WebTwainX User Manual
(Version 1.3)

Introduction

Ciansoft WebTwainX is an ActiveX control for scanning or capturing images from TWAIN-compliant devices in a browser based web application. The images can then be uploaded to a web server. WebTwainX is provided as a complete solution including the necessary server components for receiving and saving the uploaded files. This document contains comprehensive instructions on installing and using WebTwainX.

For further information, visit our website: www.ciansoft.com.

Alternatively, contact us by email, we will be pleased to answer your questions: info@ciansoft.com.

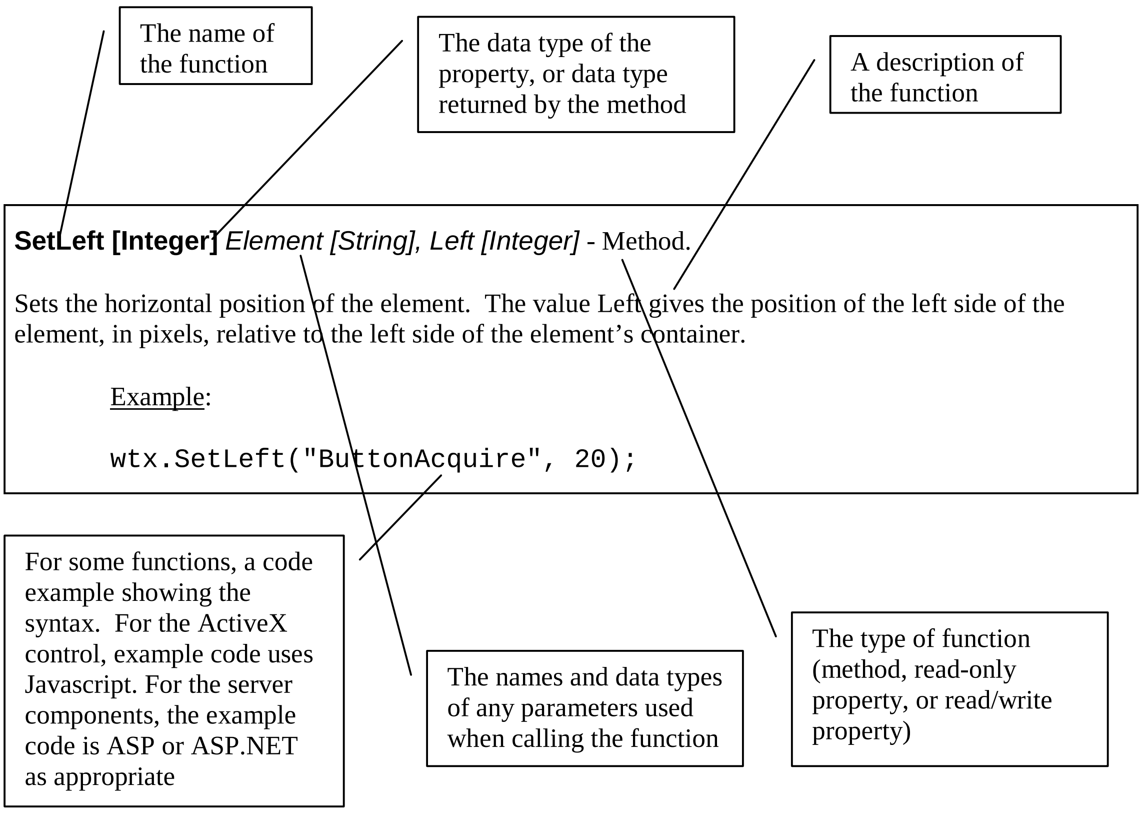
How to Use This Manual

WebTwainX is designed to be very simple to get working, with almost no coding required for a basic installation. We recommend that the section on Getting Started is read first which provides a step by step guide to setting up a simple web application to acquire images and upload to a server.

After that, the next section on Clientside Control gives a more detailed description of the features available in the user interface.

It will only be necessary to consult the remaining sections of this document if the installation is to be customised in some way, either to modify the way in which files are saved on the server, or to change the appearance and available functionality in the interface.

All the functions available, either in the ActiveX control or in the server side components for processing uploads, are described using the following nomenclature:


The name of the function
The data type of the property, or data type returned by the method
A description of the function
For some functions, a code example showing the syntax. For the ActiveX control, example code uses Javascript. For the server components, the example code is ASP or ASP.NET as appropriate
The names and data types of any parameters used when calling the function
The type of function (method, read-only property, or read/write property)

SetLeft [Integer] *Element [String], Left [Integer]* - Method.

Sets the horizontal position of the element. The value Left gives the position of the left side of the element, in pixels, relative to the left side of the element's container.

Example:

```
wtx.SetLeft("ButtonAcquire", 20);
```

TABLE OF CONTENTS

1. Getting Started

1.1. Files Supplied

The trial version of WebTwainX is distributed as an executable installer (WebTwainXTrial.exe). When this installer is run, the following files are copied to the Program Files folder under the path Ciansoft\WebTwainXTrial:

WebTwainXTrial.ocx:	The ActiveX control
WebTwainXTrial.cab:	The ActiveX control packaged in a digitally signed CAB file for use on a server
WebTwainXUploadASP.dll:	Server component for receiving uploaded files
Ciansoft.WebTwainXUploadNet.dll:	Server component for receiving uploaded files (for use with ASP.NET)
webtwainxtrial.lpk:	Licence package file for use with a web application
WebTwainXTrial.lic:	Licence file for use in a development environment such as Visual Basic
WebTwainX User Manual.pdf:	This manual
webtwainxdemo.htm:	Example web page
uploadsave.asp:	Example server script (ASP) to save uploaded file
uploadsave.aspx:	Example server script (ASP.NET) to save uploaded file
licence_trial.txt:	A copy of the licence agreement

Running the installer also registers the ocx file and provides shortcuts to the example web page and the instructions on the Windows Start menu. The installer does not register the server component WebTwainXUploadASP.dll as it is assumed the user may wish to move this file to a different location, or different machine, in order to use it.

1.2. The Trial and Full Versions

The trial version has all the functionality of the full version of the component. The only limitation is that each image uploaded using the control will have a line of text written on it indicating that trial software was used. Visit www.ciansoft.com to purchase the full version.

The full version of WebTwainX is also supplied as an executable installer. It has a different OCX file, called WebTwainX.ocx. The other files are also named as for the trial version but without the word "trial".

The trial and full versions of the OCX have different classids, so when converting an application built using the trial version to work with the full version, the classid must be changed. Refer to Section 2.2 for more details.

1.3. Programming Environments

WebTwainX is designed specifically to be used as a clientside browser control in a web application. It can however be used in any other environment that supports the use of ActiveX controls, e.g., a compiled desktop application using a language such as Visual Basic.

All example code given in this document is based on a browser application and uses Javascript as the clientside scripting language.

1.4. Setting Up a Working Example

As a first step to using WebTwainX it is recommended that a working example is set up using the trial version and the example files provided. The necessary files must be set up on a web server and then the application accessed from a browser on a client machine. For testing purposes it is possible for the server and client to be on the same physical machine, in which case the client accesses the application using a ‘localhost’ address in the browser.

On the server, create a folder where the application will be located and copy the following files to this folder.

- WebTwainXTrial.cab
- webtwainxdemo.htm
- webtwainxtrial.lpk

Open the file webtwainxdemo.htm in a suitable editor (Notepad is adequate) and edit the Javascript line that sets the value of the UploadURL property. This property sets the address of the script that WebTwainX will call to upload the files. For example, if the server is on the same machine as the client (localhost), the files are in a folder called "test" and ASP is to be used to process the upload, this line should be:

```
wtx.uploadurl = "http://localhost/test/uploadsave.asp"
```

If ASP.NET is to be used, the filename should be "uploadsave.aspx".

Next, the server component for receiving the uploaded files must be installed. We will give two examples, using ASP or ASP.NET, but other server side scripting languages, e.g., PHP can be used.

For ASP, the component WebTwainXUploadASP.dll must be registered on the server. Copy this file to any chosen location on the server and register it. To do this, the command line utility regsvr32.exe can be used. This is usually found in the Windows system folder and runs using the syntax:

regsvr32 *dllname*

where *dllname* is the path and name of the dll file to register.

Warning: It is common practice to put this dll file into the Windows\System32 directory, however, on 64-bit operating systems, this location (despite the name) is reserved for 64-bit files only and the dll must not be located here. It can be located anywhere else on the file system.

The application using the component must have appropriate permissions. This means that for use in ASP, the Internet Guest User account on the server must have Read and Execute permissions on the DLL file. Write permission must also be allowed on the folder where the application files have been placed, as the uploaded file will be saved here.

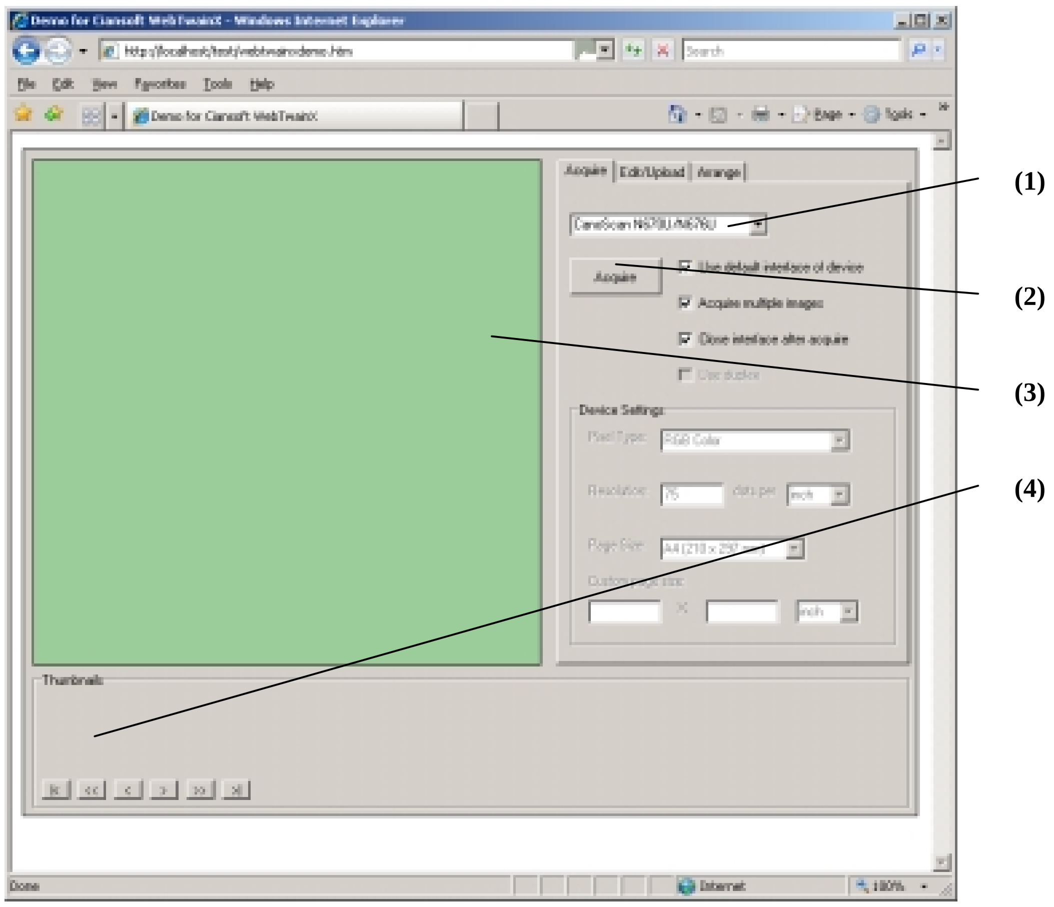
For ASP.NET, the component Ciansoft.WebTwainXUploadNet.dll must be copied to the \bin directory. It does not need to be registered. The ASP.NET machine account must be have Read and Execute permission on the DLL file.

The application is now ready to run. Open Internet Explorer on the client machine and enter the URL for the webtwainxdemo.htm page in the address bar. For example, this may be:

http://localhost/test/webtwainxdemo.htm

Depending on the version of Internet Explorer being used, and the security settings, one or more warnings may be shown asking the user if they want to allow the downloaded control to run. Accept the warnings and allow WebTwainX to run.

The page now loads and should look like this.


Demo for Ciansoft WebTwainX - Windows Internet Explorer
http://localhost/test/webtwainxdemo.htm
File Edit View Favorites Tools Help
Acquire
Edit/Upload
Arrange
Acquire
Thumbnails
(1)
(2)
(3)
(4)

CanoScan N670U/N676U

☑ Use default interface of device

☑ Acquire multiple images

☑ Close interface after acquire

☐ Use duplex

Device Settings

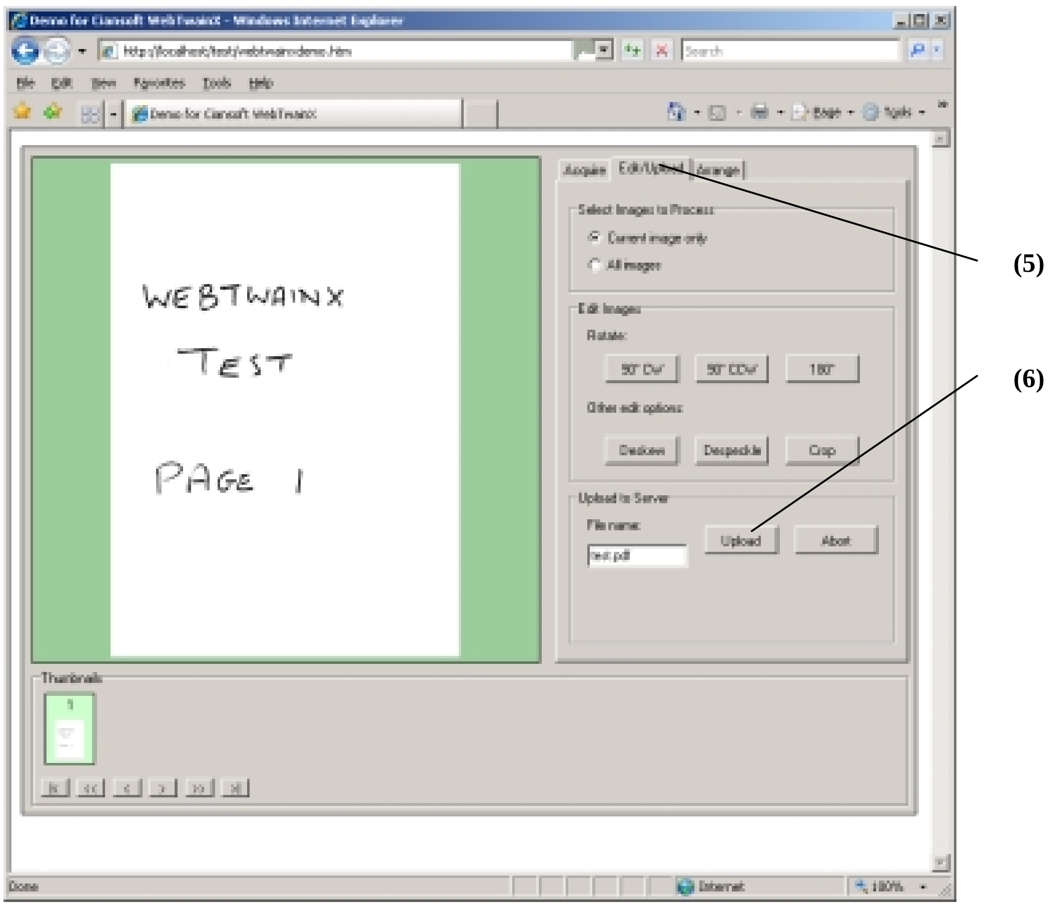
Pixel Type: RGB Color

Resolution: 75 dots per inch

Page Size: A4 (210 x 297 mm)

Custom page size: x inch

The Combobox indicated by (1) above should contain a list of TWAIN devices found on the client system. Select one these devices. Next, click the Acquire button (2) to scan or capture an image. The interface of the device driver will be displayed and an image can be acquired. After the device interface closes, the image should be displayed in the main image area of WebTwainX (3) and a thumbnail of the image will be shown in the Thumbnail area (4) as shown below.



Next, click on the Edit/Upload tab (5) and then the Upload button (6). A progress bar will indicate first that the image is being processed (converted to the required format, in this case, PDF) and then uploaded. Note that this may be too fast to see, especially if the server is on the same machine as the client.

Your first use of WebTwainX is now complete, and the uploaded file should be found in the same folder on the server as where the other application files are located.

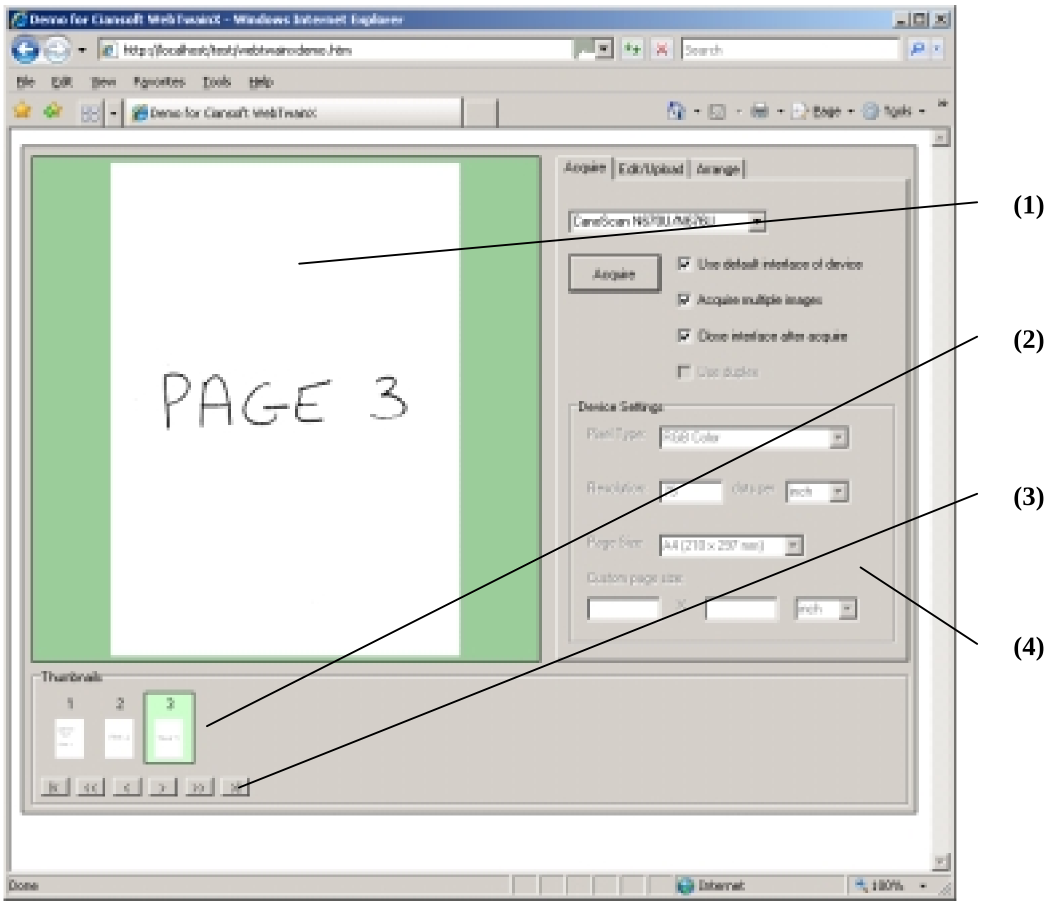
If the file has not been uploaded successfully, or a problem has been encountered at an earlier stage in the above procedure, please check the Troubleshooting section of this document which covers many of the common reasons for the application failing to work correctly.

2. Clientside Control

In a web application, WebTwainX is embedded in a web page and provides the interface for a user to acquire images using one or more TWAIN devices connected to the client, and then upload those images to the server.

When used in its default configuration, which is suitable for most purposes, only a single line of code needs to be written in a clientside script to configure WebTwainX This is to set the *UploadURL* property which identifies the server script used to receive uploaded images.

A screenshot of WebTwainX is shown in the figure below with key items highlighted. The key items are described below.


Demo for Ciansoft WebTwainX - Windows Internet Explorer
Acquire
Edit/Upload
Arrange
Acquire
Use default interface of device
Acquire multiple images
Close interface after acquire
Use duplex
Device Settings
PAGE 3
Thumbnails
(1)
(2)
(3)
(4)

(1) This is the main image area. It shows a large version of the image currently selected in the Thumbnail section.

(2) The Thumbnail area shows small versions of all the images that have been acquired since the application was started or reset by a Clear operation. An image can be selected by clicking on it.

(3) The buttons on the navigation bar can be used to step forwards or backwards through the available images. Page forward and page backward buttons move to the next row of images if there are more images available than can be seen together in the Thumbnail area. There are also buttons to move to the first or last image.

(4) The control panel provides all the functions for acquiring, editing, arranging and uploading images. It has three tabs. The first tab is used for acquiring images. The second has various options to edit images, plus the upload button. The third tab enables the images to be rearranged, e.g., moved up or down in the sequence.

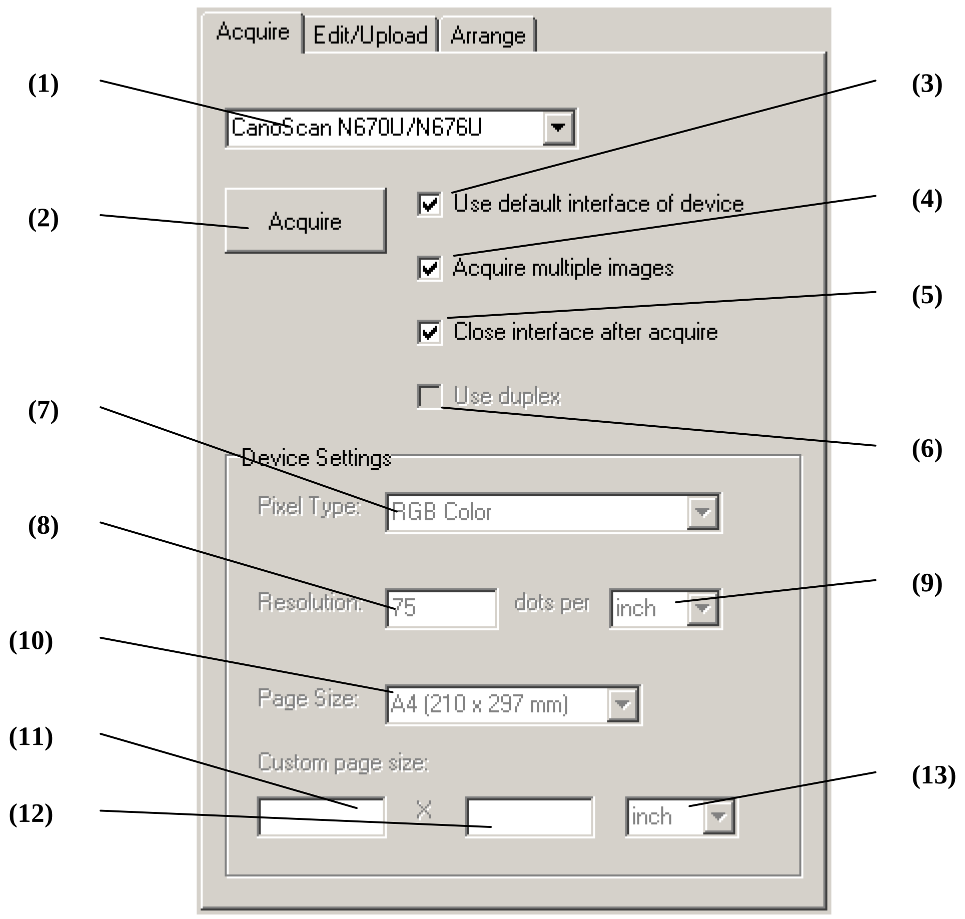
Warning: After using WebTwainX in a web page we strongly recommend that the browser should be closed and re-opened before further use. The use of memory in a TWAIN application is complicated as WebTwainX makes calls to the TWAIN Source Manager, and that in turn makes calls to the hardware manufacturer's device driver. Many drivers are less than perfect in their handling of memory allocation and can leave the browser in an unstable condition and liable to crash at a later time. Closing and re-opening the browser eliminates this problem.

The appearance of WebTwainX can be extensively customised and the methods/properties used to do this are all described in the section on Customising Appearance & Functionality. In addition to changing the layout of the form, buttons for features that are not required can be removed and all text fields can be replaced, e.g., to configure the control for use in a different language.

2.1. Control Panel Features

The control panel has three tabs and the controls available on each of these tabs are described below.

2.1.1 Acquire Tab



(1) The device (scanner or camera) to be used is selected here. This box is automatically populated with the names of devices found on the system when the control is first loaded.

(2) This button is clicked to start acquiring one or more images from the selected device.

(3) This checkbox determines whether or not the device driver's own interface will be used for acquiring images. If cleared, then the settings for acquiring an image will be taken from the

controls (7) to (13) in the frame labelled ‘Device Settings’. These controls are disabled if the interface is used.

(4) If checked, multiple images can be acquired from a single click of the Acquire button, provided that the selected device supports this.

(5) If checked, the device interface will be closed after acquiring images.

(6) Enabled duplex scanning if supported by the selected device.

(7) Selects the pixel type or colour depth.

(8) Sets the resolution. This can be either shown as a text box in which the user types a value, or as a drop-down list showing available resolution values, depending on the supported resolutions of the device.

(9) Selects the units of measure used for resolution.

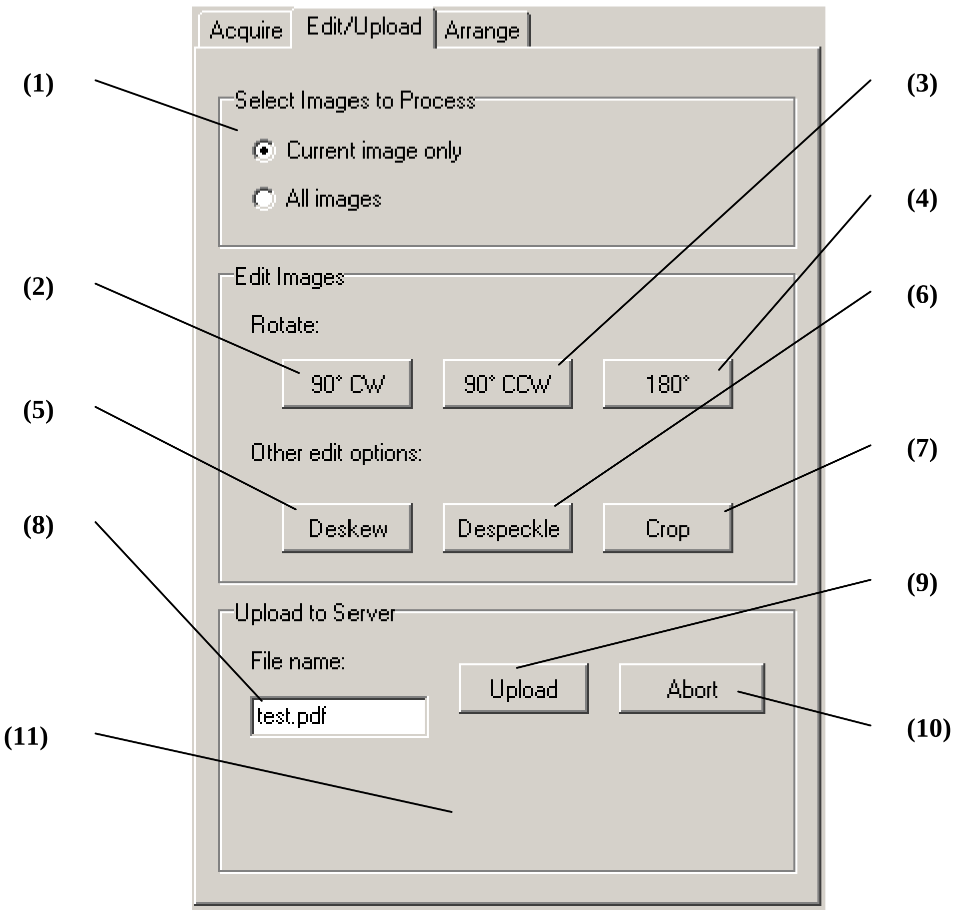
(10) Selects the page size from a list of standard sizes, or allows a custom size to be set.

(11) Page width can be entered here if a custom page size is used.

(12) Page height, as above.

(13) Units of measure for the page width and height.

2.1.2 Edit/Upload Tab



(1) A pair of option buttons to select whether a single image (the one currently displayed in the main image area) or all images will be processed by the editing or uploading options.

(2) Rotates the selected image or all images through 90° clockwise.

(3) Rotates through 90° counter-clockwise.

(4) Rotates through 180°.

(5) Deskews the image. This is intended for minor adjustments to the alignment of documents containing horizontal lines of text. It should not be used on other types of images as it can give unexpected results.

(6) Despeckles the image. As for the Deskew button, this is mainly intended for cleaning up scanned text pages, usually scanned in black and white. It removes dots from the image.

(7) When the Crop button is pressed, the cursor must then be moved to the main image area where a crosshair cursor is then displayed. The mouse must be clicked at one corner of a rectangle, then dragged to the opposite corner before being released. The image will then be cropped removing everything outside the selected rectangle. This feature can only be applied to one image at a time.

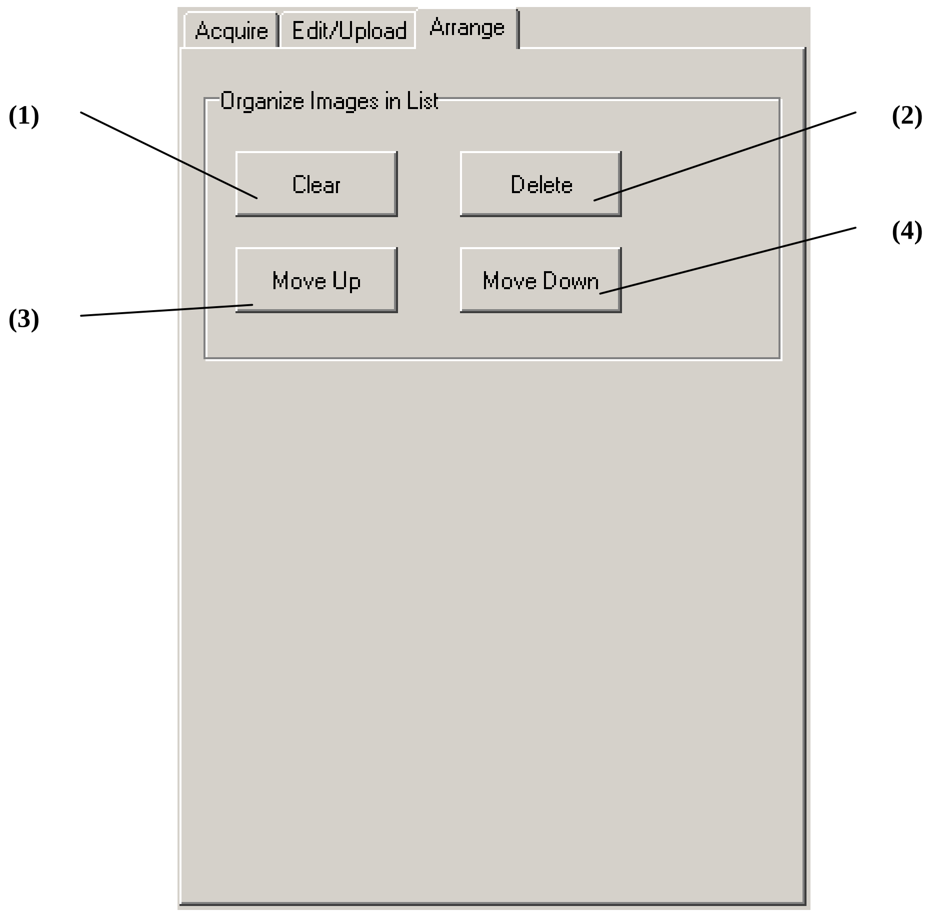
(8) A file name can be provided here for the uploaded file. This name is available to the server side script that receives the image, but does not necessarily have to be used by that script for saving the file.

(9) This button starts an upload, transmitting either the selected image or all images to a server as a single file.

(10) The Abort button is used to stop an upload while it is in progress.

(11) During an upload operation, a progress bar will appear in this space.

2.1.3 Arrange Tab



(1) Deletes all images currently held by the control.

(2) Deletes the single image currently displayed in the main image area

(3) Moves the current image one position up in the list of thumbnails.

(4) Moves the current image one position down in the list of thumbnails.

2.2. Embedding WebTwainX in a Web Page

WebTwainX is embedded in a web page by using OBJECT tags. Two tags are required, one for the control itself and one for the licence manager which references the licence package file.

For the trial version the following code should be added in the <body> section of the web page:

```
<OBJECT classid=clsid:5220cb21-c88d-11cf-b347-00aa00a28331><PARAM
NAME="LPKPath" VALUE="webtwainxtrial.lpk"></object>

<OBJECT id=wtx classid=CLSID:78C6D0A1-8D25-4238-B269-47F55108FB9C
codebase=webtwainxtrial.cab#version=1,3,0,0>failed to
load</object><br>
```

The first OBJECT tag is for the Microsoft License Manager. The classid is always the same.

The second OBJECT tag is for WebTwainX. The classid is different for the trial and full versions of the control. When using the full version, this line should be changed to:

```
<OBJECT id=wtx classid=CLSID:E1DDE407-68F1-4E89-B080-EDF751B64843
codebase=webtwainx.cab#version=1,3,0,0>failed to load</object><br>
```

The version parameter ("version=1,3,0,0") is not essential and can be removed. It forces the browser to download the latest version of the control if an older version is currently cached. If used, the version number must be correct, i.e., it must correspond to the version number of the ocx or cab file on the server.

2.3. Acquire Functions

Acquiring an image in WebTwainX can be done without writing any code but the following functions can be used to control some features either as alternatives to using the Control Panel, or to pre-set the various elements. These properties can be used to control the acquire process programmatically regardless of whether or not the related elements are hidden.

These functions can be used in an initialisation procedure or in a procedure called by the *OnSelect* event in order to adjust the settings when a new device is selected.

UseInterface [Boolean] - Read/Write Property.

This sets whether the device driver interface will be shown after the Acquire button is clicked. This property is linked to the CheckInterface checkbox if present, so changing the value of the property programmatically will change the setting of the checkbox and vice versa. (Default = True).

MultiImage [Boolean] - Read/Write Property.

This sets whether multiple images can be acquired from a single click of the Acquire button. This property is linked to the CheckMultiImage checkbox if present, so changing the value of the property programmatically will change the setting of the checkbox and vice versa. (Default = True).

CloseInterface [Boolean] - Read/Write Property.

This sets whether the device driver interface will be closed automatically after an image is acquired. This property is linked to the CheckCloseInterface checkbox if present, so changing the value of the property programmatically will change the setting of the checkbox and vice versa. (Default = True).

UseDuplex [Boolean] - Read/Write Property.

This sets whether duplex scanning will be enabled. This property is linked to the CheckDuplex checkbox if present, so changing the value of the property programmatically will change the setting of the checkbox and vice versa. It is not possible to enable duplex scanning if multiple image scanning is not enabled in the CheckMultiImage checkbox or if the selected device does not support duplex scanning. (Default = False).

UseADF [Boolean] - Read/Write Property.

This sets whether an automatic document feeder (ADF) is used when scanning multiple images. This property is linked to the CheckADF checkbox if present, so changing the value of the property programmatically will change the setting of the checkbox and vice versa. (Default = False).

DeviceCount [Integer] - Read-only Property.

Returns the number of TWAIN devices available for selection in the InputSelect Combobox.

DeviceName [String] - Read/Write Property.

The full name of the TWAIN device currently selected, as displayed in the InputSelect Combobox. When setting *DeviceName* programmatically the name must be exactly correct. It is case-sensitive.

DeviceIndex [Integer] - Read/Write Property.

The index number of the TWAIN device currently selected. This is the position of the device in the InputSelect Combobox list, the first device being 0, the second device 1 etc.

PixelType [Integer] - Read/Write Property.

This property is linked to the InputPixelType Combobox and can take one of the following values. (No default).

0	Black and White
1	Grayscale
2	RGB Color
3	Paletted Color

ResolutionUnits [Integer] - Read/Write Property.

This property is linked to the InputResolutionUnits Combobox and can take one of the following values.

0	Inches
1	Centimetres

When setting the resolution programmatically this property should be set first before setting the *Resolution* property. (No default).

Resolution [Double] - Read/Write Property.

This property is linked to the InputResolution element which can be a Combobox or a text box, depending on the available resolution options for the selected device. The units of measure for *Resolution* are determined by the InputResolutionUnits Combobox or the *ResolutionUnits* property. (No default).

PageSize [Integer] - Read/Write Property.

This property is linked to the InputPageSize Combobox and can take one of the following values. (No default).

0	A4 (210 x 297 mm)
1	A3 (297 x 420 mm)
2	A5 (148 x 210 mm)
3	Letter (8.5" x 11")
4	Tabloid (11" x 17")
5	Legal (8.5" x 14")
6	Statement (5.5" x 8.5")
7	Executive (7.25" x 10.5")
8	Custom Size

PageSizeUnits [Integer] - Read/Write Property.

This property is linked to the InputSizeUnits Combobox and can take one of the following values.

0	Inches
1	Centimetres

When setting a custom page size programmatically this property should be set first before setting the *PageWidth* and *PageHeight* properties. (No default).

PageWidth [Double] - Read/Write Property.

This property is linked to the InputWidth text box. The units of measure for *PageWidth* are determined by the InputSizeUnits Combobox or the *PageSizeUnits* property. This property is only used when the *PageSize* is set to 8 (Custom Size). (No default).

PageHeight [Double] - Read/Write Property.

This property is linked to the InputHeight text box. The units of measure for *PageHeight* are determined by the InputSizeUnits Combobox or the *PageSizeUnits* property. This property is only used when the *PageSize* is set to 8 (Custom Size). (No default).

PageSizeMode [Integer] - Read/Write Property.

This property determines the method used for setting the page size when scanning without using the device driver's interface (CheckInterface not checked). It can take one of the following values.

1 (Default)	The page size specified in the WebTwainX interface will be transmitted to the driver.
2	No page size information will be transmitted to the driver. The behaviour will be the default behaviour for that device. This may mean a standard page size is scanned, e.g., A4 or Letter, or it may mean that the page size is automatically detected, or that the previously used page size will be used.
3	The device driver will be requested to automatically detect the page size in the scanner. If the driver does not support this feature, the effect should be the same as *PageSizeMode* = 2.

MICREnabled [Boolean] - Read/Write Property.

Enables Magnetic Ink Character Recognition (MICR) on scanners which support this feature. Must be set to True before acquiring the image. The value will reset to the default value of False if a different device is selected, so it may be appropriate to set this property in the *OnSelect* event procedure. (Default = False).

MICRString [String] - Read-only Property.

Returns the MICR data associated with the current image as a string. This data can be uploaded with the image by setting a form variable.

Acquire - Method.

Starts the acquire process. This can be used to start acquiring programmatically from a script instead of clicking the Acquire button.

2.4. Upload Functions

The following properties and methods can be used to configure the upload behaviour of WebTwainX. Most of these functions will commonly be called from Javascript code in an initialisation procedure that runs when the web page and WebTwainX are first loaded, as in our example web page webtwainxdemo.htm.

UploadURL [String] - Read/Write Property.

The URL of the server script to which images will be uploaded using an HTTP POST operation. This should be prefixed with either "http:// " or "https:// ". If neither prefix is specified, "http:// " will be assumed as default.

This property must be set using Javascript or similar code in the web page. It has no default value and uploading of images is impossible without setting it.

Example:

```
wtx.UploadURL = "http://www.mydomain.com/uploadsave.asp"
```

FileName [String] - Read/Write Property.

The file name to be transmitted to the server with the upload of the image data. This string can be read by the server and may be used as the name for saving the image as a file on the server. This property is linked to the text entered in the InputFilename text box element if present, so changing the value of the property programmatically will change the value in the text box and vice versa.

If the *SaveLocal* method or the ButtonSaveLocal element is used (this is not displayed in the default interface configuration) then the value of *FileName* is used to save the image file on the local system. The file will be saved in the current users documents folder (My Documents on XP or Documents on Vista or Windows 7). Backslashes can be used in this string to define subdirectories which will be created if not already present, e.g., if FileName = "WebTwainX\Files\Image.tif" then the file 'Image.tif' will be saved under subdirectories 'WebTwainX' and 'Files'.

FileFormat [String] - Read/Write Property.

The file format to be used to encode the image data for uploading. *FileFormat* can take one of the following values to indicate a specific format:

"TIF"	Tagged Image File format
"PDF"	Portable Document format
"BMP"	Bitmap format
"GIF"	Graphics Interchange format
"JPG"	JPEG format
"PCX"	PCX format
"PNG"	Portable Network Graphics format
"WBM"	Wireless Bitmap format
"PSD"	Adobe Photoshop format

Alternatively, *FileFormat* can take the value "FileName". The file format will then be determined from the file extension from the *FileName* property or the text entered in the InputFilename text box. If this file name is not present or has an invalid extension, the format defaults to PDF.
(Default = "FileName").

FileNameExtension [String] - Read-only Property.

The file extension of the file name entered in the InputFilename text box or the *FileName* property. If the extension is not one of the valid file extensions to indicate a valid file format, or if there is no file extension, this property will be an empty string. The file extension always includes the period character and is always lower case.

UserName [String] - Read/Write Property.

The username to be used when uploading images if the server requires authentication.

Password [String] - Read/Write Property.

The password to be used when uploading images if the server requires authentication.

It is important to note that any username and password set using clientside code in a web page can easily be viewed by the user of the web page. These properties, especially *Password*, should only be set in code when a compiled application is being produced rather than a web application. If these properties are not pre-set, the user will prompted to enter login details manually when uploading.

AuthenticationType [Integer] - Read/Write Property.

The type of authentication required by the server when uploading. If Windows Integrated Authentication (NTLM) is to be used, this property must be set to 1, otherwise the upload will fail. With Basic Authentication or no authentication, *AuthenticationType* must be set to 0. As this is the default value, the property can be left unused. (Default = 0).

UploadTimeout [Integer] - Read/Write Property.

If there is a problem while uploading to a server, this property allows for the upload to cancel automatically after a length of time and return control to the user. The value is a timeout period in seconds. If set to zero there will be no timeout, so a problem on the server could lead to the browser locking up. (Default = 60).

AddFormVar [Integer] *Name [String], Variable [String]* - Method.

Adds an HTTP form variable to the upload. A variable has a name and a value, both of which are strings. They are used for passing additional information from the client to the server during an upload, e.g., they could be used to identify a user so that files are stored in a particular location. Any number of form variables can be added. All form variables are cleared after the upload is complete. The return value is the number of form variables added so far.

FormTagName [String] - Read/Write Property.

The name of the form variable containing the file for upload. Some server scripts and components can use this to identify the file. (Default = "WebTwainX").

Upload - Method.

Starts an upload. This can be used to control the upload from a script started using "ButtonCustom" and the *OnCustomClick* event.

AbortUpload - Method.

Aborts the current upload process. This can usually be called only from code triggered by the *OnStartUpload* event. It can be used to prevent the upload from proceeding if certain conditions have not been met, for example, if the user has provided a file name with an invalid extension in the InputFilename text box.

Example: Abort the upload if a file extension other than "pdf" has been specified by the user:

```
function StartUpload()
  {
  if (!(wtx.FileNameExtension == ".pdf"))
    {
      wtx.AbortUpload();
      alert("Upload aborted");
    }
  }
```

PagesSelected [Integer] - Read/Write Property.

This property is used to pre-set the option buttons OptionPages to determine whether a single page or all pages will be uploaded or edited. A value of 1 indicates a single page and 2 indicates all pages. (Default = 1).

ImageIndex [Integer] - Read/Write Property.

The index number of the current image.

ImageCount [Integer] - Read-only Property.

The number of images currently held in the control.

MaxUploadSize [Integer] - Read/Write Property.

The maximum allowed size of an uploaded file, in bytes. If this value is exceeded, the upload will be aborted automatically. A value of 0 means that there is no limit on the size. (Default = 0).

The following properties are set when an upload is completed and can be used to confirm the status of the upload or to diagnose problems.

UploadStatus [Integer] - Read-only Property.

This will contain the HTTP return code following an upload operation, or a specific error code as described below. A value of 200 indicates that the upload was successful. 404 indicates that the URL was not found and 500 indicates a server error. A value of 1 means that the upload was aborted by the user clicking the Abort button and a value of 2 indicates that the upload was automatically aborted due to exceeding the maximum size limit specified by *MaxUploadSize*.

UploadReturnString [String] - Read-only Property.

This is the full contents of the data returned to the browser following an upload operation. It may contain information from the server that can be useful in diagnosing the cause of a problem.

SaveUploadReturnString *Filename [String]* - Method.

The data returned by *UploadReturnString* is often too long to be displayed in a dialogue box, for example, by calling the ‘alert’ command in Javascript. As the string is a full HTML page, the important information relating to error messages from the server is usually near the end of the string.

SaveUploadReturnString saves the data in *UploadReturnString* to a local file so that it can be viewed in its entirety. The file will be saved in the current users documents folder (My Documents on XP or Documents on Vista or Windows 7).

2.5. Editing Functions

Editing functions are used to control the operation of the various buttons for editing images.

AddText [String] - Read/Write Property.

The text string to be added to the image when ButtonAddText is clicked. The text is added close to the top-left corner of the image and will be written in black on a white background. This button is not displayed in the default interface configuration.

This property is linked to the text entered in the InputAddText text box element if present, so changing the value of the property programmatically will change the value in the text box and vice versa.

RotateCW - Method.

Rotates one or all images by 90° clockwise. This can be called from a script and has the same effect as clicking the button ButtonRotateCW. The *PagesSelected* property should be set first to determine whether a single image or all images are rotated.

RotateCCW - Method.

Rotates one or all images by 90° counter-clockwise. This can be called from a script and has the same effect as clicking the button ButtonRotateCCW. The *PagesSelected* property should be set first to determine whether a single image or all images are rotated.

Rotate180 - Method.

Rotates one or all images by 180°. This can be called from a script and has the same effect as clicking the button ButtonRotate180. The *PagesSelected* property should be set first to determine whether a single image or all images are rotated.

2.6. Miscellaneous Functions

Clear - Method.

Deletes all images. This is equivalent to clicking “ButtonClear” and can be used to clear the images from the control from a script.

CustomText1 … 10 [String] - Read/Write Properties.

These string properties are linked to the text entered in elements InputText1 to InputText10. They can be used either to preset the text displayed in these text boxes, or to retrieve the value entered by the user.

They operate in a similar way to the *FileName* property and the InputFilename text box and can be used to obtain additional input from the user which can, for example, be uploaded to the server as form variables.

SaveLocal - Method.

This saves to the local file system in the same way as if “ButtonSaveLocal” is clicked.

SaveLocalSuccess [Boolean] - Read-only Property.

Indicates whether a local save operation using the “ButtonSaveLocal” button or the *SaveLocal* method has succeeded. True indicates success, False indicates failure.

SaveLocalError [String] - Read-only Property.

Contains an error message if a local save operation using the “ButtonSaveLocal” button or the *SaveLocal* method has failed.

CheckConnection [Boolean] - Read/Write Property.

When the user interface of WebTwainX is initialised, the connection to each device driver is checked and any that do not connect are disabled. Setting this property to False overrides this behaviour and makes the device available from the interface. This setting should be used with caution as attempting to use a device that is not correctly connected can give unpredictable results. (Default = True).

The following two functions can be used to confirm the version number of the OCX file being called by the browser. This can be useful after changing to a newer version on the server to confirm that an earlier version is not being cached by the browser.

Version [String] - Read-only Property.

This returns the version information for the OCX file.

AboutBox - Method.

Displays a dialogue box with information about the OCX file, including the version number.

2.7. Events

The following events can be used. By adding event handling procedures to an application, specific actions can be carried out, e.g., a message could be displayed when an upload is completed.

OnStartUpload - Event.

This is raised when the Upload button is clicked before the upload process begins.

OnFinishUpload - Event.

This is raised when an upload is completed, regardless of whether or not the upload was successful.

OnSelect - Event.

This is raised when a new device is selected in the InputSelect Combobox. It can be used to set default properties such as PixelType or Resolution when a new device is selected by the user. This event is not raised when the device selection is changed programmatically using the *DeviceName* or *DeviceIndex* properties.

OnAcquire - Event.

This is raised when an image is successfully acquired by the control. If multiple images are being acquired, the event is raised for each individual image as it is received.

OnFinishAcquire - Event.

This is raised when the control completes an Acquire operation, either as a result of clicking the btnAcquire button or calling the *Acquire* command. The event is raised regardless of whether the Acquire was successful or not. The *ImageCount* property can be used to confirm that images were actually acquired.

OnCustomClick - Event.

This is raised when the "ButtonCustom" button is clicked.

OnSaveLocal - Event.

This is raised after a local save operation using the "ButtonSaveLocal" button. The event is fired regardless of whether or not the save succeeded. The status if the save operation can be checked using the *SaveLocalSuccess* and *SaveLocalError* properties.

To use an event in a Javascript application the event must be linked to a procedure. For example, the following code in the <body> section of an HTML page links the *OnFinishUpload* event to a procedure called FinishUpload.

```
<script language="JavaScript" for="wtx" event="onFinishUpload()">
  FinishUpload();
</script>
```

An example of this can be seen in the webtwainxdemo.htm demo page provided with WebTwainX.

3. Uploading to ASP

In ASP, uploads to the server are handled using the COM object WebTwainXUploadASP.dll.

In the demo script uploadsave.asp provided in the WebTwainX installation package, this component is used very simply to save a file in the folder where the script is located. The file name sent from the client with the upload is used and there is no error checking. The whole script is:

```
<%
  Set Upload = Server.CreateObject("WebTwainXUploadASP.FileSave")

  'Save to the current folder using the existing file name
  Upload.SaveFile Server.MapPath(Upload.FileName)
%>
```

This section describes all the features of this component.

3.1. Object Creation

In any script that uses the component an object instance must be created. The syntax in ASP is:

```
Set Upload = Server.CreateObject("WebTwainXUploadASP.FileSave")
```

The object name is "Upload", but any variable name could be used.

There is a property called *Version* which returns the version number of the component. This can be used to confirm that the component is running successfully.

Version [String] - Read-only Property.

This returns the version information and can be used to confirm that the component is running.

Example:

```
Set Upload = Server.CreateObject("WebTwainXUploadASP.FileSave")
Response.Write Upload.Version
```

3.2. Operating System Issues

There are a number of issues to be aware of on some versions of Windows.

3.2.1 Windows 2003 and Later

Windows 2003 (IIS 6) introduced a property that limits the amount of data that can be received using a POST operation, and this has a relatively low default value. This property can be changed by editing the metabase.xml file and it is called AspMaxRequestEntityAllowed. If a file is uploaded that is larger than specified by this property an error will result.

In IIS 7, if Friendly Names are used, it will be called Maximum Entity Requesting Body Limit.

3.2.2 64-bit Operating Systems

WebTwainXUploadASP.dll is a 32-bit DLL but it can be used on 64-bit systems if it is added to a COM+ Application. An online description of configuring Component Services is available on our partner website here:

http://www.chestysoft.com/component-services.asp.

It is important to "Allow intrinsic IIS properties" in the COM+ component properties.

On 64-bit systems, the Windows\System32 folder is reserved for 64-bit files only, so the DLL must not be located in this folder.

3.3. Saving or Exporting the Uploaded File

The uploaded file can be saved to disk on the server using *SaveFile*. It can be exported as a variant array using *FileAsVariant*, and this would be used to store the image in a binary database field or to send the image to a server component for image processing. If image processing on the server is required, we recommend the csImageFile component from Chestysoft, which supports this method of image transfer.

Some properties are available to provide information about the file, such as *FileSize* and *Filename*.

3.3.1 Saving to Disk

SaveFile *Filename [String]* - Method.

This saves the uploaded file to the full physical path specified in *Filename*. The Internet Guest User must have Write permission on the destination directory to save the file and Modify permission to overwrite an existing file.

Example: Saving using a hard-coded file name:

```
Upload.SaveFile "C:\files\newfile.pdf"
```

Use Server.MapPath to convert a virtual path to a physical path.

Example:

```
Upload.SaveFile Server.MapPath("newfile.pdf")
```

3.3.2 Exporting as Binary Data

FileAsVariant [Variant array] - Read-only Property.

The uploaded file exported as a binary variable.

Example: Saving the file to a binary database field called "Image":

```
RSet("Image") = Upload.FileAsVariant
```

Example: Exporting the file to the Chestysoft csImageFile component for further processing where the instance of csImageFile is called "Image":

```
Image.ReadVariant = Upload.FileAsVariant
```

3.3.3 File Properties

The following properties provide information about the uploaded file.

ContentType [String] - Read-only Property.

The MIME type of the uploaded file.

Filename [String] - Read-only Property.

The name of the file that was uploaded, complete with extension. This will correspond to the *FileName* property set in the clientside control before uploading.

FileExtension [String] - Read-only Property.

The file extension, without the period character.

FilenameNoExtension [String] - Read-only Property.

The file name, with the extension removed.

FileSize [Integer] - Read-only Property.

The size of the uploaded file in bytes.

3.3.4 OverwriteMode

The *OverwriteMode* property provides a way to prevent files with duplicate names from being overwritten.

OverwriteMode [Integer] - Read/Write Property.

This property determines what happens if the *SaveFile* method attempts to write to a file that already exists. It can take the value 0, 1 or 2 which have the meanings given below. (Default = 0).

0	Any existing file will be overwritten.
1	The new file will not be saved if an existing file has the same name.
2	If the new file name matches an existing name the characters "~x" will be added at the end where "x" is the smallest number needed to make the name unique. The character "~" can be changed using the *OverwriteChr* property.

NewName [String] - Read-only Property.

When an *OverwriteMode* of 2 is used, the file property *NewName* will be set to the name of the file that was actually saved.

Example:

```
Upload.OverwriteMode = 2
Upload.SaveFile "C:\temp\newfile.tif"
Response.Write Upload.NewName
```

The above example will save the uploaded file as "newfile.tif", if that name is not already used. If a file with that name exists, it will save it as "newfile~1.tif" (or newfile~2.tif … etc.) The name actually used will be written out in the Response.Write statement which could then be read from the *UploadReturnString* property of the clientside control.

OverwriteChr [String] - Read/Write Property.

The character or characters added before the number when a file is renamed using *OverwriteMode*. This might be necessary on Windows 2003 or if URLScan is used because URLs containing this character may be blocked from downloading. (Default = "~").

3.4. Form Variables

Form variables added to the upload using the *AddFormVar* function in the clientside control cannot be read using Request.Form, so the following properties must be used instead. *FormVariable* reads the variable given its name. *FormVariableCount* returns the number of variables in the form, and *FormVariableByIndex* returns the value given the index of the variable within the form.

FormVariable [String] *VariableName [String]* - Read-only Property.

The value of the variable where *VariableName* is the string name of the variable.

FormVariableCount [Integer] - Read-only Property.

The number of form variables in the upload. This includes the name of the uploaded file.

FormVariableByIndex [String] *Index [Integer]* - Read-only Property.

The value of the variable specified by *Index*. *Index* represents the number of the variable within the form, where the first has an index of zero and the last has an index of *FormVariableCount* - 1.

4. Uploading to ASP.NET

In ASP.NET, uploads to the server are handled using Ciansoft.WebTwainXUploadNet.dll which is a .NET class.

In the demo script uploadsave.aspx provided in the WebTwainX installation package this component is used very simply to save a file in the folder where the script is located. The file name sent from the client with the upload is used and there is no error checking. The whole script is:

```
<%@ Page language="vb" debug="true" %>
<%@ Import NameSpace = "WebTwainXUploadNet" %>
<%
Dim Upload As New UploadClass
Upload.ReadUpload
Upload.SaveFile(Server.MapPath(Upload.Filename))
%>
```

This section describes all the features of this component.

4.1. Installation

The DLL file Ciansoft.WebTwainXUploadNet.dll must be placed in the \bin folder for the web application. The ASP.NET machine account must have Read and Execute permission on the DLL file.

The permissions on the \bin folder should not allow access for the anonymous internet user, IUSR_machine_name or IUSR. This is to prevent users from downloading DLLs or components.

The ASP.NET script must import the namespace WebTwainXUploadNet. The class name is UploadClass and the *ReadUpload* method must be called to extract the data from the upload.

4.2. Object Creation

In any script that uses the component an object instance must be created. The syntax is:

In VB.NET:

```
<%@ Page language="vb" debug="true" %>
<%@ Import NameSpace = "WebTwainXUploadNet" %>
<%
Dim Upload As New UploadClass
Upload.ReadUpload
...
%>
```

In C#:

```
<%@ Page language="c#" debug="true" %>
<%@ import Namespace = "WebTwainXUploadNet" %>
<%
UploadClass Upload = new UploadClass();
Upload.ReadUpload();
...
%>
```

All the remaining examples are shown using VB.NET.

The object name is "Upload", but any variable name could be used.

There is a property called *Version* which returns the version number of the component. This can be used to confirm that the component is running successfully.

Version [String] - Read-only Property.

This returns the version information and can be used to confirm that the component is running.

Example:

```
Dim Upload As New UploadClass
Response.Write(Upload.Version)
```

4.3. Saving or Exporting the Uploaded File

The uploaded file can be saved to disk on the server using *SaveFile*. It can be exported as an array of bytes, and this would be used to store the image in a binary database field. It can be exported to a MemoryStream which can be used to pass the data into a Bitmap object or a server component for image processing. If image processing on the server is required, we recommend the csImageFile component from Chestysoft, which supports this method of image transfer.

Some properties are available to provide information about the file, such as *FileSize* and *Filename*.

Before files can be saved or exported, the *ReadUpload* method must be called.

ReadUpload - Method.

This method extracts the file data from the upload.

4.3.1 Saving to Disk

SaveFile *Filename [String]* - Method.

This saves the uploaded file to the full physical path specified in *Filename*.

Example: Saving using a hard-coded file name

```
Upload.SaveFile("C:\files\newfile.pdf")
```

Use Server.MapPath to convert a virtual path to a physical path.

Example:

```
Upload.SaveFile(Server.MapPath("newfile.pdf"))
```

4.3.2 Exporting as Binary Data

FileToArray [Array of bytes] - Read-only Property.

This returns an array of bytes containing the file.

FileToStream [MemoryStream] - Read-only Property.

This returns a MemoryStream containing the file.

4.3.3 File Properties

The following properties provide information about the uploaded file.

ContentType [String] - Read-only Property.

The MIME type of the uploaded file.

Filename [String] - Read-only Property.

The name of the file that was uploaded, complete with extension. This will correspond to the *FileName* property set in the clientside control before uploading.

Extension [String] - Read-only Property.

The file extension, including the period character.

FileSize [Integer] - Read-only Property.

The size of the uploaded file in bytes.

4.4. Examples

Here are some further examples of code to process the uploaded images.

4.4.1 Saving the File to a Database Field

In most cases, when a file is uploaded to the server it is saved on disk. Sometimes there is a requirement to write the file into a binary database field. The following example shows how a file can be written to a Microsoft Access database using a field type of OLE Object.

```
<%@ Page language="vb" debug="true" %>
<%@ Import NameSpace = "System.Data.OleDB" %>
<%@ Import NameSpace = "System.Data" %>
<%@ Import NameSpace = "System.DateTime" %>
<%@ Import NameSpace = "WebTwainXUploadNet" %>
<%
Dim Upload As New UploadClass
Upload.ReadUpload
Dim ConnectionString As String =
"PROVIDER=MICROSOFT.JET.OLEDB.4.0;DATA   SOURCE=" &
Server.Mappath("sample.mdb")
Dim AConnection As OleDbConnection = New
OleDbConnection(ConnectionString)
Dim SQLString As String = "SELECT * FROM Table1 WHERE 1=2"
Dim DA As OLEDBDataAdapter = New OLEDBDataAdapter(SQLString,
ConnectionString)
Dim CommandBuilder As OleDbCommandBuilder = New
OleDbCommandBuilder(DA)
Dim DS As DataSet = New DataSet("Table1")
DA.MissingSchemaAction = MissingSchemaAction.AddWithKey
DA.Fill(DS, "Table1")
Dim Row As DataRow = DS.Tables("Table1").NewRow
Row("FileName") = Upload.FileName
Row("FileData") = Upload.FileToArray
Row("ContentType") = Upload.ContentType
Row("Extension") = Upload.Extension
Row("UploadDate") = DateTime.Now
DS.Tables("Table1").Rows.Add(Row)
DA.Update(DS, "Table1")
AConnection.Close
%>
```

A lot of the code is creating the various classes that are used to open and write to the database. It must use a data adapter in order to handle complex data types, such as the OLE Object. The database fields "FileName", "ContentType" and "Extension" contain information about the file and "UploadDate" contains the current date. "FileData" is the binary data and this can be written directly using the FileToArray method from WebTwainXUploadNet.

4.4.2 Copying the File to a Bitmap Object

The *FileToStream* method can be used to copy uploaded images into a System.Drawing.Bitmap class. The following code checks the extension of the uploaded image and if it is .bmp or .jpg it loads the image.

```
<%@ Page language="vb" debug="true" %>
<%@ Import NameSpace = "WebTwainXUploadNet" %>
<%@ Import NameSpace = "System.Drawing" %>
<%
Dim Upload As New UploadClass
Upload.ReadUpload
If (Upload.Extension = ".bmp") or _
  Upload.Extension = ".jpg") Then
  Dim BMP As New Bitmap(Upload.FileToStream)
End If
%>
```

Once the image has been loaded into a Bitmap object, further image processing can be done before saving the image. Other file types are supported by the Bitmap object but only jpeg and bmp have been used here for simplicity.

4.4.3 Copying the File into the csImageFile COM Object

Image files can be copied directly into the Chestysoft csImageFile COM object, using similar code to that shown above. The *FileToArray* method is used.

```
<%@ Page language="vb" debug="true" %>
<%@ Import NameSpace = "WebTwainXUploadNet" %>
<%@ Import NameSpace = "System.Drawing" %>
<%
Dim Image = Server.CreateObject("csImageFile.Manage")
Dim Upload As New UploadClass
Upload.ReadUpload
If (Upload.Extension = ".bmp") or _
  (Upload.Extension = ".jpg") Then
  Image.ReadVariant(Upload.FileToArray)
End If
%>
```

The Extension method can be used to check if the file is in a valid format, and if so, it can be loaded using the csImageFile ReadVariant method. As with the Bitmap example in the previous section, further image processing can be carried out before saving the image. Other file types are supported by csImageFile but only jpeg and bmp have been used here for simplicity.

4.5. Server Settings and Configuration

This section summarises the configuration settings that must be made on the server to run WebTwainXUploadNet.

4.5.1 Location and Permissions for the DLL

The DLL file, Ciansoft. WebTwainXUploadNet.dll must be located in the binary folder for the web application. By default, this folder is called "\bin". The permission settings on the DLL must allow the ASP.NET machine account Read and Execute permission. The Internet Guest User account (IUSR_machine_name or IUSR) should not be allowed to read the DLL, to protect it from unauthorised downloads.

4.5.2 Permissions for Saving Files

The ASP.NET machine account must have Write permission in the folder where the uploaded files are to be saved.

4.5.3 Limiting the Size of File Uploads

There is a size limit on file uploads, which is determined by the web application configuration. The size limit is defined by the maxRequestLength attribute of the httpRuntime element in either the Machine.config or Web.config files. The default value is 4 MB and usually this value is left in the Machine.config file and a Web.config file is placed in the application root folder, or the script folder, with a new value to override the default.

Here is an example of the code to be placed inside a Web.config file to set the upload size limit to 10 MB.

```
<configuration>
  <system.web>
    <httpRuntime maxRequestLength="10240" />
  </system.web>
</configuration>
```

The value is measured in kilobytes.

When a file is uploaded which is over the size limit, it produces a "Cannot find server or DNS error" message. This is not user friendly and it might be preferable to set the maxRequestLength to a value that is larger than any expected file size and if the file size needs to be limited to a smaller value, read the Request.TotalBytes property to find the uploaded size before calling WebTwainXUploadNet.

5. Customising Appearance & Functionality

The default appearance of the WebTwainX interface is designed to be easy to use and to provide a comprehensive set of features. By adding configuration commands as described in this section, the appearance and content of the interface can be extensively customised to provide an alternative view to the end user.

Customisation can include:

- Changing the size or position of the main interface elements, i.e., the main image area, the thumbnails area and the control panel.
- Removing items from the interface that are not required.
- Changing the size or position of items on the control panel, including moving items to a different tab, or associating items together in a frame.
- Changing the text strings displayed on buttons and elsewhere on the interface. This is especially useful if a different language is to be used.

All the commands described in this section should be used by placing them in an initialisation procedure that is executed as soon as the page containing WebTwainX is loaded.

If customisation that changes the overall height or width of the control is used, the browser might not immediately update its display. To force the browser to redraw, the following line of code can be added at the end of the initialisation procedure:

```
wtx.style.display = "block";
```

5.1. Elements

Each item on the interface that can be modified is referred to as an "Element". Each Element has a unique name that is used by the various functions. For example, the thumbnails area is an element called "Thumbnails" and to change the width of this area to 500 pixels, the following code could be used:

```
wtx.SetWidth("Thumbnails", 500);
```

Element names are not case sensitive, but must be spelt correctly.

There are three main elements that make up the interface and these are the main image area ("CurrentImage"), the thumbnails area ("Thumbnails") and the control panel ("ControlPanel"). The size (width and height) of each of these can be changed as well as the position (left and top co-ordinates). The overall size of the interface cannot be set but will be automatically determined in order to fit the three main elements inside it.

The remaining elements are all components of the control panel, including buttons, checkboxes, etc. The control panel has up to five tabs (three are used in the default configuration), which are themselves defined as elements, and each remaining element must be contained either by one of the tabs, or by a frame. Any frames must be contained by a tab.

To understand this more clearly, consider the default interface configuration as shown on Page 6. There is a drop-down list or combobox (element: "InputPixelType") for selecting the pixel type which is contained in a frame with the title "Device Settings" (element: "Frame1"). The frame is contained by the first tab (element: "Tab1") of the control panel. So the following relationships are defined, and can be changed:

Frame1 is the container for InputPixelType.

Tab1 is the container for Frame1.

Also, ControlPanel is the container for Tab1, but this relationship cannot be changed.

A complete list of available elements is given in the tables in Section 5.3.

5.1.1 Additional Elements

A number of spare or custom elements are available and these can be used to add additional features to the interface. Frame 6 to Frame10 can be used in addition to the five frames used in the default configuration. There are ten labels, Label1 to Label10 which can be used for any purpose.

Ten text input boxes, InputText1 to InputText10, can also be used. These are linked to a set of properties CustomText1 to CustomText10.

Five drop-down lists or comboboxes, ComboBox1 to ComboBox5 are also available. If used, these need to be configured using the commands *SetComboBoxItems* and *SetText*, and the selected items can be found by reading the *ComboBoxIndex* property.

5.2. Configuration Functions

DebugAlerts [Boolean] - Read/Write Property.

When set to True, a message will be displayed on screen whenever a non-zero error code is generated by one of the configuration functions. It is recommended that *DebugAlerts* be set to True to assist with debugging while a new configuration is being developed, but then reset to False before deploying the application. (Default = False).

Example:

```
wtx.DebugAlerts = True;
```

ControlPanelTabs [Integer] - Read/Write Property.

The number of tabs on the control panel. If this is reduced to 1, no tabs will be displayed and the control panel will appear as a single frame. Any elements assigned to tabs that are not displayed will be unavailable, so for example, the buttons shown on Tab3 in the default configuration will have to be moved to another tab if *ControlPanelTabs* is reduced to 1 or 2 and those buttons are still required. (Default = 3).

The maximum number of tabs is 5. Note that Tab4, which is not used in the default configuration, has spare labels and frames assigned to it. These need to be hidden if Tab4 is activated and they are not required.

Example:

```
wtx.ControlPanelTabs = 2;
```

ThumbRows [Integer] - Read/Write Property.

The number of rows of thumbnail images to be displayed in the thumbnails area. (Default = 1).

The layout of the thumbnails area is determined as follows. The height and width of the "Thumbnails" element can be set (using *SetHeight* and *SetWidth*) and the number of rows. WebTwainX will then check whether or not that number of rows can fit in the height, given that there is a minimum size for displaying any image and there must be space for borders, for image numbering, and for the navigation bar. If the specified number of rows cannot fit, it will be reduced automatically. If the height is too small for even one row to fit, the height will be increased to fit one row.

The following functions can be used to configure elements. Each function gives a return value which is an error code to indicate whether the command succeeded or not. The possible values of this error code are:

0	Ok
1	Element not found (check spelling)
2	This property is not available for the specified element
3	The value is not valid and is ignored
4	The value is inconsistent with other values, but is used
5	The value is inconsistent with other values; a different value has been set
6	Index number out of range for SetText
7	Specified container is not valid
8	Number of thumbnail rows reduced to fit in available height
9	Height of Thumbnails element increased to fit single image
10	Width of Thumbnails element increased to fit single image
99	Unidentified error

SetLeft [Integer] *Element [String], Left [Integer]* - Method.

Sets the horizontal position of the element. The value *Left* gives the position of the left side of the element, in pixels, relative to the left side of the element's container.

Example:

```
wtx.SetLeft("ButtonAcquire", 20);
```

SetTop [Integer] *Element [String], Top [Integer]* - Method.

Sets the vertical position of the element. The value *Top* gives the position of the top of the element, in pixels, relative to the top of the element's container.

Example:

```
wtx.SetTop("InputSelect", 50);
```

SetWidth [Integer] *Element [String], Width [Integer]* - Method.

Sets the width of the element in pixels.

Example:

```
wtx.SetWidth("ButtonRotateCW", 150);
```

SetHeight [Integer] *Element [String], Height [Integer]* - Method.

Sets the height of the element in pixels.

Example:

```
wtx.SetHeight("Frame1", 220);
```

SetContainer [Integer] *Element [String], Container [String]* - Method.

Assigns the element *Element* to be contained by the element *Container*.

Example: Set the upload button to be contained by the first tab of the control panel, Tab1:

```
wtx.SetContainer("ButtonUpload", "Tab1");
```

SetText [Integer] *Element [String], Text [String], Index [Integer]* - Method.

Changes the text string displayed for the element. For elements which have more than one text string, *Index* defines the string to be changed. For combobox elements, *SetText* sets the strings in the list, the first string having an *Index* value of 1.

Example: Change the acquire button text to "Scan" instead of "Acquire":

```
wtx.SetText("ButtonAcquire", "Scan", 0);
```

Example: Change the first option in the InputPixelType element to read "Monochrome" instead of "Black and White":

```
wtx.SetText("InputPixelType", "Monochrome", 1);
```

SetBorder [Integer] *Element [String], Border [Integer]* - Method.

This method is only applicable to two elements.

For "CurrentImage", this sets the minimum size in pixels of the coloured border around the displayed image.

For "Thumbnails" this sets the amount of space in pixels around each thumbnail image. The minimum value it can be set to is 8.

Example:

```
wtx.SetBorder("CurrentImage", 12);
```

SetColor [Integer] *Element [String], Color [OLE Color]* - Method.

This method is only applicable to two elements.

For "CurrentImage", this sets the colour of the border around the displayed image.

For "Thumbnails" this sets the colour used to highlight the currently selected image.

The parameter *Color* is of data type OLE Color. When used in a Javascript application this is effectively an integer value and can be represented in hexadecimal notation with six digits where the first two digits indicate blue, the next two digits indicate green and the last two digits indicate red.

Example: Set the border for the main image area to red:

```
wtx.SetColor("CurrentImage", 0x0000FF);
```

HideElement [Integer] *Element [String]* - Method.

Hides the element. Some elements cannot be hidden as the interface is unusable without them, i.e., the Acquire button and the control panel. If the thumbnails area is hidden, only single images can be processed.

Example:

```
wtx.HideElement("Thumbnails");
```

ShowElement [Integer] *Element [String]* - Method.

Shows the element.

SetThumbnails *Width [Integer], Height [Integer], Rows [Integer]* - Method.

Sets the width and height of the Thumbnails element as well as the number of rows of thumbnails, simultaneously.

This method is the equivalent of calling *SetWidth* and *SetHeight* for the Thumbnails element and setting a value for *ThumbRows*. It is recommended that this single command should be used to configure the thumbnails area otherwise the individual commands could fail if called in the wrong sequence. For example, an increased number of rows may be rejected if set before providing sufficient height.

SetComboBoxItems [Integer] *Element [String], Count [Integer]* - Method.

This sets the number of drop-down items in one of the comboboxes ComboBox1 to ComboBox5.

Example: Set ComboBox1 to have five drop-down items:

```
wtx.SetComboBoxItems("ComboBox1", 5);
```

ComboBoxIndex [Integer] *Element [String]* **-** Read/Write Property.

Returns the index number of the currently selected option of a combobox, or allows the currently selected option to be preset.

Example: Set ComboBox1 to preselect the 2nd item in the list:

```
wtx.ComboBoxIndex("ComboBox1") = 2;
```

5.3. List of Available Elements

Element Name	Description	Hide	Container	Left	Top	Width	Height
Main Interface Elements							
CurrentImage	Main image area showing currently selected image	Yes		8	8	450	450
Thumbnails	Displays thumbnails of all images and navigation bar	Yes		8	464	777	121
ControlPanel	Tabbed panel containing all control elements	No		472	8	313	449
Control Panel Elements							
Tab1, Tab2,.., Tab5	Tabs on the control panel	Use ControlPanelTabs property					
Frame1	Frame (device settings in default configuration)	Yes	Tab1	8	192	289	217
Frame2	Frame (single page / all pages selection in default configuration)	Yes	Tab2	8	16	289	81
Frame3	Frame (edit buttons in default configuration)	Yes	Tab2	8	104	289	161
Frame4	Frame (upload controls in default configuration)	Yes	Tab2	8	272	289	137

Element Name	Description	Hide	Container	Left	Top	Width	Height
Frame5	Frame (arrange images in default configuration)	Yes	Tab3	8	16	289	137
Frame6, Frame7,.., Frame10	Spare frames (not used in default configuration)	Yes	Tab4	Yes	Yes	Yes	Yes
Label1, Label2,.., Label10	Spare labels (not used in default configuration)	Yes	Tab4	Yes	Yes	No	No
InputText1, InputText2,.., InputText10	Custom text input boxes (not used in default configuration)	Yes	Tab4	Yes	Yes	89	No
ComboBox1, ComboBox2,.., ComboBox5	Spare comboboxes or drop-down lists (not used in default configuration)	Yes	Tab4	Yes	Yes	89	No
ButtonAcquire	Acquires one or more images from a TWAIN device	No	Tab1	8	64	81	33
InputSelect	Combobox to select device from a list	Yes	Tab1	8	24	177	No
CheckInterface	Checkbox to select whether or not to use the default interface of the device driver	Yes	Tab1	104	64	185	No
CheckMultiImage	Checkbox to select whether or not more than one image can be acquired at one time	Yes	Tab1	104	96	185	No
CheckCloseInterface	Checkbox to select whether or not the device driver interface will be closed automatically after acquiring an image or images	Yes	Tab1	104	128	185	No

Element Name	Description	Hide	Container	Left	Top	Width	Height
CheckDuplex	Checkbox to select whether or not duplex scanner will be enabled if available	Yes	Tab1	104	160	185	No
CheckADF	Checkbox to select whether or not an automatic document feeder will be used	Yes	Tab4	Yes	Yes	185	No
LabelPixelType	Label for InputPixelType	Yes	Frame1	16	24	No	No
InputPixelType	Combobox to select the pixel type (colour depth)	Yes	Frame1	80	24	169	No
LabelResolution	Label for InputResolution	Yes	Frame1	16	72	No	No
InputResolution	Combobox or text box to select the resolution	Yes	Frame1	80	72	57	No
LabelResolutionUnits	Label for InputResolutionUnits	Yes	Frame1	144	72	No	No
InputResolutionUnits	Combobox to select the units of measure for resolution	Yes	Frame1	192	72	57	No
LabelPageSize	Label for InputPageSize	Yes	Frame1	16	120	No	No
InputPageSize	Combobox to select the page size	Yes	Frame1	80	120	129	No
LabelCustomSize	Label for custom page size controls	Yes	Frame1	16	152	No	No
InputWidth	Text box to enter page width	Yes	Frame1	16	176	65	No
LabelBy	Label between the input boxes for width and height. Shows 'X' by default	Yes	Frame1	96	176	No	No

Element Name	Description	Hide	Container	Left	Top	Width	Height
InputHeight	Text box to enter page height	Yes	Frame1	120	176	65	No
InputSizeUnits	Combobox to select the units of measure for page size	Yes	Frame1	200	176	57	No
OptionPages	Option buttons to select single page or all pages for processing	Yes	Frame2	16	24	185	No
LabelRotate	Label above rotate buttons	Yes	Frame3	16	24	No	No
ButtonRotateCW	Button to rotate an image, or all images, by 90° clockwise	Yes	Frame3	32	48	65	25
ButtonRotateCCW	Button to rotate an image, or all images, by 90° counter-clockwise	Yes	Frame3	112	48	65	25
ButtonRotate180	Button to rotate an image, or all images, by 180°	Yes	Frame3	192	48	65	25
LabelEdit	Label above edit buttons	Yes	Frame3	16	88	No	No
ButtonDeskew	Button to deskew an image, or all images	Yes	Frame3	32	120	65	25
ButtonDespeckle	Button to despeckle an image, or all images	Yes	Frame3	112	120	65	25
ButtonCrop	Button to crop an image	Yes	Frame3	192	120	65	25
ButtonAddText	Button to add text to an image	Yes	Tab4	Yes	Yes	65	25
InputAddText	Text box for entering string to be added using ButtonAddText	Yes	Tab4	Yes	Yes	89	No

Element Name	Description	Hide	Container	Left	Top	Width	Height
LabelFilename	Label for InputFilename	Yes	Frame4	16	24	No	No
InputFilename	Text box for entering name of file for upload	Yes	Frame4	16	48	89	No
ButtonUpload	Button to upload an image, or all images, to a web server	Yes	Frame4	120	32	65	25
ButtonAbort	Button to abort an upload while in progress	Yes	Frame4	200	32	65	25
ButtonSaveLocal	Button to save an image, or all images, on the local system (not used in default configuration)	Yes	Tab4	Yes	Yes	65	25
ProgressBar	Progress bar showing progress of image uploading	Yes	Frame4	16	104	257	37
ButtonClear	Button to delete all images	Yes	Frame5	16	32	81	33
ButtonDelete	Button to delete one image	Yes	Frame5	128	32	81	33
ButtonMoveUp	Button to move current image up by one page	Yes	Frame5	16	80	81	33
ButtonMoveDown	Button to move current image down by one page	Yes	Frame5	128	80	81	33
ButtonCustom	Button to run a script in the OnCustomClick event procedure (not used in default configuration)	Yes	Tab4	Yes	Yes	65	25

6. Troubleshooting

Problems encountered in running this software usually fall into two categories. These are problems with loading the clientside control in the browser, and problems uploading files to the server.

The most common issues in each of these categories are described here.

6.1. Problems Loading WebTwainX in the Browser

If the ActiveX control WebTwainX is not visible in the browser page, check the following list of likely causes.

Browser Compatibility

WebTwainX can only be used in a browser that supports the use of ActiveX controls. At the current time, the only major browser that supports this technology is Internet Explorer. It is not possible to use it in other browsers, e.g., Firefox.

Browser Security Settings

The loading of ActiveX controls is restricted in Internet Explorer to ensure that malicious software cannot be loaded without the user knowing about it. Both the trial and full versions of WebTwainX are digitally signed by Ciansoft, so the security settings of Internet Explorer must be set to allow signed ActiveX controls to load with prompting.

The actual behaviour will vary depending on the browser version, but usually a bar will appear at the top of the browser window to advise that some content has been blocked. Clicking on this bar gives the option to install the control, after which a dialogue will be displayed confirming that the control is signed by Ciansoft. The security certificate can be viewed if desired.

The Microsoft Licence Manager also has to be allowed to load, and this may generate similar warning messages that have to be accepted.

Licence File Not Found

To run WebTwainX, the browser must be able to read the licence package file (.lpk) held on the server. This should normally be placed in the same location on the server as the web page containing the control, and the control itself. If WebTwainX will not load and all security settings are correct, check that the licence package file is present and is correctly referenced in an <OBJECT> tag in the HTML code.

6.2. Problems Uploading to the Server

Many problems with uploading are due to issues regarding installation and registration of the server components, or permissions on files and directories.

It is important to be able to receive meaningful error messages when diagnosing problems on the server, but in recent versions of Windows, the default behaviour is for the server not to generate error messages and for the browser not to display them.

On the server, in IIS Manager, open ASP and under "Debugging Properties" set "Send Errors to Browser" to True.

In the browser, under the Advanced tab of Internet Options, turn off the "Show friendly HTTP error messages" option.

To confirm that the component is correctly installed and working, use a simple script to create an instance of the component and then write out the version information to the browser.

For ASP:

```
<%
Set Upload = Server.CreateObject("WebTwainXUploadASP.FileSave")
Response.Write Upload.Version
%>
```

For ASP.NET:

```
<%@ Page language="vb" debug="true" %>
<%@ Import NameSpace = "WebTwainXUploadNet" %>
<%
Dim Upload As New UploadClass
Response.Write(Upload.Version)
%>
```

These scripts must be called directly from the browser, not by uploading from the WebTwainX clientside control. If the version information is displayed, the component is correctly installed and working. If an error message is shown, check the following possibilities. The exact wording of error messages may vary with different versions of Windows and IIS.

WebTwainXUploadASP Not Registered

In ASP, if WebTwainXUploadASP.dll has not been successfully registered, the error message will be:

> Server object, ASP 0177 (0x800401F3)
> Invalid class string

On recent versions of Windows, when running regsvr32.exe to register the dll, the command prompt from which regsvr32.exe is run must have been started using the "Run as Administrator" option.

WebTwainXUploadASP Access Denied

In ASP, the Internet Guest User must have Read and Execute permissions on WebTwainXUploadASP.dll. If it does not, the error message will be:

> Server object, ASP 0178 (0x80070005)
> The call to Server.CreateObject failed while checking
> permissions. Access is denied to this object.

Ciansoft.WebTwainXUploadNet.dll Access Denied

In ASP.NET, the ASP.NET machine account must have Read and Execute permissions on Ciansoft.WebTwainXUploadNet.dll. If it does not, a lengthy error message will appear which should include the line:

> Parser Error Message: Access is denied: 'Ciansoft.WebTwainXUploadNet'.

Access Denied when Saving Files

In ASP, the Internet Guest User, or in ASP.NET, the ASP.NET machine account must have Write permission on the directory where any files are to be saved. It is not easy to get a definitive error message returned from the server to indicate this problem, but if the upload component has been confirmed to be working correctly and files still cannot be saved, this should be checked.

7. Revision History

The current version of WebTwainX is 1.3

New in Version 1.1

Properties to pre-set device settings, e.g., PixelType, Resolution.
OnSelect event.
AbortUpload method.
FileNameExtension property.
PagesSelected property.

New in Version 1.2

ButtonSaveLocal & ButtonAddText elements.
MICR support.
Upload method.
ImageIndex property.
ButtonCustom element and OnCustomClick event.
TWAIN 2.1 compatibility.
OnAcquire event.
Clear method.
Custom InputText elements.

New in Version 1.3

SaveLocalSuccess and SaveLocalError properties.
OnSaveLocal event.
ImageCount property.
MaxUploadSize property.
SaveUploadReturnString property.
UseDuplex property.
PageSizeMode property.
TWAIN 2.2 compatibility.

8. Alphabetical Lists of Functions

8.1. Clientside Control, WebTwainX.ocx / WebTwainXTrial.ocx

8.2. ASP Component, WebTwainXUploadASP.dll

8.3. ASP.NET Component, Ciansoft.WebTwainXUploadNet.dll