Dynamic Web TWAIN

Developer's Guide

July 2018

Dynamsoft

15 Years of Experience in TWAIN SDKs and Version Control Solutions

Contents

Preface

Description

This guide provides instructions on how to use Dynamsoft’s Dynamic Web TWAIN SDK. It provides an overview of most of the things you can achieve with the SDK.

Audience

This guide is meant for all developers interested in the Dynamic Web TWAIN SDK.

For new developers, there is a step-by-step guide to help you develop a scanning page in your web application from scratch.

For those who have used the SDK before, you can find information on advanced or new APIs that you can use to polish your scanning page.

Getting Started

What is TWAIN | ICA | SANE

TWAIN is a standard software protocol and application programming interface (API) that regulates communication between software applications and imaging devices such as scanners and digital cameras.

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

Dynamsoft Corporation is a member of the TWAIN Working Group.

The TWAIN protocol works very well on Windows but not as good on macOS. Thus, a substitute is also used by Dynamsoft's SDK on macOS which is called Image Capture Architecture or ICA for short.

On **Linux**, TWAIN isn't available, therefore **SANE** is used. As described on the official introduction page, **SANE** stands for "Scanner Access Now Easy" and is an application programming interface (API) that provides standardized access to any raster image scanner hardware (flatbed scanner, hand-held scanner, video- and still-cameras, frame-grabbers, etc.).

What is Dynamic Web TWAIN

Dynamic Web TWAIN is a scanning SDK specifically optimized for **web applications**. It was designed for Windows only at the beginning and only TWAIN protocol was supported at the time which is why ***TWAIN*** is in the name of the SDK. However, it has been greatly improved and extended over the years and now it supports TWAIN on Windows & macOS, ICA on macOS and SANE on Linux. The SDK enables you to write code, in just a few lines, to **scan** documents from a **TWAIN|ICA|SANE Compliant** device which typically is a scanner. Users can then **edit** the images, **save** them locally, or **upload** them to a remote server in a variety of formats.

With the SDK, you can also import files in the formats BMP|JPG|PNG|TIF|PDF from a local disk or from the web via HTTP(s) or FTP.

Basic Requirements

- **Server Side:**
  - Operating System: Windows, macOS, Linux, etc.
  - Web Server: IIS, Apache, Tomcat, ColdFusion, etc.
  - Programming Languages:
    - Front-end: HTML, JavaScript, TypeScript, CSS, etc.
    - Back-end: ASP.NET (C# and VB), PHP, JSP (JAVA), ASP, Python, NodeJS, etc.
- **Client Side:**
  - Browser/OS Support
    - Windows XP/7/8/2008/2012/2016 and 10; 32-bit and 64-bit
      - IE 6-9: **ActiveX**
      - IE 10-11: **HTML5/ActiveX**
      - Edge: **HTML5**
      - Chrome/Firefox 27+: **HTML5**
    - Mac OS X 10.6.8 and later
      - Chrome/Firefox 27+, Safari 7+: **HTML5**
    - Ubuntu 12.0.4+, Debian 8+, Fedora 24+, mint 18.3; 64-bit
      - Chrome/Firefox 27+: **HTML5**
    - IOS *(with optional MBC edition, not part of the main installer as of v14.0)*
      - Safari v11+: **MBC**
    - Android *(with optional MBC edition, not part of the main installer as of v14.0)*
      - Chrome v58+: **MBC**
  - Scanner/Camera/Other Devices (must be TWAIN compliant)

Deciding which Dynamic Web TWAIN Edition to use

Dynamic Web TWAIN has five editions: ActiveX, HTML5 for Windows, HTML5 for Mac, HTML5 for Linux and Mobile Browser Capture for IOS & Android. Based on the browser(s) your end users use, you can decide which edition(s) you need.

- **ActiveX**: supports IE 6-9 by default, it can be configured to support IE 10, 11 as well
- **HTML5 for Windows**: supports Firefox/Chrome 27+, IE 10/11 and Edge
- **HTML5 for Mac**: supports Chrome/Firefox 27+, Safari 7+
- **HTML5 for Linux**: supports Chrome/Firefox 27+
- **MBC for IOS & Android**: supports Safari v11+ on IOS and Chrome v58+ on Android

Building the "Hello World" Scan Page

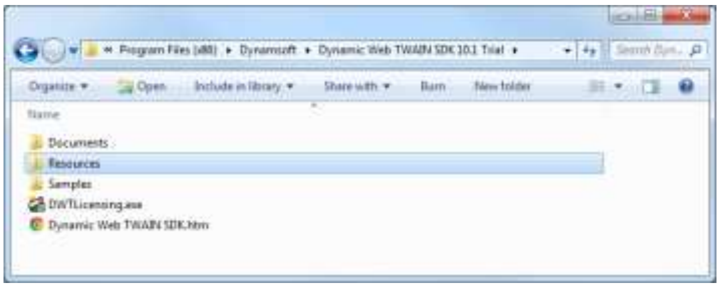
NOTE: Before you start, please make sure you've downloaded and installed the latest version of Dynamic Web TWAIN. If you haven't done so, you can get the 30-day free trial here.

The following 3 steps will show you how to create your first web-based scanning application in just 5 minutes!

Step 1: Start a Web Application

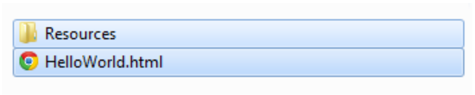
1.1 Copy the Dynamsoft Resources folder to your project

The Resources folder can normally be copied from C:\Program Files (x86)\Dynamsoft\Dynamic Web TWAIN SDK {Version Number} {Trial}\



1.2 Create an empty HTML page

Put an empty html page together with the Resources folder, as shown below



Step 2: Add Dynamic Web TWAIN to the HTML Page

2.1 Include the two Dynamsoft JS files in the <head> tag

```
<script src="Resources/dynamsoft.webtwain.initiate.js"></script>
<script src="Resources/dynamsoft.webtwain.config.js"></script>
```

2.2 Add Dynamic Web TWAIN container to the <body> tag

```
<div id="dwtcontrolContainer"></div>
```

***Note**: "dwtcontrolContainer " is the default id for the div. You can change it in the file dynamsoft.webtwain.config.js if necessary.*

Step 3: Use Dynamic Web TWAIN

3.1 Add a Scan button and the minimum code to scan

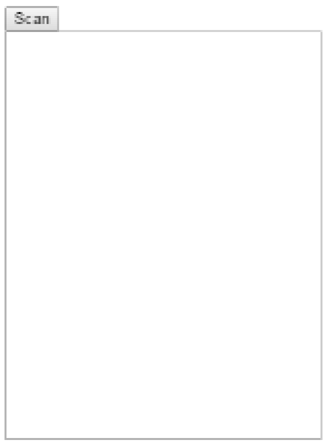
```
<input type="button" value="Scan" onclick="AcquireImage();" />
<script type="text/javascript">
    var DWObject;
    function Dynamsoft_OnReady() {
        DWObject = Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
    }
    function AcquireImage() {
        if (DWObject) {
            DWObject.SelectSource(function () {
                DWObject.OpenSource();
                DWObject.AcquireImage();
            }, function () {console.log("SelectSource failed!"); });
        }
    }
</script>
```

3.2 Review the completed code

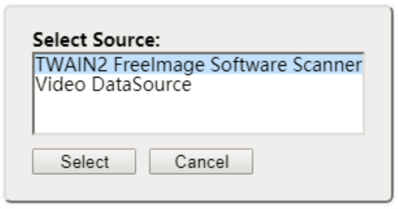
```
<html>
<head>
    <title>Hello World</title>
    <script src="Resources/dynamsoft.webtwain.initiate.js"> </script>
    <script src="Resources/dynamsoft.webtwain.config.js"> </script>
</head>
<body>
    <input type="button" value="Scan" onclick="AcquireImage();" />
    <div id="dwtcontrolContainer"> </div>
    <script type="text/javascript">
        var DWObject;
        function Dynamsoft_OnReady() {
            DWObject = Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
        }
        function AcquireImage() {
            if (DWObject) {
                DWObject.SelectSource(function () {
                    DWObject.OpenSource();
                    DWObject.AcquireImage();
                }, function () {console.log("SelectSource failed!"); });
            }
        }
    </script>
</body></html>
```

3.3 See the scan page in action

If you open the Hello World page in your browser, it should look like this:



Now, you can click on the **Scan** button to select a device, as shown below:

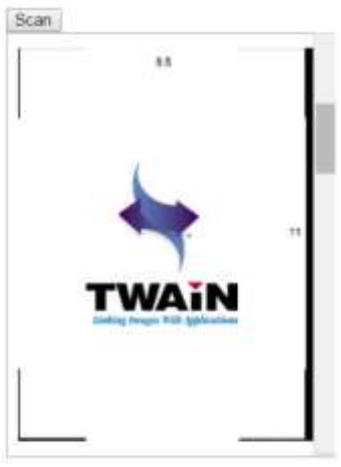


NOTE:

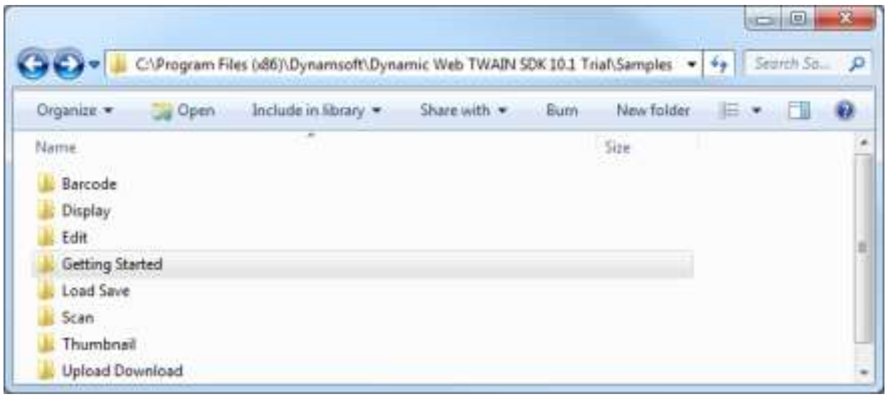
1) *Only **TWAIN|ICA|SANE**-compliant devices are listed in the Select Source dialog. If your connected scanner is not shown in the list, please follow this article to troubleshoot.*

2) *If you are using Windows and don't have a real scanner at hand, you can install the Virtual Scanner – a scanner simulator which is developed by the TWAIN Working Group for testing purposes.*

Once scanning is done, image(s) will show up in the built-in Dynamic Web TWAIN viewer:



If you have installed the 30-day trial version of Dynamic Web TWAIN, you can normally find the complete Hello World application in C:\Program Files (x86)\Dynamsoft\Dynamic Web TWAIN SDK {Version Number} {Trial}\Samples\Getting Started\.



C:\Program Files (x86)\Dynamsoft\Dynamic Web TWAIN SDK 10.1 Trial\Samples

Name	Size
Barcode	
Display	
Edit	
Getting Started	
Load Save	
Scan	
Thumbnail	
Upload Download	

As you can see, there are many other samples (source code provided) there for you to try out the many features of Dynamic Web TWAIN. We also provide advanced samples in the online sample gallery.

Customizing the Dynamic Web TWAIN Object

Change the name of the object

By default, the (first) Dynamic Web TWAIN object is named "**DWObject**". You should set it before using any other Dynamic Web TWAIN's properties or methods. A good place to do this is the built-in function ***Dynamsoft_OnReady***. For example, in our Hello World sample:

```
function Dynamsoft_OnReady() {
    DWObject = Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
}
```

The div with id 'dwtcontrolContainer' is the place holder for Dynamic Web TWAIN. Its name and size are defined in the file *dynamsoft.webtwain.config.js* as shown below. You can change it if necessary.

```
Dynamsoft.WebTwainEnv.Containers=[{ContainerId:'dwtcontrolContainer',Width:270,Height:350}];
```

Change the Size of the Viewer

You can change the size of the container (the built-in viewer) in *dynamsoft.webtwain.config.js.* You can use either a number or a percentage here. For example

```
Dynamsoft.WebTwainEnv.Containers = [{ ContainerId: 'dwtcontrolContainer', Width: '50%', Height: '513px' }];
```

Using Dynamic Web TWAIN

By default, Dynamic Web TWAIN is automatically initialized after the page finishes loading. Once the Dynamic Web TWAIN object is initialized, you can start to call its methods, set its properties, etc. You can refer to our online API Documentation to check all properties, methods and events of Dynamic Web TWAIN.

Properties

Properties are used to get or set a certain value in the Dynamic Web TWAIN object at runtime such as **Resolution**, **Duplex**, **IfShowUI**, etc.

```
DWObject.Resolution = 200; // Scan pages in 200 DPI
```

Methods

Methods are used to call the built-in functions of the Dynamic Web TWAIN object such as **AcquireImage**, **SaveAsJPEG**, **Rotate**, etc. The syntax is fairly simple:

```
DWObject.Rotate(0, 45, false); // rotate the 1st image in the buffer by 45 degrees
```

Events

Events are triggered when certain trigger points are reached. For example, we have an **OnMouseClick** event for mouse clicking, an **OnPostTransfer** event for the end of the transferring of one image, etc. Compared with Properties and Methods, Events are a bit tricky to use. We'll talk about it a little more here.

Handling Events

Add an event listener

To add an event listener, you can use the built-in method **RegisterEvent**. Please refer to the sample code below:

```
Dynamsoft.WebTwainEnv.RegisterEvent('OnWebTwainReady', Dynamsoft_OnReady);
var DWObject;
/* OnWebTwainReady event fires as soon as Dynamic Web TWAIN is initialized. It is the
best place to add event listeners */
function Dynamsoft_OnReady() {
    DWObject = Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
    DWObject.RegisterEvent("OnPostTransfer", Dynamsoft_OnPostTransfer);
}
function Dynamsoft_OnPostTransfer() {
    /* This event OnPostTransfer will be triggered after a transfer ends. */
    /* your code goes here*/
}
```

In the above code, we added the JavaScript function Dynamsoft_OnPostTransfer() as an event listener for the event **OnPostTransfer**. Alternatively, you can also write code as shown below:

```
Dynamsoft.WebTwainEnv.RegisterEvent('OnWebTwainReady', Dynamsoft_OnReady);
var DWObject;
function Dynamsoft_OnReady() {
    DWObject = Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
    DWObject.RegisterEvent("OnPostTransfer", function () {
        /* your code goes here*/
    };
}
```

Event with parameter(s)

Some of the events have parameter(s). Take the OnMouseClick event for an example:

```
OnMouseClick(short sImageIndex) /* sImageIndex refers to the image you clicked on*/
```

When you create the corresponding JavaScript function (A.K.A., the event listener), you can include the parameter(s) and retrieve the value at runtime.

```
function DynamicWebTwain_OnMouseClick(index) {
    console.log(index);
}
```

Or

```
DWObject.RegisterEvent("OnPostTransfer", function (index) {
    console.log(index);
};
```

Special Event - 'OnWebTwainReady'

To check all the events, please refer to the online API Documentation. Of these events, there is one called ***'OnWebTwainReady'*** that is special. Basically, this event fires as soon as the Dynamic Web TWAIN object is initialized. As you may have seen earlier in the document, the recommended way to use it is:

```
Dynamsoft.WebTwainEnv.RegisterEvent('OnWebTwainReady', Dynamsoft_OnReady);
var DWObject;
function Dynamsoft_OnReady() {
    DWObject = Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
}
```

Or

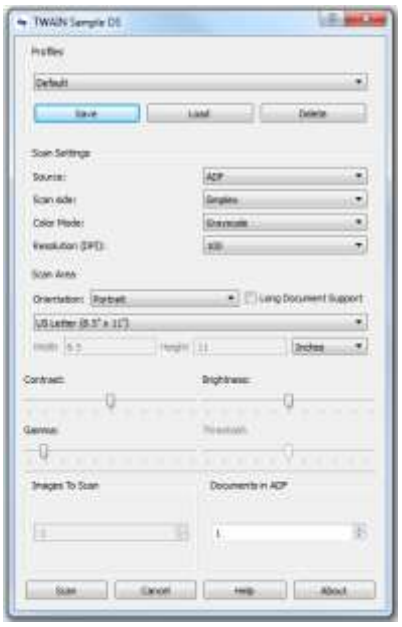
```
Dynamsoft.WebTwainEnv.RegisterEvent('OnWebTwainReady', function () {
    DWObject = Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
};
```

Exploring the Features

As we introduced in the previous section Three ways to use Dynamic Web TWAIN, you can control Dynamic Web TWAIN objects in three ways: Properties, Methods and Events. The complete list of all built-in properties, methods and events of Dynamic Web TWAIN is available in our online API Documentation for your reference. Here we will introduce Dynamic Web TWAIN's functionality in more details.

Customizing scan settings

Before you start an actual scan session, you can choose how you want your documents to be scanned. Normally, you can change all the settings in the scanner's built-in User Interface. Take the Virtual scanner for example:



Profiles: Default

Scan Settings

Source: ADF

Scan side: Simplex

Color Mode: Grayscale

Resolution (DPI): [illegible]

Scan Area

Orientation: Portrait ☐ Long Document Support

US Letter (8.5" x 11")

Width 8.5 Height 11 Inches

Contrast: Brightness:

Gamma: Threshold:

Images To Scan Documents in ADF 1

All these settings might be overwhelming for end users, especially for those without a technical background. With Dynamic Web TWAIN, you can customize all these settings in your JavaScript code. For example:

Manipulating the image(s)

When you have scanned or loaded images in Dynamic Web TWAIN, you can start manipulating the images. You can:

```
DWObject.SelectSource();
DWObject.OpenSource();// You should customize the settings after opening a source
DWObject.IfShowUI = false;// Hide the User Interface of the scanner
DWObject.IfFeederEnabled = true;// Use the document feeder to scan in batches
DWObject.IfDuplexEnabled = false;// Scan in Simplex mode (only 1 side of the page)
DWObject.PixelType = EnumDWT_PixelType.TWPT_GRAY; // Scan pages in GRAY
DWObject.Resolution = 200; // Scan pages in 200 DPI
DWObject.AcquireImage();// Start scanning
```

1. Go through each image by changing the property CurrentImageIndexInBuffer

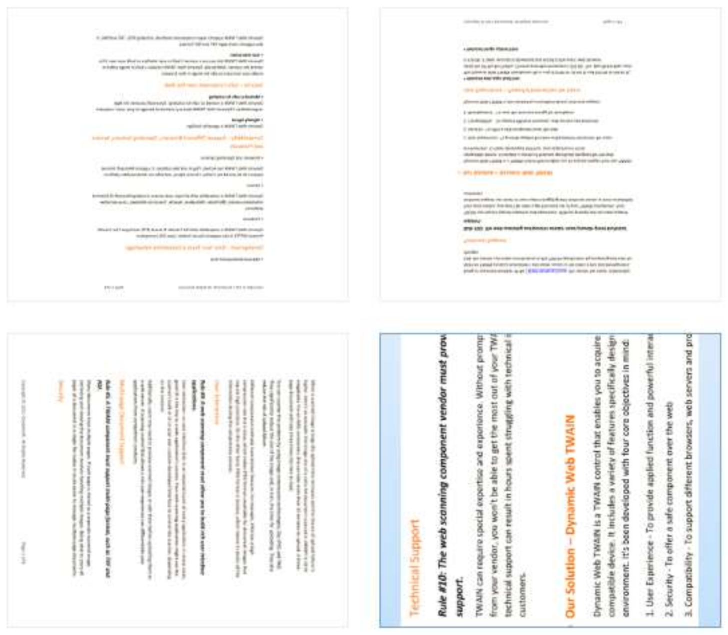
```
DWObject.CurrentImageIndexInBuffer = 2; // Show the 3rd image in the buffer
```

2. Show multiple images by changing the view mode (other than 1*1 or -1*-1) using SetViewMode()

```
DWObject.SetViewMode(2, 2); // Show images in buffer with 2 * 2 view
```

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

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



Also, you can remove/delete an image by its index or remove/delete selected or all images at once. The methods are **RemoveImage**, **RemoveAllSelectedImages**, **RemoveAllImages**

Using the Image Editor

What is the Image Editor

The Image Editor is a built-in feature of the SDK which can save you a lot of time in designing your scan page.

What can you do with the Image Editor

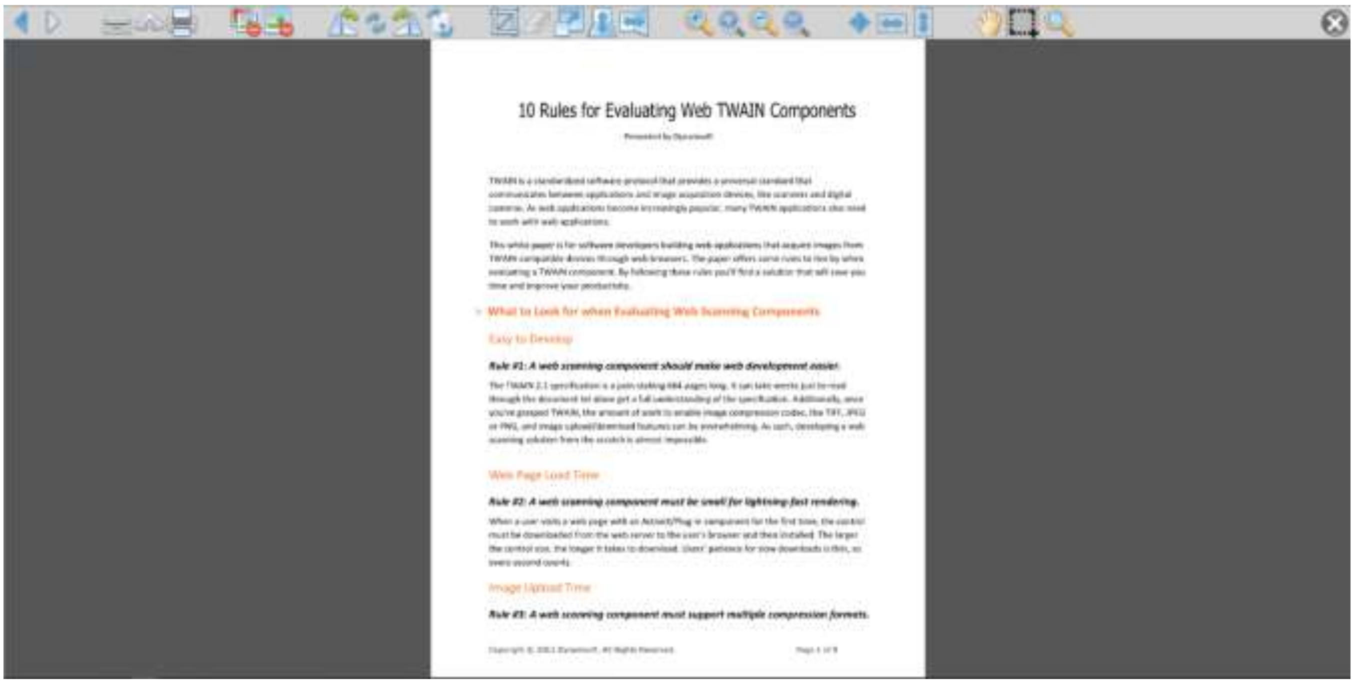
In Image Viewer, you can:

1. Go through all the images currently scanned or loaded
2. Scan, load, print, remove images, etc.
3. Edit an image in the following ways: rotate, mirror, crop, flip, change size, erase, etc.

Opening the Image Editor

You can use the method ShowImageEditor() to show the editor window:

```
DWObject.ShowImageEditor();
```



10 Rules for Evaluating Web TWAIN Components

Presented by Dynamsoft

[illegible]

[illegible]

What to Look for when Evaluating Web Scanning Components

Easy to Develop

Rule #1: A web scanning component should make web development easier.

[illegible]

Web Page Load Time

Rule #2: A web scanning component must be small for lightning fast rendering.

[illegible]

Image Upload Time

Rule #3: A web scanning component must support multiple compression formats.

[illegible]

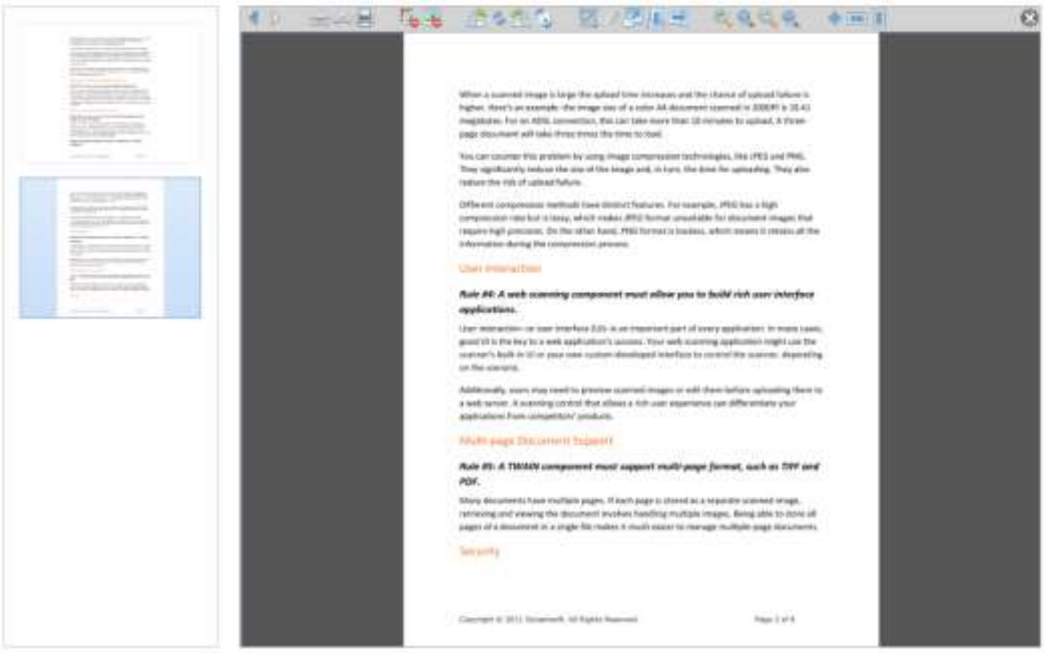
The image editor takes up the full screen. In v14.0, the editor can also be created within a DIV element on the page, this is discussed in the following section.

Creating A Thumbnails View

In version 14.0, you can use the built-in editor of Dynamic Web TWAIN together with the main image viewer to create a thumbnails view. The following code shows how it's done.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Thumbnails View</title>
    <script src="Resources/dynamsoft.webtwain.config.js"></script>
    <script src="Resources/dynamsoft.webtwain.initiate.js"></script>
</head>
<body>
    <div id="dwtcontrolContainer" style="float: left; margin-right:20px;"></div>
    <div id="dwtcontrolContainerLargeViewer" style="float: left;"></div>
    <script type="text/javascript">
        Dynamsoft.WebTwainEnv.RegisterEvent('OnWebTwainReady', Dynamsoft_OnReady);
        var DWObject;
        function Dynamsoft_OnReady() {
            DWObject = Dynamsoft.WebTwainEnv.GetWebTwain('dwtcontrolContainer');
            if (DWObject) {
                DWObject.Width = 200;
                DWObject.Height = 600;
                DWObject.ShowImageEditor("dwtcontrolContainerLargeViewer", 750, 600);
                DWObject.SetViewMode(1, 4);
            }
        }
    </script>
</body>
</html>
```

The following snip shows what it looks like



When a scanned image is large the upload time increases and the chance of upload failure is higher. Here's an example: the image size of a color A4 document scanned in 300DPI is [illegible] megabytes. For an ADSL connection, this can take more than 10 minutes to upload. A three-page document will take three times the time to load.

You can counter this problem by using image compression technologies, like JPEG and PNG. They significantly reduce the size of the image and, in turn, the time for uploading. They also reduce the risk of upload failure.

Different compression methods have distinct features. For example, JPEG has a high compression rate but is lossy, which makes JPEG format unsuitable for document images that require high precision. On the other hand, PNG format is lossless, which means it retains all the information during the compression process.

User Interaction

Rule #4: A web scanning component must allow you to build rich user interface applications.

User interaction - or user interface (UI) - is an important part of every application. In many cases, good UI is the key to a web application's success. Your web scanning application might use the scanner's built-in UI or your own custom developed interface to control the scanner, depending on the scenario.

Additionally, users may need to preview scanned images or edit them before uploading them to a web server. A scanning control that allows a rich user experience can differentiate your applications from competitors' products.

Multi-page Document Support

Rule #5: A TWAIN component must support multi-page format, such as TIFF and PDF.

Many documents have multiple pages. If each page is stored as a separate scanned image, retrieving and viewing the document involves handling multiple images. Being able to store all pages of a document in a single file makes it much easier to manage multiple-page documents.

Security

Copyright © 2011 Dynamsoft. All Rights Reserved [illegible]

As the snip shows, the built-in editor comes with many features itself. You can scan, load, remove images as well as edit them, zoom in/out to better view them, etc. directly in the editor by simply clicking a button. These buttons can be configured too in the file dynamsoft.webtwain.config.js.

In the dynamsoft.webtwain.config.js

The following are the default configurations, you can edit the visibility to show/hide certain features.

```
bShowAllButtons: true,
visibility: {
//only valid when bShowAllButtons is true, otherwise changing visibility does nothing
    'scan': true, 'load': true, 'print': true,
    'removeall': true, 'removeselected': true,
    'rotateleft': true, 'rotate': true, 'rotateright': true, 'deskew': true,
    'crop': true, 'erase': true, 'changeimagesize': true, 'flip': true, 'mirror': true,
    'zoomin': true, 'originalsize': true, 'zoomout': true, 'stretch': true,
    'fit': true, 'fitw': true, 'fith': true,
    'hand': true, 'rectselect': true, 'zoom': true
}
```

Adding/Removing Extra Dynamic Web TWAIN object(s)

To add/remove an extra Dynamic Web TWAIN object, you can use the method Dynamsoft.WebTwainEnv.CreateDWTObject(id, OnSuccessCallback, OnFailureCallback) and Dynamsoft.WebTwainEnv.DeleteDWTObject(id).

Add a Dynamic Web TWAIN object at runtime

To add a Dynamic Web TWAIN object on the web page, you need to first put a div element as the placeholder for this object:

HTML code:

```
<div id="dwtcontrolContainer2"></div>
```

The next step is to use the method Dynamsoft.WebTwainEnv.CreateDWTObject(id, OnSuccessCallback, OnFailureCallback) to create and initialize the Dynamic Web TWAIN object that will be embedded in the div with id 'dwtcontrolContainer2'.

JavaScript code:

```
var DWObject2;
Dynamsoft.WebTwainEnv.CreateDWTObject(
    "dwtcontrolContainer2",
    function (newDWObject) { DWObject2 = newDWObject; },
    function (errorString) { alert(errorString); }
);
```

NOTE:

If the div element with id "dwtcontrolContainer2" already exists in Dynamsoft.WebTwainEnv.Containers, you will get the following error:

"Duplicate ID detected for creating Dynamic Web TWAIN objects, please check and modify."

When this happens, please check if you've set "dwtcontrolContainer2" in dynamsoft.WebTwainEnv.Containers in dynamsoft.webtwain.config.js

```
Dynamsoft.WebTwainEnv.Containers = [{ContainerId:'dwtcontrolContainer2', Width: 270, Height: 350},];
```

If so, please change the id "dwtcontrolContainer2" in HTML and JavaScript and try again.

Okay, that's it. You can now add some style to the new Dynamic Web TWAIN object and manipulate it. For example:

```
DWObject2.Width = 580;
DWObject2.Height = 600;
DWObject2.SelectSource();
DWObject2.AcquireImage();
```

Remove a Dynamic Web TWAIN object at runtime

To remove a Dynamic Web TWAIN object from the web page. Just call the method Dynamsoft.WebTwainEnv.DeleteDWTObject(id) specifying the container id.

```
Dynamsoft.WebTwainEnv.DeleteDWTObject("dwtcontrolContainer2");
```

NOTE:

Dynamsoft.WebTwainEnv.Unload() can't release the Dynamic Web TWAIN objects generated by the method Dynamsoft.WebTwainEnv.CreateDWTObject. You can only use the method Dynamsoft.WebTwainEnv.DeleteDWTObject(id) to release that object.

Scanning Lots of Documents At a Time - Disk Caching

Sometimes, you may need to scan hundreds or even thousands of documents. In this case, the disk caching feature will come in handy. The related properties are **IfAllowLocalCache** and **BufferMemoryLimit**.

Although Dynamic Web TWAIN can run both as a 32bit application and a 64bit application, it's 32bit by default which means it can utilize no more than 2 GB of physical memory. However, the SDK deals with images which takes up a lot of space. For example, one A4 paper scanned in 300 DPI takes around 24MB in memory (DIB) and even if you can use 2GB to store images, you can store no more than 85 of them. This is why Dynamsoft added the disk caching feature to the SDK which, when enabled, caches most images temporarily on the disk while keeping a few active ones in the memory so that performance stays high.

The disk caching feature is enabled by default and can be disabled by setting **IfAllowLocalCache** to **false**.

Because the machine that the SDK runs on may have small or big RAM, we can set how much memory we want the SDK to use before images start to be cached. By default, 800MB is used. You can change it by the property **BufferMemoryLimit.**

NOTE:

1. *All cached data will be encrypted and can only be accessed by Dynamic Web TWAIN.*
   For ActiveX Edition:
   The cached data is stored in "C:\Users\{User Name}\AppData\LocalLow\Dynamsoft\cache ".
   For HTML5 Edition:
   It is stored at "C:\Windows\SysWOW64 {or system32}\Dynamsoft\DynamsoftService\cache"
2. *When the scanning page is closed, the cached data will be destroyed and removed from the disk automatically.*
3. *Although you can scan and load as many images as you want into Dynamic Web TWAIN control, you will need to process them in smaller volume instead of all at once. For example, you shouldn't upload too many images as one file as that could exceed the memory limit.*

Loading local image(s) into Dynamic Web TWAIN

Preparation

First, bear in mind that as a lightweight component running in web browsers, Dynamic Web TWAIN is only designed to deal with the most basic images in the following formats: BMP, JPEG, PNG, TIFF and PDF. We only guarantee that images generated by Dynamic Web TWAIN can be successfully loaded. If you are trying to load an image that was not generated by Dynamic Web TWAIN, you can check out this article.

Calling the methods

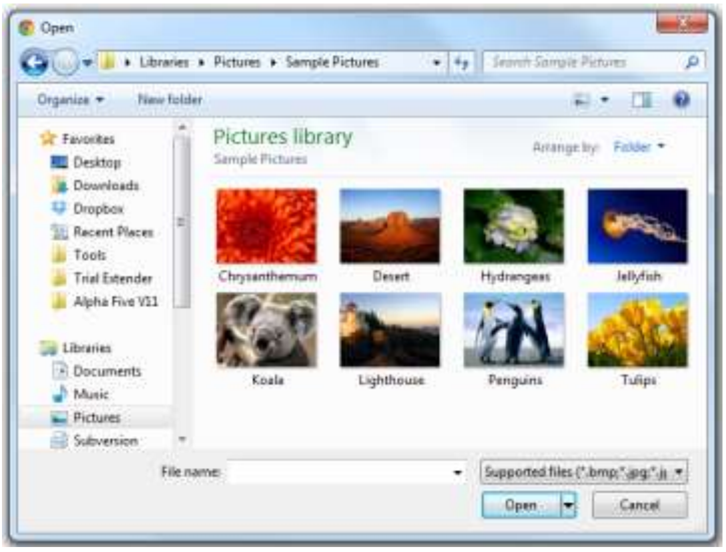
With Dynamic Web TWAIN, you can load local images with the methods **LoadImage**() or **LoadImageEx**(). Below is a simple code snippet:

```
DWObject.LoadImage("C:\\WebTWAIN\\Images\\ImageData.jpg", optionalAsyncSuccessFunc,
optionalAsyncFailureFunc);
DWObject.LoadImageEx("C:\\WebTWAIN\\Images\\ImageData.jpg", EnumDWT_ImageType.IT_JPG,
optionalAsyncSuccessFunc, optionalAsyncFailureFunc); // ImageType: JPG

//Callback functions for async APIs
function optionalAsyncSuccessFunc() {
    console.log('successful');
}
function optionalAsyncFailureFunc(errorCode, errorString) {
    alert(errorString);
}
```

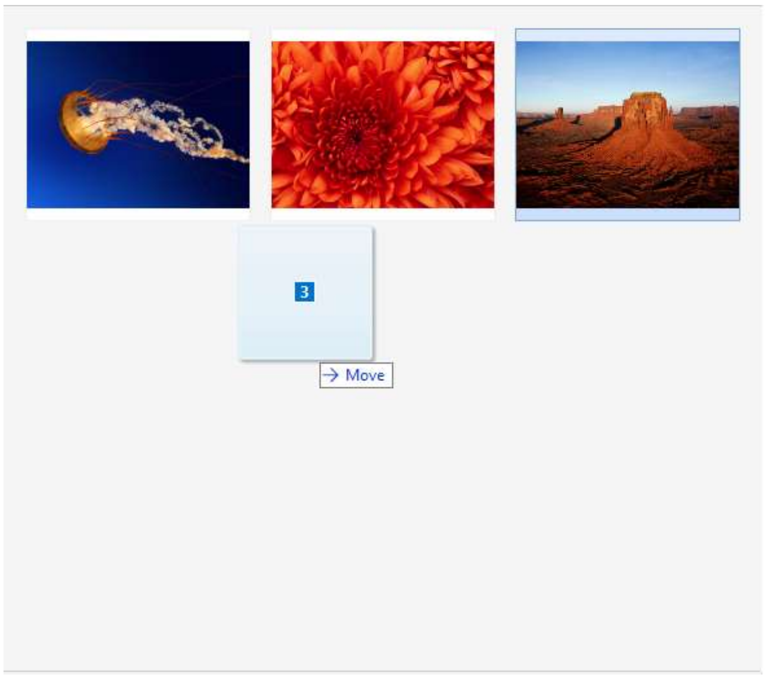
As you can see, you need to provide the complete file path in order to load an image. This is somewhat clumsy especially when you need to load more than one image. But no worries, Dynamic Web TWAIN can open a "Select File..." dialog for you to locate the image(s) you want to load. And like other properties and methods, it's very easy to use. Below is a code snippet:

```
DWObject.IfShowFileDialog = true;
DWObject.LoadImageEx("", EnumDWT_ImageType.IT_ALL); //ALL (BMP, JPG, PNG, PDF, TIFF)
```

Please note that the second parameter "ImageType" in the method **LoadImageEx**() would determine the file filter in the "Select File..." dialog.



Starting from v14.0, you can also just drag and drop images onto the Dynamic Web TWAIN viewer to load them.



Saving image(s) locally

Preparation

Dynamic Web TWAIN can save all scanned or loaded images locally in the following formats: BMP, JPEG, PNG, TIFF (single-page or multi-page) and PDF (single-page or multi-page).

Calling the methods

With Dynamic Web TWAIN, you can choose one of the following methods to save an image or images:

Format	Methods

Single-Page File	SaveAsBMP()
	SaveAsJPEG()
	SaveAsPDF()
	SaveAsPNG()
	SaveAsTIFF()

Multi-page PDF	SaveSelectedImagesAsMultiPagePDF()
	SaveAllAsPDF()

Multi-page TIFF	SaveAllAsMultiPageTIFF()
	SaveSelectedImagesAsMultiPageTIFF()

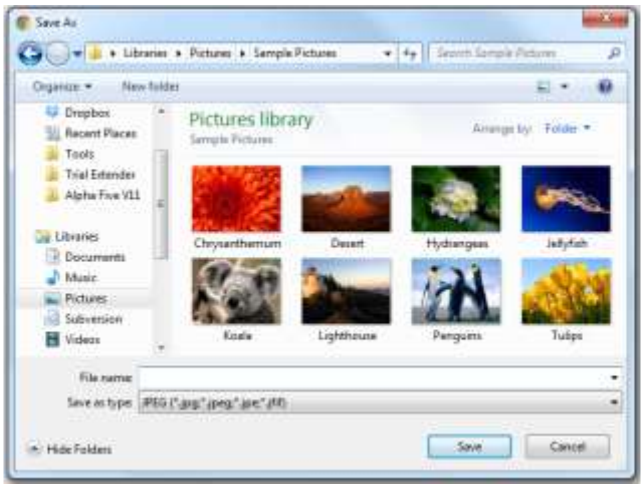
Code snippet:

```
//Use it synchronously
DWObject.SaveAsJPEG("C:\\WebTWAIN\\Images\\ImageData.jpg", 0);

//Use it asynchronously
DWObject.SaveAllAsPDF("C:\\WebTWAIN\\Images\\ImageData.pdf", optionalAsyncSuccessFunc,
optionalAsyncFailureFunc);

//Callback functions for Async APIs
function optionalAsyncSuccessFunc() {
    console.log('successful');
}
function optionalAsyncFailureFunc(errorCode, errorString) {
    alert(errorString);
}
```

From the above code, you can see that you need to provide the complete file path to save an image locally, which is sometimes inconvenient. But no worries, just like loading an image, Dynamic Web TWAIN can also open a "Save As..." dialog for you to locate the path that you want to save the image(s) to. Below is a code snippet:

```
DWObject.IfShowFileDialog = true;
DWObject.SaveAsJPEG("",0);
```

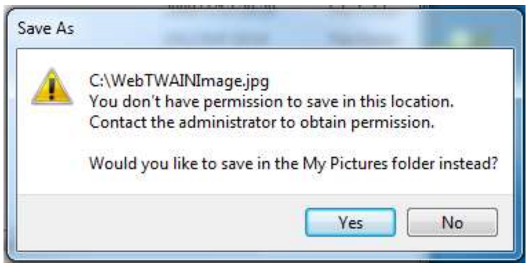
This will bring up this dialog box with the "Save as type" specified by the method you use:



File name:
Save as type: JPEG (*.jpg;*.jpeg;*.jpe;*.jfif)

NOTE:

On Windows 7 and above, Microsoft has strengthened security which means you can only save images to certain places where you have the write permission. If you try to save to other places, you will get the below error message. You can then save to a different directory or first obtain permission for that directory.



Save As

C:\WebTWAINImage.jpg
You don't have permission to save in this location.
Contact the administrator to obtain permission.

Would you like to save in the My Pictures folder instead?

Yes No

Uploading image(s) to the web server

Preparation

Before we upload the image(s), we need to set the server IP/name, set the port number, as well as define the path for the action page. An action page refers to the target script that receives the HTTP Post request containing the image data and handles all the server-side operation like saving the data on the hard disk or database, etc. Here is an example:

```
var strHTTPServer = location.hostname;
DWObject.HTTPPort = location.port == "" ? 80 : location.port;
var CurrentPathName = unescape(location.pathname);
var CurrentPath = CurrentPathName.substring(0, CurrentPathName.lastIndexOf("/") + 1);
var strActionPage = CurrentPath + "actionPage.aspx";
var uploadfilename = "TestImage.pdf";
```

In the code snippet

strHTTPServer is used to store the server name which specifies which server the image(s) will be uploaded to. You can also use the server's IP for the same purpose. If you want to upload image(s) to the same server as the current page, we suggest you use "location.hostname**"** to get the hostname at runtime.

The property **HTTPPort** specifies the HTTP port to be used for the upload. Normally, port 80 is for HTTP, port 443 is for HTTPS, etc. If you are not sure about the port number, it's recommended that you use "location.port == "" ? 80 : location.port" to get the current port number at runtime.

CurrentPathName and **CurrentPath** are used to build the relative path of the action page.

strActionPage stores the relative path of the action page.

uploadfilename stores the file name for the uploaded image(s). You should change the extension of the name accordingly.

NOTE:

In version 10.0 and above, we are using the browser as the upload agent. Due to browser security restrictions, client-side scripts (e.g., JavaScript) are not allowed to make requests to another domain. Therefore, when you try to upload an image to a server with a different domain, subdomain, port, or protocol, you need to configure your server to allow such requests by adding one HTTP Response Header, namely:

```
Access-Control-Allow-Origin: *
// the asterisk wild-card permits scripts hosted on any site to load your resources
```

Take IIS 7 for example. What you need to do is merge the following lines into the web.config file at the root of your application / site:

```
<?xml version="1.0" encoding="utf-8"?>
  <configuration>
    <system.webServer>
      <httpProtocol>
        <customHeaders>
          <add name="Access-Control-Allow-Origin" value="*" />
                <add name="Access-Control-Allow-Methods" 
value="OPTIONS,POST,GET,PUT"/>
          <add name="Access-Control-Allow-Headers" value="x-requested-with"/>
          <add name="Access-Control-Allow-Credentials" value="true" />
        </customHeaders>
      </httpProtocol>
    </system.webServer>
  </configuration>
```

If you don't have a web.config file already, just create a new file called "web.config" and add the snippet above.

Calling the methods

Now, we can call one of the HTTP upload methods to upload the image(s). We have 8 methods:

Format	Method
All kinds of files	HTTPUploadThroughPostDirectly()

All supported image formats	HTTPUpload()
	HTTPUploadThroughPost()
	HTTPUploadThroughPostEx()

Multi-page PDF	HTTPUploadAllThroughPostAsPDF()
	HTTPUploadThroughPostAsMultiPagePDF()

Multi-page TIFF	HTTPUploadAllThroughPostAsMultiPageTIFF()
	HTTPUploadThroughPostAsMultiPageTIFF()

Let's take the method **HTTPUploadAllThroughPostAsPDF**() for example:

```
DWObject.HTTPUploadAllThroughPostAsPDF(
    strHTTPServer,
    strActionPage,
    uploadfilename,
    OnHttpUploadSuccess, OnHttpUploadFailure
);
```

With this method, all the images in the Dynamic Web TWAIN control will be sent to the web server as one multi-page PDF file.

In the above code, the parameters **OnHttpUploadSuccess** and **OnHttpUploadFailure** are optional callback functions. If they are present, the method is asynchronous, otherwise, the method is synchronous. It's recommended that you use the methods asynchronously to avoid possible browser hanging.

The following is a simple implementation of these two functions:

```
function OnHttpUploadSuccess() {
    console.log('successful');
}
function OnHttpUploadFailure(errorCode, errorString, sHttpResponse) {
    alert(errorString + sHttpResponse);
}
```

If you want to upload one image as a single-page file, you can use **HTTPUploadThroughPost** or **HTTPUploadThroughPostEx**.

If you want to upload selected images as a multi-page file, you can use **HTTPUploadThroughPostAsMultiPagePDF** or **HTTPUploadThroughPostAsMultiPageTIFF**.

Action Page

The HTTP upload method(s) makes a standard HTTP post request to the action page on the server. The request contains the image data, image name, etc. In the action page, you can process the image data according to your requirements. Technically you can write the action page in any server-side language (C#, VB, PHP, Java, etc...).

Here is an example in C#:

This action page retrieves the image data from the current http request object and saves it as a local file on the server.

```
HttpFileCollection files = HttpContext.Current.Request.Files;
HttpPostedFile uploadfile = files["RemoteFile"];
uploadfile.SaveAs(System.Web.HttpContext.Current.Request.MapPath(".") + "/" +
uploadfile.FileName);
```

Note:

*Please note that '**RemoteFile**' is the default name/key for the uploaded image data. If necessary, you can change it using the property **HttpFieldNameOfUploadedImage**.*

To do the same thing in PHP:

```
$fileTempName = $_FILES['RemoteFile']['tmp_name'];
$fileSize = $_FILES['RemoteFile']['size'];
$fileName = $_FILES['RemoteFile']['name'];
move_uploaded_file($fileTempName, $fileName) ;
```

Uploading to FTP

Besides the HTTP upload methods, you can also use the FTP Upload methods to update image(s) to your FTP web server. The available APIs are:

Format	Method
All files	FTPUploadDirectly ()

All supported formats	FTPUpload()
	FTPUploadEx()

Multi-page PDF	FTPUploadAllAsPDF()
	FTPUploadAsMultiPagePDF()

Multi-page TIFF	FTPUploadAllAsMultiPageTIFF()
	FTPUploadAsMultiPageTIFF()

Code Snippet

```
DWObject.FTPUserName = 'test';
DWObject.FTPPort = 21;
DWObject.FTPPassword = 'test';
DWObject.FTPUploadAllAsPDF(
    '192.168.8.222',
    'test.pdf',
    OnFtpUploadSuccess,
    OnFtpUploadFailure
);
```

Uploading image(s) to a Database

Dynamic Web TWAIN doesn’t save/upload the image(s) to your database directly. Instead, we can upload the image data to the action page first and then use the action page to store the image data in the database.

If you are not sure how to upload the image data to the server, please refer to the previous section Uploading image(s) to the web server.

Different database systems have different data types for image data. We normally use ***BLOB*** or ***varbinary*** in SQL Server, ***Long raw*** or ***BLOB*** in Oracle, ***BLOB*** in MySQL, etc.

Here is an example in C# with SQL Server:

```
int iFileLength;
HttpFileCollection files = HttpContext.Current.Request.Files;
HttpPostedFile uploadfile = files["RemoteFile"];
String strImageName = uploadfile.FileName;

iFileLength = uploadfile.ContentLength;
Byte[] inputBuffer = new Byte[iFileLength];
System.IO.Stream inputStream;
inputStream = uploadfile.InputStream;
inputStream.Read(inputBuffer,0,iFileLength);

// add code to connect to database
String SqlCmdText = "INSERT INTO tblImage (strImageName,imgImageData) VALUES
(@ImageName,@Image)";
System.Data.SqlClient.SqlCommand sqlCmdObj = new
System.Data.SqlClient.SqlCommand(SqlCmdText, sqlConnection);

sqlCmdObj.Parameters.Add("@Image",System.Data.SqlDbType.Binary,iFileLength).Value =
inputBuffer;
sqlCmdObj.Parameters.Add("@ImageName",System.Data.SqlDbType.VarChar,255).Value =
strImageName;

sqlConnection.Open();
sqlCmdObj.ExecuteNonQuery();
sqlConnection.Close();
```

In the code snippet, we get the file object from the current http request and write the image data to a byte array. In the SQL statement, we pass the byte array to the database as *System.Data.SqlDbType.Binary* and store the data in a BL field called "imgImageData".

Uploading image(s) with extra data

If you are not sure how to upload the image data to the server, please refer to the previous topic "Uploading image(s) to the web server".

Sometimes we need to pass more information to the server. For example, document type, employee ID, document description, etc. Since we don't have any options to pass extra data in the HTTP upload method, we need to use a method called **SetHTTPFormField**.

SetHTTPFormField(string sFieldName, string sFieldValue)

- string sFieldName: specifies the name of a text field in the web form.
- string sFieldValue: specifies the value of a text field in the web form.

We need to use this method before the HTTP Upload method. Here is an example:

```
DWObject.ClearAllHTTPFormField(); // Clear all fields first
DWObject.SetHTTPFormField("EmployeeID", "2012000054");
DWObject.SetHTTPFormField("DocumentType", "Invoice");
DWObject.SetHTTPFormField("DocumentDesc", "This is an invoice from ...");
```

In the action page, you can retrieve the data from the request object by the field names. For example:

```
String EmployeeID = HttpContext.Current.Request.Form["EmployeeID"];
```

Downloading image(s) from the web

You can use the method **HTTPDownload**() or **HTTPDownloadEx**() to download an image from the web server into Dynamic Web TWAIN.

```
DWObject.HTTPDownload("www.dynamsoft.com", "/images/dwt-logo.png",
optionalAsyncSuccessFunc, optionalAsyncFailureFunc);

//Callback functions for async APIs
function optionalAsyncSuccessFunc() {
    console.log('successful');
}

function optionalAsyncFailureFunc(errorCode, errorString) {
    alert(errorString);
}
```

This is especially useful when you want to review an image created and uploaded by Dynamic Web TWAIN. Even when the image data is stored in the database, you can write an action page to pull the data from the database and get it downloaded (in this case, you need to use the method **HTTPDownloadEx** because the image format needs to be specified explicitly). Besides the HTTP download methods, you can also use the FTP download methods to download image(s) from a FTP server. Available methods are **FTPDownload**, **FTPDownloadEx**, etc.

NOTE:

As mentioned earlier in the section Uploading image(s) to the web server, special configuration has to be made on the server to overcome browser security restrictions. When you try to download an image from a server with a different domain, subdomain, port, or protocol, you need to configure your server to allow such requests by adding one HTTP Response Header, namely:

```
Access-Control-Allow-Origin: *
```

Take IIS 7 for example, what you need to do is merge the following lines into the web.config file at the root of your application / site:

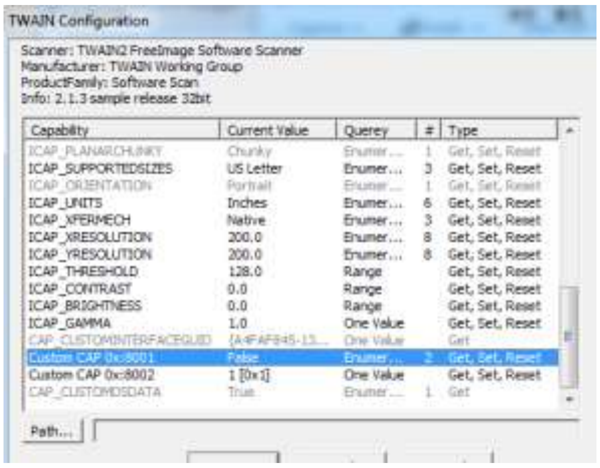
```
<?xml version="1.0" encoding="utf-8"?>
  <configuration>
    <system.webServer>
      <httpProtocol>
        <customHeaders>
          <add name="Access-Control-Allow-Origin" value="*" />
                <add name="Access-Control-Allow-Methods"
value="OPTIONS,POST,GET,PUT"/>
          <add name="Access-Control-Allow-Headers" value="x-requested-with"/>
          <add name="Access-Control-Allow-Credentials" value="true" />
        </customHeaders>
      </httpProtocol>
    </system.webServer>
  </configuration>
```

If you don't have a web.config file already, just create a new file called "web.config" and add the snippet above.

Using Custom capabilities

To use a custom capability, you need to know what code represents this capability. To do this, you can follow the steps below:

1. Install the TWAIN sample application
   - 32-bit: http://www.dynamsoft.com/download/support/twainapp.win32.installer.msi
   - 64-bit: http://www.dynamsoft.com/download/support/twainapp.win64.installer.msi
2. Use the TWAIN Sample App to open the source and check what the code of the capability is.


TWAIN Configuration

Scanner: TWAIN2 FreeImage Software Scanner
Manufacturer: TWAIN Working Group
ProductFamily: Software Scan
Info: 2.1.3 sample release 32bit

Capability	Current Value	Querey	#	Type
ICAP_PLANARCHUNKY	Chunky	Enumer...	1	Get, Set, Reset
ICAP_SUPPORTEDSIZES	US Letter	Enumer...	3	Get, Set, Reset
ICAP_ORIENTATION	Portrait	Enumer...	1	Get, Set, Reset
ICAP_UNITS	Inches	Enumer...	6	Get, Set, Reset
ICAP_XFERMECH	Native	Enumer...	3	Get, Set, Reset
ICAP_XRESOLUTION	200.0	Enumer...	8	Get, Set, Reset
ICAP_YRESOLUTION	200.0	Enumer...	8	Get, Set, Reset
ICAP_THRESHOLD	128.0	Range		Get, Set, Reset
ICAP_CONTRAST	0.0	Range		Get, Set, Reset
ICAP_BRIGHTNESS	0.0	Range		Get, Set, Reset
ICAP_GAMMA	1.0	One Value		Get, Set, Reset
CAP_CUSTOMINTERFACEGUID	{A4FAF845-13...	One Value		Get
Custom CAP 0x:8001	False	Enumer...	2	Get, Set, Reset
Custom CAP 0x:8002	1 [0x1]	One Value		Get, Set, Reset
CAP_CUSTOMDSDATA	True	Enumer...	1	Get

3. In the above example, code "0x8001" is the hexadecimal value for the highlighted custom capability. Now, we can use the code to negotiate the capability with the scanner driver:

```
DWObject.Capability = 0x8001;
DWObject.CapType = EnumDWT_CapType.TWON_ONEVALUE; // TWON_ONEVALUE
DWObject.CapValue = 1;
if (DWObject.CapSet())
    alert("Successful");
else
    alert("Source doesn't support this capability");
```

Please refer to How to perform Capability Negotiation for more details.

License Verification

Basic Information

Since version 9.0, Dynamic Web TWAIN has been using the property ProductKey for license verification at runtime. The property accepts a series of alphanumeric code as the product key which is generated based on the ***license***(s) you own. All editions share the same authentication mechanism.

Note:

1. *One product key can be generated from one or many license(s), this is done by Dynamsoft*
2. *The product key represents the encrypted license(s); every product key is unique*
3. *The product key can also be bound to a specific domain (since version 11)*

Using the Product Key

Set it during initialization (recommended)

Set ProductKey in the file *`Dynamsoft.webtwain.config.js`*

```
Dynamsoft.WebTwainEnv.ProductKey = 't0068MgAA…';
```

Set it when necessary (not recommended)

Set ProductKey in your own code before calling AcquireImage() method

```
function AcquireImage() {
    DWObject.SelectSource();
    DWObject.OpenSource();
    DWObject.IfShowUI = false;
    DWObject.ProductKey = 't0068Mg…';
    DWObject.AcquireImage();
}
```

Multiple Product Keys

If you have multiple product keys generated from multiple serial numbers, you can combine all of them and assign them to the ProductKey property. You will need to separate the keys by semi-colons (;). For example

```
Dynamsoft.WebTwainEnv.ProductKey = 't0068MgAA…;t006…;t00…';
```

Upgrading from Previous Versions

Please refer to the Knowledge Base article below:

http://developer.dynamsoft.com/dwt/kb/2575

Troubleshooting

Installing the Virtual Scanner for testing

If you don’t have an actual scanner available for testing, you can download and install a virtual scanner (a scanner simulator), which was developed by the TWAIN Working Group, to test the basic scanning features of Dynamic Web TWAIN.

Download 32 bit virtual scanner   Download 64 bit virtual scanner

Is my scanner driver TWAIN|ICA|SANE compliant

On Windows:

If you are not sure whether your scanner is TWAIN compliant, you can use TWACKER, a tool developed by the TWAIN Working Group, to verify it. TWACKER can be downloaded here: TWACKER 32 bit   TWACKER 64 bit

After installation:

- Launch the program.
- Click menu File->Select Source and select your device in the ‘Select Source’ list.
    - If your device is not listed, please check if the driver is installed. Or, you can try running TWACKER as "Admin" since you may not have permission to access the data source.
- Click menu File->Acquire to do scanning.

If the scanning is successful without any errors, then your device should be TWAIN compliant.

If it fails, you may have to search online or contact the device vendor to obtain a TWAIN driver.

On macOS:

Please check out https://developer.dynamsoft.com/dwt/kb/trouble-shooting-for-end-users/how-to-test-if-your-scanner-supports-native-scanning-on-mac-os-x

On Linux:

Please check out https://developer.dynamsoft.com/dwt/kb/trouble-shooting-for-end-users/how-to-check-if-your-scanner-is-sane-compatible

Why is my scanner not shown or not responding in the browser

Please check out http://developer.dynamsoft.com/dwt/kb/2541.

[ActiveX] Why does Dynamic Web TWAIN fail to upload my documents?

When you use forms authentication in your web application, you might fail to upload documents to the server using **Dynamic Web TWAIN in IE 6-9.** This happens because Dynamic Web TWAIN still works as an ActiveX in IE 6-9 and it performs the upload as an independent user-agent, one that might not have permission to get the authentication cookie during the upload.

The root cause of the issue is the missing cookie for the authentication. So, the solution is to locate the cookie and bind it to the upload request. To do this, Dynamsoft has provided an API called **SetCookie**. The following are the steps, (take ASP.NET (C#) for example):

1. Print the cookie out explicitly in the page and assign it to a JavaScript variable:
   var cookie = "<%=Request.Headers["Cookie"] %>";
2. Set the cookie at runtime before you call any HTTP Upload method:
   DWObject.SetCookie(cookie);
   DWObject.HTTPUpload…

That's it.

NOTE:

The above solution is very straightforward but it exposes the authentication cookie. If this is not an acceptable solution, we also offer a workaround using AJAX. Check out https://developer.dynamsoft.com/dwt/kb/2841.

More Troubleshooting Topics

For more troubleshooting topics, please check out:
http://developer.dynamsoft.com/dwt/kb

Useful Resources

Online Demo Site

https://demo.dynamsoft.com/dwt/online_demo_scan.aspx

Knowledge Base

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

API Documentation

http://developer.dynamsoft.com/dwt/api-reference

Contact Us

Email: support@dynamsoft.com

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

Telephone: +**1 604.605.5491** | +**1 877.605.5491** (Toll-Free)

FAX: 1-866-410-8856

Appendix A: How to Perform Capability Negotiation

This article describes how to negotiate capabilities with different capability container types using Dynamic Web TWAIN. For an advanced sample, please check out this demo page.

Description

Capabilities represent the features that a specified TWAIN source (e.g. a scanner) provides. In order to make full use of such a source/device, TWAIN applications need to perform the operation called capability negotiation. With the negotiation, TWAIN applications can understand the source and then guide it to provide the images they would like to receive from it.

Capability Containers

Capabilities exist in many varieties but all have a Default Value, Current Value, and may have other values available that can be supported if selected. To help categorize the supported values into clear structures, TWAIN defines four types of containers for capabilities which are

Container Data Structure	Type of Contents
TW_ONEVALUE	A single value whose current and default values are coincident. The range of available values for this type of capability is simply this single value. For example, a capability that indicates the presence of a document feeder could be of this type.
TW_ARRAY	An array of values that describes the current logical item. The available values may be a larger array of values. For example, a list of the supported capabilities list returned by the CAP_SUPPORTEDCAPS capability, would use this type of container.
TW_RANGE	Many capabilities allow users to select their current value from a range of regularly spaced values. The capability can specify the minimum and maximum acceptable values and the incremental step size between values. For example, resolution might be supported from 100 to 600 in steps of 50 (100, 150, 200, ..., 550, 600).
TW_ENUMERATION	This is the most general type because it defines a list of values from which the Current Value can be chosen. The values do not progress uniformly through a range and there is not a consistent step size between the values. For example, if a Source’s resolution options did not occur in even step sizes then an enumeration would be used (for example, 150, 400, and 600).

What is involved

To perform capability negotiation, you basically do two things

- Get a Capability. This is used to ask the source about a specified capability and get its type, value, etc.
- Set a Capability. This is generally used to request the source to set/change the value of a capability.

Now we'll talk about how to perform capability negotiation

Before doing any capability negotiation, please keep in mind that it can only be done when the source is open (**DataSourceStatus** is 1). Basically you need to use the methods **SelectSource**() and **OpenSource**() to select a TWAIN source and get it ready for the negotiation.

Get

To get information about a capability, you can simply use the following code snippet

```
DWObject.OpenSource();
DWObject.Capability = EnumDWT_Cap.***;
DWObject.CapGet();
var tempValue = '';
DWObject.CapValueType > 8 ?
/*STR*/tempValue = DWObject.CapValueString
    :
/*NUM*/tempValue = DWObject.CapValue;
/*
* Special for BOOL
*/
if (DWObject.CapValueType == EnumDWT_CapValueType.TWTY_BOOL) {
    DynamsoftCapabilityNegotiation.CurrentCapabilityHasBoolValue = true;
    tempValue == 0 ? tempValue = 'FALSE' : tempValue = 'TRUE';
}
alert('The type of the capability is ' + DWObject.CapType); /*More info*/
alert('The value of the capability is ' + tempValue);
```

Set

Please NOTE that the container type and available values for a capability is generally dictated by the TWAIN source vendor. When you try to set a capability, you are basically just trying to change it so that it uses a different but available value. Therefore, we recommend that you try to 'get' the capability before you set it.

TW_ONEVALUE

To set a capability with TW_ONEVALUE container, the following steps are needed:

- Set the **Capability** property to the capability to be negotiated;
- Set the **CapType** property to TW_ONEVALUE (`EnumDWT_CapType.TWON_ONEVALUE(5)`) (if you did CapGet() first, this step can be skipped);
- Input the value that you'd like to set to this capability. Please NOTE that you can only set certain values (for example, only 'true' and 'false' are allowed for a capability with the CapValueType of `TWTY_BOOL);`
- If the data type (**CapValueType**) of the capability is ***string*** (`EnumDWT_CapValueType.TWTY_STR32/64/128/255`), set the value using the **CapValueString** property. Otherwise, set the value using the **CapValue** property;
- Call **CapSet**() to actually set the value.

```
DWObject.OpenSource();
DWObject.Capability = EnumDWT_Cap.***; /*Make sure this Capability is TW_ONEVALUE*/
DWObject.CapGet(); /*Recommended*/
DWObject.CapType = EnumDWT_TWAINCONTAINERTYPE.TWON_ONEVALUE;
DWObject.CapValueType > 8 ?
/*STR*/DWObject.CapValue = someStringValue;
:
/*NUM*/DWObject.CapValue = someNonStringValue;
DWObject.CapSet();
```

TW_ARRAY

TW_ARRAY capability container type can be used to set or return a group of associated individual values.

To set a capability with TW_ARRAY container, the following steps are needed:

- Set the **Capability** property to the capability to be negotiated;
- Set the **CapType** property to TWON_ARRAY (`EnumDWT_CapType.TWON_ARRAY (3)`) (if you did CapGet() first, this step can be skipped);
- Set the **CapNumItems** property to indicate how many items are in the array;
- If the data type (**CapValueType**) of the capability is ***string*** (`EnumDWT_CapValueType.TWTY_STR32/64/128/255`), set the value using the **CapValueString** property. Otherwise, set the value using the **CapValue** property;
- Call **CapSet**() to actually set the value.

```
DWObject.OpenSource();
DWObject.Capability = EnumDWT_Cap.***; /*Make sure this Capability is TWON_ARRAY*/
DWObject.CapGet(); /*Recommended*/
DWObject.CapType = EnumDWT_TWAINCONTAINERTYPE.TWON_ARRAY;
DWObject.CapNumItems = *;
if (DWObject.CapValueType > 8) {
    /*STR*/
    DWObject.SetCapItemsString(0, someStringValue);
    DWObject.SetCapItemsString(1, someStringValue);
    …
} else {
    /*NUM*/
    DWObject.SetCapItems(0, someNonStringValue);
    DWObject.SetCapItems(1, someNonStringValue);
    …
}
DWObject.CapSet();
```

TW_RANGE

TW_RANGE capability container type can be used to set or return a range of individual values describing a capability. The values are uniformly distributed between a minimum and a maximum value. The step size between every two values is constant.

To set a capability with TW_RANGE container, the following steps are needed:

- Set the **Capability** property to the capability to be negotiated;
- Set the **CapType** property to TWON_RANGE (`EnumDWT_CapType.TWON_RANGE (6)`) (if you did CapGet() first, this step can be skipped);
- Set the **CapMinValue**, **CapMaxValue**, **CapStepSize** and **CapCurrentValue** properties;
- Call **CapSet**() to actually set the value.

```
DWObject.OpenSource();
DWObject.Capability = EnumDWT_Cap.***; /*Make sure this Capability is TWON_RANGE*/
DWObject.CapGet(); /*Recommended*/
DWObject.CapType = EnumDWT_TWAINCONTAINERTYPE.TWON_RANGE;
DWObject.CapMinValue = 80;
DWObject.CapMaxValue = 200;
DWObject.CapStepSize = 20;
DWObject.CapCurrentValue = 100;
DWObject.CapSet();
```

TW_ENUMERATION

TW_ENUMERATION capability container type can be used to set or return a group of associated individual values describing a capability. The values are ordered from the lowest to highest values, but the step size between every two values is probably not uniform.

To set a capability with TW_ENUMERATION container, the following steps are needed:

- Set the **Capability** property to the capability to be negotiated;
- Set the **CapType** property to TWON_ENUMERATION (`EnumDWT_CapType.TWON_ENUMERATION (4)`) (if you did **CapGet**() first, this step can be skipped);
- Set the **CapNumItems** property to indicate how many items are in the array;
- If the data type (**CapValueType**) of the capability is ***string*** (`EnumDWT_CapValueType.TWTY_STR32/64/128/255`), set the value using the **CapValueString** property. Otherwise, set the value using the **CapValue** property;
- Set the **CapCurrentIndex** to indicate the index of the current value;
- Call **CapSet**() to actually set the value.

```
DWObject.OpenSource();
DWObject.Capability = EnumDWT_Cap.***; /*Make sure this Capability is
TWON_ENUMERATION*/
DWObject.CapGet(); /*Recommended*/
DWObject.CapType = EnumDWT_TWAINCONTAINERTYPE.TWON_ENUMERATION;
DWObject.CapNumItems = *;
if(DWObject.CapValueType > 8 ){
    /*STR*/
    DWObject. SetCapItemsString (0, someStringValue);
    DWObject. SetCapItemsString (1, someStringValue);
…
} else {
/*NUM*/
    DWObject. SetCapItems(0, someNonStringValue);
    DWObject. SetCapItems(1, someNonStringValue);
…
}
DWObject.CapCurrentIndex = 1;
DWObject.CapSet();
```