");// saves images to disk
}                                                               // closes ds and dsm
```

Scanning in Windows Form

```
public partial class ScanningForm : Form
{
    DataSourceManager dsm = null;
    DataSource ds = null;
    public ScanningForm()
    {
        dsm = new DataSourceManager(this);
        InitializeComponent();
    }
    private void buttonScann_Click(object sender, EventArgs e)
    {
        dsm.SelectDefaultSourceDlg(); // sets default scanner
        ds = dsm.OpenSource();        // opens default scanner
        ds.AcquireAsync(onFinishedScanning, true, true, TwSX.Memory, -1);
    }
    private void onFinishedScanning(ImageCollector collector)
    {
        collector.SaveAllToMultipageTiff(@"C:\someFolder\someFile.tiff"); //saving
        collector.Dispose();    //disposing collector
        ds.Dispose();           //disposing scanner
        ds = null;
    }
    private void ScanningForm_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (ds != null) ds.Dispose(); ds = null;
        if (dsm != null) dsm.Dispose(); dsm = null;
    }
}
```