



Developer's Guide

Atalasoftware WingScan 10.0

Contents

| | |
|---|-----------|
| Web Scanning | 1 |
| Basic Requirements | 2 |
| Getting Started with Web Capture | 3 |
| Web Capture Guide | 5 |
| Web Scanning Server Reference | 13 |
| Upload Sizes and Limits | 14 |
| Extending the WebCaptureRequestHandler | 16 |
| Extending the KicHandler | 19 |
| Connecting to Kofax Import Connector (KIC) Web Services | 20 |
| Configuring Kofax Import Connector (KIC) | 22 |
| Connecting to Sharepoint | 25 |
| Web Scanning Client Reference | 30 |
| Initializing the Control on the Client | 31 |
| Connecting to UI Controls | 34 |
| Handling Events | 35 |
| Handling Errors | 38 |
| Setting Scanning Options | 43 |
| Connecting to the Web Document Viewer | 49 |
| Using VirtualReScan (VRS) | 51 |
| Testing Your Application | 52 |
| Troubleshooting Web Scanning Problems | 53 |
| Uninstalling ActiveX Control and Plugin | 56 |
| Client API Reference | 58 |
| Web Document Viewer | 61 |
| Web Document Viewer Overview | 62 |
| Web Document Viewer Guide | 64 |

Web Scanning

The following sections describe the WingScan web scanning controls and available back-end handlers.

- Basic Requirements** **2**
- Getting Started with Web Capture** **3**
- Web Capture Guide** **5**

Basic Requirements

WingScan Web Scanning has the following requirements:

On the Client

- Windows XP, Windows Vista, or Windows 7.
- A scanning device with a working TWAIN driver.
- 32-bit Microsoft Internet Explorer 7, 8, 9, Mozilla Firefox 4.0 or later, or Google Chrome
- On Internet Explorer, the EZTwainX Scanning Control must be allowed to install and run, and must not be disabled.
- On Firefox and Chrome, the WingScan Web Scanning Plugin must be installed or be allowed to install, and must not be disabled.

On the Server

- A writable location for files to be uploaded: either a location on the disk that the application can read and write to, or a database.
- **If You Use Our .NET Handlers:**
 - .NET 3.0 or greater is needed for WCF.
 - IIS 6 or 7.
 - Any WingScan assemblies you use are licensed in your request handler.

If you connect to Kofax Capture through Kofax Import Connector

- The Kofax Capture (KC), and Kofax Import Connector (KIC) must have at least a "KIC - Electronic Documents - Web Service interface" license.
- If connecting to KC through KIC, then KC must be version 9 or 10.

Getting Started with Web Capture

Once you have satisfied the [Basic Requirements](#) for web scanning, use the in-depth guide or follow the steps below to scan-enable a web application using the WingScan Web Scanning Control.

Use the in-depth guide

To get started with in-depth instructions, follow the [Web Capture Guide](#). The guide will step you through everything needed to build a page with functional web scanning. If instead you want a quick overview of the steps required, follow the basic steps below.

Basic Steps

1. Open an existing, or create a new ASP.Net web application project in Visual Studio, for example using the ASP.NET Empty Web Site template.
2. Add WingScan references that are appropriate to the version of .Net being used. At least the following will need to be added:

Atalasoftware.dll

Atalasoftware.WebControls

Atalasoftware.Lib

Atalasoftware.Shared

3. Copy the *web capture resources* into your application, placing them either in the root folder of your application, or in a subfolder added to the application root. The web capture resources are normally installed here:

```
C:\Program Files (x86)\Atalasoftware\WingScan 10.0\bin\2.0\x86\WebResources\WebCapture
```

As demonstrated in the WebCaptureDemo, whose source code is in:

```
C:\Users\<user>\AppData\Local\Atalasoftware\WingScan 10.0\WingScan Demos Source\C# Source\VS 08\WebCaptureDemo
```

4. Configure your web server to serve these resource files with the specified MIME types:

| File | MIME type | Purpose |
|------------------|--------------------------------|----------------------------|
| npWebCapture.xpi | application/x-xpinstall | Firefox plugin installer |
| WebCapture.crx | application/x-chrome-extension | Chrome extension installer |

5. By default the maximum upload size for IIS is 30 MB. Estimate the maximum size of upload your application might send to the server, and adjust the server's maximum upload size limit accordingly, as described in [Upload Sizes and Limits](#).
6. Create, or open the page (.html, .aspx, etc...) that will offer scanning and follow the instructions in [Initializing the Control on the Client](#).
7. Add the UI controls to control scanning and importing, as described in [Connecting to UI Controls](#).
8. Add a generic server-side handler to your project by [Extending the KicHandler](#) or [Extending the WebCaptureRequestHandler](#).
9. In your project, or on your server's filesystem, create a folder to receive uploaded images e.g. 'atala-capture-upload'
10. Modify the web.config to point to the upload folder.

Chapter 2

11. To make sure everything works, follow our suggestions for [Testing Your Application](#).
12. Deploy and support. You may find [Troubleshooting Web Scanning Problems](#) useful.

Web Capture Guide

This step-by-step guide is designed to bring you through all the steps of creating a new capture-enabled web project. Topics include adding the document viewer and scanning controls to your web page, and handling uploaded content on the server. Several steps will contain cross-references to other topics with more detailed information.

This guide is intended to be followed exactly, but it is not intended to give you a solution that is ready to deploy. Once you have succeeded building the example project, you can begin modifying it to fit your organization.

Setting up a new project

A capture-enabled web application requires these basic elements:

- A client-side ASPX page containing the web scanning controls and document viewer.
- A server-side ASHX handler for the Web Document Viewer.
- A server-side ASHX handler for the Web Capture back end.
- WebCapture and WebDocumentViewer resources files.
- An upload location for scanned documents.

Start by creating a new ASP.NET Web Application in Visual Studio. For the purposes of this guide, it is assumed this project is called `BasicWebCapture`.

Visual Studio automatically gives you `Default.aspx` as a page, which we will use for placing the web scanning controls and viewer.

Adding assembly references

Add the following WingScan assemblies to your project:

- `Atalasoftware.Image.WebControls`
- `Atalasoftware.Shared`

In a default installation, these assemblies can be found in `C:\Program Files (x86)\Atalasoftware\WingScan 10.0\bin\2.0\x86`.

There may be further dependencies on any of the remaining WingScan assemblies. Include all WingScan assemblies in your project if there are problems resolving them.

Copy in WingScan resources

WingScan comes with two sets of web resources: `WebCapture` and `WebDocumentViewer`. In a default installation, these directories are located in `C:\Program Files (x86)\Atalasoftware\WingScan 10.0\bin\2.0\x86\WebResources`.

Copy the `WebCapture` and `WebDocumentViewer` directories into the root of your project.

Create the upload location

Create a new directory in the root of your project called `atala-capture-upload`. This is the default path that will be used for storing images uploaded by the web scanning controls.

If you need to change the location of the upload path (for example, to place it in a location outside of your document root), you can set an `atala_uploadpath` value in the `appSettings` section of either your `web.config` or `app.config`. Settings in `web.config` take precedence.

```
<appSettings>
  <add key="atala_uploadpath" value="c:\path\to\location"/>
</appSettings>
```

Adding the Web Document Viewer Handler

The Web Document Viewer handler is responsible for communicating with the Web Document Viewer embedded in your page, and is separate from the capture handler.

Add a new Generic Handler to your project. For the purposes of this guide, it is assumed this file will be called `WebDocViewerHandler.ashx`.

Change the class definition to extend `WebDocumentRequestHandler` (part of `Atalasoftware.Imaging.WebControls`). Your handler should resemble the following example.

```
C#
using Atalasoftware.Imaging.WebControls;

namespace BasicWebCapture
{
    public class WebDocViewerHandler : WebDocumentRequestHandler
    {
    }
}
```

There is no need for further modification to your handler.

Adding the Web Capture Handler

The Web Capture handler is responsible for handling file uploads from the scanning controls embedded in your page, and routing them to their next destination along with any necessary metadata. It is also responsible for supplying the scanning controls with the available content and document types, and status information.

For this guide, we will create a custom handler that provides a few static content and document types, and saves uploaded files to another location. Using this baseline, you can continue modifying the handler to suit your own document handling needs.

If your organization uses Kofax Import Connector (KIC) or Microsoft SharePoint for document management, WingScan ships with handlers to connect to these services. Check their respective topics for more information on how to use these handlers.

- For connecting to KIC, see: [Connecting to Kofax Import Connector \(KIC\) Web Services](#) and [Extending the KicHandler](#).
- For connecting to SharePoint, see: [Connecting to Sharepoint](#).

Creating your own handler

Add a new Generic Handler to your project. For the purposes of this guide, it is assumed this file will be called `WebCaptureHandler.ashx`.

The handler should be modified to extend from `WebCaptureRequestHandler` (part of `Atalasoftware.Imaging.WebControls.Capture`), and should not implement the `IHttpHandler` interface, as is done when a generic handler is first created. Instead your handler will need to override several methods of `WebCaptureRequestHandler`. Your handler should resemble the following example.

```

C#
using System;
using System.Collections.Generic;
using System.IO;
using System.Web;
using Atalasoftware.Imaging.WebControls.Capture;

namespace BasicWebCapture
{
    public class WebCaptureHandler : WebCaptureRequestHandler
    {
        protected override List<string> GetContentTypeList(HttpContext context)
        {
            // ...
        }

        protected override List<Dictionary<string, string>> GetContentTypeDescription
        (HttpContext context, String contentType)
        {
            // ...
        }

        protected override Dictionary<string, string> ImportDocument(HttpContext context,
        string filename,
            string contentType, string contentTypeDocumentClass, string
        contentTypeDescription)
        {
            // ...
        }
    }
}

```

The three stubs represent the minimum number of methods that must be implemented for basic functionality, but there are other methods available in the public API that can also have their behavior overridden, such as methods to generate IDs or query the status of documents. Refer to the accompanying object reference for the complete `WebCaptureRequestHandler` API.

GetContentTypeList

This method returns the collection of available content types that can be used to organize scanned and uploaded documents. Content types are the top-level organizational unit, and each one has its own collection of document types (also called document classes) below it.

For this example, `GetContentTypeList` will be implemented to return a fixed list of two types: Accounts and HR. In a real system, this would probably query a database or other data source instead. In the KIC and SharePoint handlers, this method queries the system for these values.

```

C#
protected override List<string> GetContentTypeList(HttpContext context)
{
    return new List<string>() { "Accounts", "HR" };
}

```

GetContentTypeDescription

This method returns a collection of data describing all the document types under a single content type. The return data is a list of dictionaries, where each dictionary contains a set of properties describing a single document type. In this example, the only property returned for a document type is its `documentClass`, which serves as its name.

C#

```
protected override List<Dictionary<string, string>> GetContentTypeDescription(HttpContext
context, String contentType)
{
    switch (contentType)
    {
        case "Accounts":
            return CreateDocumentClassDictionaryList(new string[] { "Invoices", "Purchase
Orders" });
        case "HR":
            return CreateDocumentClassDictionaryList(new string[] { "Resumes" });
        default:
            return base.GetContentTypeDescription(context, contentType);
    }
}

private List<Dictionary<string, string>> CreateDocumentClassDictionaryList(string[] docList)
{
    List<Dictionary<string, string>> list = new List<Dictionary<string, string>>();
    foreach (var doc in docList)
    {
        Dictionary<string, string> d = new Dictionary<string, string>();
        d["documentClass"] = doc;
        list.Add(d);
    }
    return list;
}
```

A helper method is provided to produce the actual list of document types, while `GetContentTypeDescription` switches on a given content type to determine what document types should be included in the list. As with content types, it is expected that this data will originate from another data source, instead of being hard-coded.

ImportDocument

This method is responsible for actually moving a document and its metadata to its real destination, which could be a directory, database, or system like KIC and SharePoint.

C#

```
protected override Dictionary<string, string> ImportDocument(HttpContext context, string
filename,
    string contentType, string contentTypeDocumentClass, string contentTypeDescription)
{
    string docId = Guid.NewGuid().ToString();
    string importPath = @"C:\DocumentStore";

    importPath = Path.Combine(importPath, contentType);
    importPath = Path.Combine(importPath, contentTypeDocumentClass);
    importPath = Path.Combine(importPath, docId + "." + Path.GetExtension(filename));

    string uploadPath = Path.Combine(UploadPath, filename);

    File.Copy(uploadPath, importPath);

    return new Dictionary<string, string>()
    {
        { "success", "true" },
        { "id", docId },
        { "status", "Import succeeded" },
    };
}
```

In this example, imported documents are copied into a directory tree rooted at C:\D-documentStore, using the content type and document class as subdirectories for organizing files. The imported file is copied and given a new name based on a GUID, which is also passed back to the client in the "id" field of a dictionary. The id could be used by the client to query the handler at a future time for the status of the imported document, but this functionality is not included in the guide.

Setting up the scanning controls and viewer

The setup for web scanning just requires placing some JavaScript, CSS, and HTML into your page. The page itself could be HTML, ASPX, JSP, or anything else, as the client-side technology is not directly tied to .NET or IIS. For this guide however, we will update the document Default.aspx, which was originally included in the new web project.

Include the web resources

Include the following script and link tags in your page's head section to include the necessary Web Document Viewer and Web Capture code and dependencies.

```
HTML
<!-- Script includes for Web Viewing -->
<script src="WebDocViewer/jquery-1.7.1.min.js" type="text/javascript"></script>
<script src="WebDocViewer/jquery.easing.1.3.js" type="text/javascript"></script>
<script src="WebDocViewer/jquery-ui-1.8.14.custom.min.js" type="text/javascript"></script>
<script src="WebDocViewer/atalaWebDocumentViewer.js" type="text/javascript"></script>

<!-- Style for Web Viewer -->
<link href="WebDocViewer/atalaWebDocumentViewer.css" rel="Stylesheet" type="text/css" />

<!-- Script includes for Web Capture -->
<script src="WebCapture/atalaWebCapture.js" type="text/javascript"></script>
```

Configure the controls

The web scanning and web viewing controls need to be initialized and configured to set up connections to the right handlers, specify behavior for events, and so forth. This can be done with another block of JavaScript, either included or pasted directly within your page's head somewhere below the included dependencies.

JavaScript

```

<script type="text/javascript">
  // Initialize Web Scanning and Web Viewing
  $(function() {
    try {
      var viewer = new Atalasoftware.Controls.WebDocumentViewer({
        parent: $('.atala-document-container'),
        toolbarparent: $('.atala-document-toolbar'),
        serverurl: 'WebDocViewerHandler.ashx'
      });

      Atalasoftware.Controls.Capture.WebScanning.initialize({
        handlerUrl: 'WebCaptureHandler.ashx',
        onUploadCompleted: function(eventName, eventObj) {
          if (eventObj.success) {
            viewer.OpenUrl("atala-capture-upload/" + eventObj.documentFilename);
            Atalasoftware.Controls.Capture.CaptureService.documentFilename =
eventObj.documentFilename;
          }
        },
        scanningOptions: { pixelType: 0 }
      });

      Atalasoftware.Controls.Capture.CaptureService.initialize({
        handlerUrl: 'WebCaptureHandler.ashx'
      });
    }
    catch (error) {
      alert('Thrown error: ' + error.description);
    }
  });
</script>

```

Note that the URL for the WebDocViewer handler is specified once and the URL for the Web-Capture handler is specified twice, since two capture services must be initialized.

There are several additional options and handlers that can be specified in the initialization routines for web scanning and viewing. See [Client API Reference](#) for the available handlers and options, and [Initializing the Control on the Client](#) for a more complete initialization example. The web scanning demos included with WingScan also include more complete examples.

This example represents the minimal configuration necessary for web scanning with an integrated document viewer.

Add the UI

Add the following HTML to your project to create a basic viewer UI. This includes the Web Document Viewer, drop-down boxes to choose scanners, content types, and document types, and buttons to drive the UI. See [Connecting to UI Controls](#) for more information on the available web scanning components that can be exposed in the UI. The web scanning demos included with WingScan also include more complete examples.

HTML

```

<p>Select Scanner:
  <select class="atala-scanner-list" disabled="disabled" name="scannerList" style="width:
22em">
    <option selected="selected">(no scanners available)</option>
  </select>
</p>
<p>Content Type:
  <select class="atala-content-type-list" style="width:30em"></select>
</p>
<p>Document Type:
  <select class="atala-content-type-document-list" style="width:30em"></select>
</p>
<p>
  <input type="button" class="atala-scan-button" value="Scan" />
  <input type="button" class="atala-import-button" value="Import" />
</p>
<div>
  <div class="atala-document-toolbar" style="width: 670px;"></div>
  <div class="atala-document-container" style="width: 670px; height: 500px;"></div>
</div>

```

Wrapping Up

Your project should be ready to deploy to an IIS server. It is also ready to run from Visual Studio, for testing purposes.

Web Server - MIME Types

The scanning control in WingScan is compatible with Internet Explorer, Mozilla Firefox, and Google Chrome. In order to work in Firefox and Chrome, you will need to make sure your web server is properly configured to deliver XPI and CRX extensions to users. Configure your server so that the files in the table below are served with the associated MIME types listed.

| File | MIME type | Purpose |
|------------------|--------------------------------|----------------------------|
| npWebCapture.xpi | application/x-xpinstall | Firefox plugin installer |
| WebCapture.crx | application/x-chrome-extension | Chrome extension installer |

For IIS servers, this can be configured in the IIS Manager, in the MIME Types section of your site.

It is not possible to make the Visual Studio integrated IIS server recognize the above extensions and MIME-types. If you test under Visual Studio, you will not be prompted to install missing extensions in Firefox or Chrome.

Web Server - Upload Size Limits

By default, IIS limits uploads to 30MB. Estimate the maximum upload size your application could generate, and adjust the server limits accordingly, as described in [Upload Sizes and Limits](#)

Troubleshooting

If you have difficulty getting this project to run, consider using a tool like Fiddler Web Debugger, which allows you to monitor the HTTP requests and responses that pass between the web scanning controls, and the handlers on the back-end. Exceptions in your handlers will present as 500 errors and will likely contain the exception information embedded in the

Chapter 2

response. Other errors in your handlers will present as JSON data in the response that does not contain the data you expect.

Remember, when implementing the web capture handler, all of the data returned from the methods you override is converted into an equivalent JSON representation. Examining the JSON is an easy way to verify outside of the debugger that you are returning the right data.

Client errors will usually present as JavaScript errors. Use your browser's equivalent of F12 tools to access the JavaScript console to check for errors. The most likely source of errors is not correctly including all of the necessary web resources, not initializing the controls correctly, or running your page in an incompatible browser.

Web Scanning Server Reference

The following sections cover the server-side handlers for displaying and processing scanned documents. This also includes sections on forwarding documents to remote services such as Kofax Import Connector and SharePoint.

| | |
|--|-----------|
| Upload Sizes and Limits | 14 |
| Extending the WebCaptureRequestHandler | 16 |
| Extending the KicHandler | 19 |
| Connecting to Kofax Import Connector (KIC) Web Services | 20 |
| Configuring Kofax Import Connector (KIC) | 22 |
| Connecting to Sharepoint | 25 |

Upload Sizes and Limits

By default, IIS limits uploads to 30MB. If your application may sometimes generate larger uploads, you will need to adjust this limit for the server, or at least for your web application.

Estimating Upload Sizes

The size of an upload is approximately the sum of the compressed sizes of the uploaded images x 4/3 (1.333).

The calculations below are for *images*. Remember that duplex scanning generates two images per page, minus any blank sides discarded by setting **discardBlankPages:true**.

Raw Uncompressed Image Size

Uncompressed image size in bytes = (width x DPI x height x DPI x depth) / 8

Where *depth* is 24 for color, 8 for grayscale, and 1 for B&W images.

Example, an 8.5" x 11" color page, scanned at 200 DPI:

$$(8.5 \times 200 \times 11 \times 200 \times 24) / 8 = 11,220,000 \text{ bytes } (\sim 11\text{MB})$$

Compression Ratios

Typical office documents in B&W will compress by ~10X. White space increases the compression, lots of text or detailed graphics of any kind decreases the compression. 50KB per compressed B&W image is not a bad average, 70KB is conservative.

Grayscale and color images will compress by 20X-30X, sometimes more. As with B&W, blank paper compresses more, detailed content compresses less.

For our example 8.5" x 11" color page scanned at 200 DPI, with a raw size of 11MB we estimate a compressed size in the range 374KB - 560KB.

Factor in Base64 Encoding

We multiply by 4/3 (1.33) because uploads are encoded in [Base64](#), which encodes 3 binary bytes as 4 text characters.

Adjusting The IIS Upload Limit

IIS, by default, limits any single upload to 30MB. If you attempt a larger upload, the server will (oddly) return a 404 error.

If you expect to upload larger files, you will need to increase this limit.

Edit web.config

(from <http://www.webtrenches.com/post.cfm/iis7-file-upload-size-limits>)

```
Add to web.config
<system.webServer>
  <security>
    <requestFiltering>
      <requestLimits maxAllowedContentLength="524288000"/>
    </requestFiltering>
  </security>
</system.webServer>
```

If you add the above code to the web.config file for your site, you can control the maximum upload size for your site. In many cases, the system.webServer node will already be in the file, so just add the security node within that.

Note that the `maxAllowedContentLength` is in BYTES not kilobytes.

You may also need to restart your Web site (not the whole server) to enable the setting.

Configure Interactively

The limit can also be changed interactively (quoting from the same blog post)

1. Open IIS 7 SnapIn
2. Select the website you want enable to accept large file uploads.
3. In the main window double-click 'Request filtering'
4. once the window is opened you may see on top a list of tabs e.g. file name extensions, rules, hidden segments and so on...
5. regardless of the tab you select, in the main window right-click and select "Edit Feature Settings" and modify the "Maximum allowed content length (bytes)"

In-Memory Limitation

Note that you should not expect WingScan to collect and upload a set of images that exceeds (approximately) 500MB of memory when uncompressed, whether the uploaded file is compressed or not. This corresponds to approximately 20 pages of 200 DPI full-color US Letter or A4 size. Grayscale images use 1/3 the space of color images, and B&W images use 1/24.

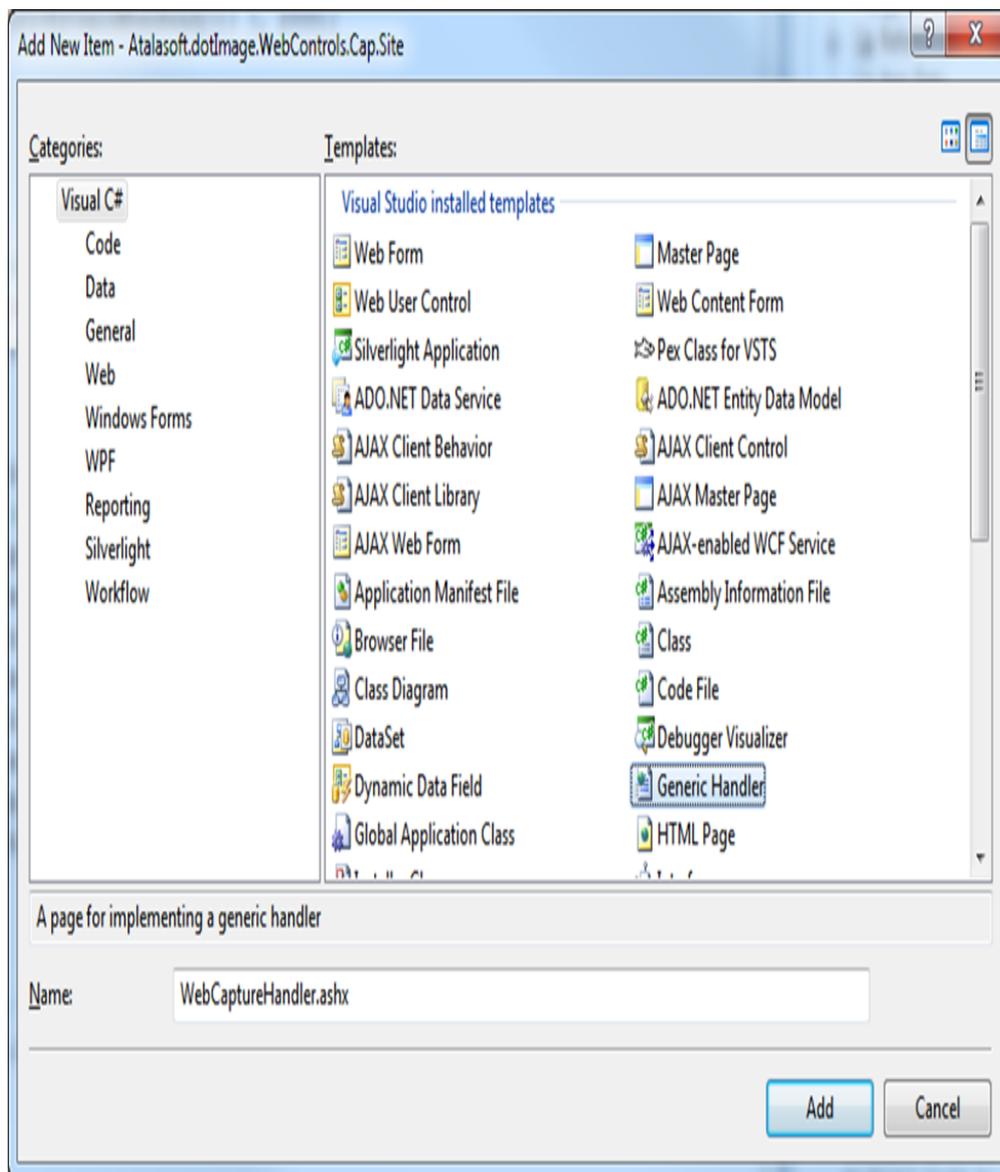
Server Timeouts

With larger uploads, you may need to also increase the [Params.serverTimeout: Integer](#) value, which is 20 seconds by default.

Extending the WebCaptureRequestHandler

Follow these instructions if you just need a basic capture uploader or need to interface with a custom connector. If connecting to KIC (Kofax Import Connector) Web Service is desired please follow the instructions in [Extending the KIC Handler](#), and [Connecting to KIC](#).

1. Open the project for the application in Visual Studio.
2. Right click on the root of the project, and select “Add-> New Item”
3. Select “Generic Handler” from the list of options, and give it a name.



4. Open the handler that was just created, and modify it to extend the Web-CaptureRequestHandler class that is included with WingScan. Change the generated class to look like the following:

Example (C#)

```
using System;
using System.Collections.Generic;
using System.Web;
using Atalasoftware.Imaging.WebControls.Capture;

namespace TheApplicationNamespace
{
    public class WebCaptureHandler : WebCaptureRequestHandler
    {

    }
}
```

5. Override `WebCaptureRequestHandler` methods that are relevant to the application. For example:

Example (C#)

```

using System;
using System.Collections.Generic;
using System.Web;
using Atalasoftware.Imaging.WebControls.Capture;

namespace MyWebCaptureApplication
{
    /// <summary>
    /// A test version of the handler that gets requests from the web capture client
    /// </summary>
    public class WebCaptureHandler : WebCaptureRequestHandler
    {
        protected override List<string> GetContentTypeList(HttpContext context)
        {
            return new List<string>(new string[]{"b1", "b2", "b3"});
        }

        protected override List<Dictionary<string, string>> GetContentTypeDescription
        (HttpContext context, String contentType)
        {
            if (contentType == "b1")
            {
                return new List<Dictionary<string, string>>
                (CreateDocumentClassDictionaryList(new string[] { "d1-1", "d1-2", "d1-3" }));
            }
            else if (contentType == "b2")
            {
                return new List<Dictionary<string, string>>
                (CreateDocumentClassDictionaryList(new string[] { "d2-1", "d2-2", "d2-3" }));
            }
            else if (contentType == "b3")
            {
                return new List<Dictionary<string, string>>
                (CreateDocumentClassDictionaryList(new string[] { "d3-1", "d3-2", "d3-3" }));
            }
            else
            {
                return base.GetContentTypeDescription(context, contentType);
            }
        }

        private List<Dictionary<string, string>> CreateDocumentClassDictionaryList
        (string[] docList)
        {
            List<Dictionary<string, string>> list = new List<Dictionary<string,
            string>>();
            foreach (var doc in docList)
            {
                Dictionary<string, string> d = new Dictionary<string, string>();
                d["documentClass"] = doc;
                list.Add(d);
            }
            return list;
        }
    }
}

```

Extending the KicHandler

1. Open the web application project Visual Studio.
2. Add a generic handler to the project.
3. Extend the handler that was just created with the KicHandler found in the Atalsoft.dotImage.WebControls.Capture namespace that in the Atalsoft.dotImage.WebControls.dll assembly.

For example:

Code Snippet

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace TheApplicationNamespace
{
    public class MyKicHandler : KicHandler
    {
    }
}
```

No other modifications are necessary.

Connecting to Kofax Import Connector (KIC) Web Services

These instructions are for configuring an application to connect to an existing KIC server.

For information on configuring KIC itself, see [Configuring Kofax Import Connector \(KIC\)](#).

Modifications Needed in the web.config to Connect to a KIC server

To connect to KIC a WCF endpoint, a binding must be added to the application's web.config, or app.config. In the provided example a standard basicHttpBinding will be used, but other appropriate binding types are possible choices to use as the WCF binding.

Setting The WCF EndPoint:

```
web.config Endpoint
<system.serviceModel>
  <client>
    <endpoint address="http://servername.domain.com:[http or https port]/soap/tsl"
binding="basicHttpBinding" bindingConfiguration="importBinding" contract="importPortType"
name="importPort" />
  </client>
</system.serviceModel>
```

Setting The Binding

HTTP:

```
web.config Binding
<system.serviceModel>
  <bindings>
    <basicHttpBinding>
      <binding name="importBinding" closeTimeout="00:01:00" openTimeout="00:01:00"
receiveTimeout="00:10:00" sendTimeout="00:01:00" allowCookies="false"
bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard" maxBufferSize="1655360"
maxBufferPoolSize="15242880" maxReceivedMessageSize="1655360" messageEncoding="Text"
textEncoding="utf-8" transferMode="Buffered" useDefaultWebProxy="true">
      <readerQuotas maxDepth="32" maxStringContentLength="256000" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
      <security mode="None">
        <transport clientCredentialType="None" proxyCredentialType="None" realm="" />
        <message clientCredentialType="UserName" algorithmSuite="Default" />
      </security>
    </binding>
  </basicHttpBinding>
</bindings>
</system.serviceModel>
```

HTTPS:**web.config Binding**

```
<system.serviceModel>
  <bindings>
    <basicHttpBinding>
      <binding name="importBinding" closeTimeout="00:01:00" openTimeout="00:01:00"
receiveTimeout="00:10:00" sendTimeout="00:01:00" allowCookies="false"
bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard" maxBufferSize="1655360"
maxBufferPoolSize="15242880" maxReceivedMessageSize="1655360" messageEncoding="Text"
textEncoding="utf-8" transferMode="Buffered" useDefaultWebProxy="true">
      <readerQuotas maxDepth="32" maxStringContentLength="256000" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384" />
      <security mode="Transport">
        <transport clientCredentialType="None" proxyCredentialType="None" realm="" />
        <message clientCredentialType="UserName" algorithmSuite="Default" />
      </security>
    </binding>
    <basicHttpBinding>
  </bindings>
</system.serviceModel>
```

Configuring Kofax Import Connector (KIC)

This is not meant to be a full set of instructions on installing, setting up, and maintaining a KIC (Kofax Import Connector) server, but instead meant to provide the minimum amount of configuration needed for the WingScan Web Scanning Control to successfully connect, and import into KIC.

For information on connecting to an already configured KIC server, see [Connecting to Kofax Import Connector \(KIC\) Web Services](#).

Required KIC License:

For the KIC webserver to accept documents imported from the WingScan assembly, a "KIC – Electronic Documents – Web Service interface"

license must be installed on your KIC server. To verify that the correct minimum license has been installed go to the Message Connector Monitor, which by default is located on the KIC server at <https://localhost:25086/file/index.html> where under the "Status->license" section this should be seen:

| Licensed Features | | | | | | | | |
|-------------------|------|-------------|------------|---------|--------|---------------------|------------|---------|
| Foip | SMTP | Web Service | Fax Server | Mailbox | Folder | Last Updated | Mode | Command |
| ● | ● | ● | ● | ● | ● | 2012-03-08 04:46:09 | KC License | |

Configuring the KIC Web Service:

The WingScan Web scanning control connects via KIC's web service via a server-side handler that extends the KicHandler found in the Atalsoft.dotImage.WebControls assembly. To configure KIC's web service start by going into the KIC Message Connector Configuration (Start->All Programs->Kofax->KIC Electronic Documents->Message Connector Configuration). Once in the message connector, go to the "General" section, and verify that the "Own Computer Name" field is filled in with the current server's domain qualified name. For example:

General

Own Fax Number Called Station Identification (CSI) for incoming fax calls. Should contain only digits, '+' and blanks

Operator Email Inbound messages that could not be imported into Kofax Capture are forwarded to this email address. Refer to "Email Outbound" configuration tab below. Leave empty to disable forwarding.

Email From This email address is displayed in the "From" field of messages forwarded to operator

Keep Failed Failed inbound messages are those not imported into Kofax Capture and also not forwarded to an operator. If selected, failed inbound messages are kept pending. These pending messages have to be handled and deleted manually, otherwise the storage will eventually run out of disk space.

Storage Size Size of disk space in MB reserved for storage of messages. Minimum size is 10 MB, maximum 64000 MB. 100

Storage File Storage file name and location common_appdata\Kofax\KIC-ED\MC\Storage\bin

Prefetched Messages Number of messages prefetched from mail boxes, folders and other passive inputs. Minimum number is 1, maximum 100. 1

Own Computer Name Domain or computer name where the solution will be installed (for SMTP and web-services)

Next go to the "Web-Service Input section. If only a HTTP based connection is desired set the HTTPS port to 0. For example,

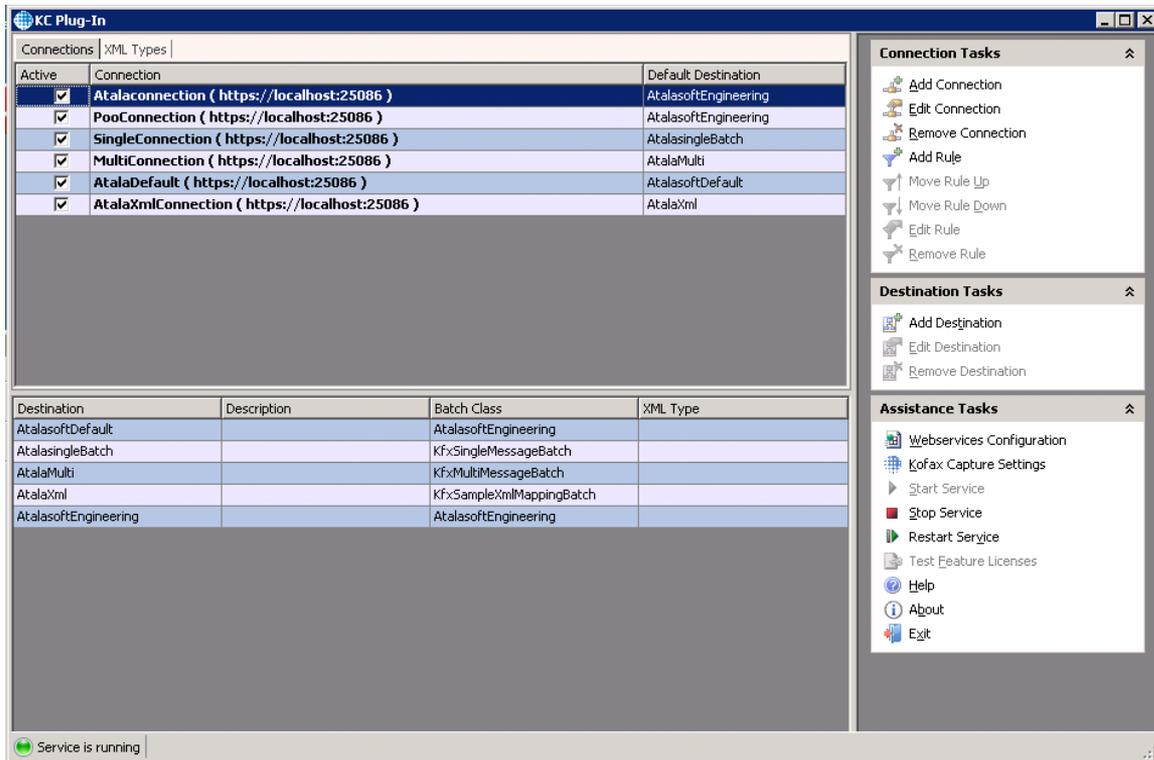
| Web-Service Input | | |
|-----------------------------|--|---|
| Local IP Address | <input type="text"/> | IP address of local interface used for web-service input. If empty, all local interfaces are used. |
| HTTP Port | <input type="text" value="25087"/> | Port used for web-service input without TLS (HTTP). (0 means disable) 25087 |
| HTTPS Port | <input type="text" value="0"/> | Port used for web-service input with TLS (HTTPS). (0 means disable) 25088 |
| KC Plugin URL | <input type="text" value="http://at01d031:8001/KC"/> | URL to KC plugin (only used for GetContentTypeList and GetContentTypeDescription web service calls) http://localhost:8001/KIC-Electronic-Documents |
| MC Cluster Enabled | <input type="checkbox"/> | Enable Message Connector Web Services Cluster mode |
| Local MC Cluster IP Address | <input type="text" value="127.0.0.1"/> | Local IP used for MC cluster internal communication |
| MC Cluster Port | <input type="text" value="25099"/> | Local TCP port for internal MC cluster communication 25099 |
| MC Cluster Members | <input type="text" value="127.0.0.1:25100"/> | List of IP addresses[:port] of other MC cluster members (separated by comma or blank) |

This will be the port which the endpoint in the applications web.config will point to. If HTTPS is desired, then enter the port which will be used. If HTTPS is enabled the HTTP port will not be able to be connected to, and the endpoint in the application's web.config will need to point at the URL using the HTTPS port. Once all of the desired changes to the KIC Message Connector have been made save, and restart the Message Connector service.



Configuring the Electronic Documents plugin

In the Kofax Capture (KC) Administration application, open the 'Electronic Documents->Configuration' window, and configure the necessary Connections, and Destinations.



Once finished stop, and start the service.

Testing the configuration:

To test that the KIC server has been minimally configured correctly in a browser either on the server, or at a client that might connect to the server enter the following URLs (all on one line of course):

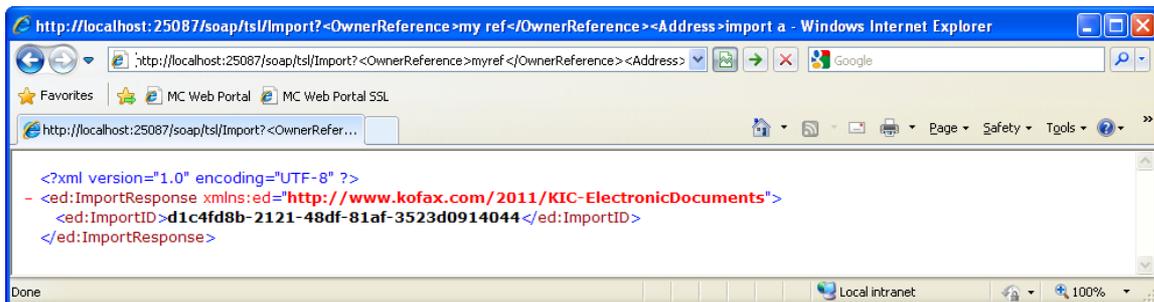
HTTP enabled webservice:

```
http://[kic_servername]:[http_port]/soap/tsl/Import?<OwnerReference>myref</OwnerReference>
<Address>importaddr</Address><Part><ContentType>text/plain</ContentType>
<Content><Text>hello</Text></Content></Part>
```

HTTPS enabled webservice:

```
https://[kic_servername]:[https_port]/soap/tsl/Import?<OwnerReference>myref</OwnerReference>
<Address>importaddr</Address><Part><ContentType>text/plain</ContentType>
<Content><Text>hello</Text></Content></Part>
```

If the KIC server is correctly configured the response should look like:



Connecting to Sharepoint

Requirements

The SharepointHandler will only connect to a Sharepoint 2010 site that has the Administrator Toolkit (<http://technet.microsoft.com/en-us/library/cc508849.aspx>) installed on it, and has setup the CMIS Connector (<http://technet.microsoft.com/en-us/library/ff934619.aspx>) that is included along with the Administrator toolkit.

Limitations of the SharepointHandler

- Sub-sites are not supported.
- Folder navigation is not currently supported, documents will only be imported into the top.
- Document libraries are currently the only type of Sharepoint list that importing a document into is supported for.
- Setting additional fields other than the title, and name for specific list ContentTypes is not supported.

Adding a Sharepoint Handler to a Project

These are the basic steps to add a Sharepoint handler to your application.

General Setup

- Create or open a web application project in Visual Studio.
- Add the Atalsoft.dotImage.WebControls reference.
- Add licensing information to the project.
- Make the necessary configuration changes to your web.config or app.config (see [Web-Config](#)).

Adding The Handler

- Add a new generic handler (.ashx) to the project.
- Extend the new handler from the SharepointHandler class.
- Add any additional authentication or handler logic if you need it (see [Authentication](#)).

Adding The Client

- Add the WebDocViewer client files and handler to your project.
- Modify the client .aspx to point to the correct WebDocViewer and Sharepoint handler URLs.

Refer to the included WebCaptureDemo for an already configured client and web.config. The demo's client .aspx file will need to be modified to point to the SharepointHandler for its capture handler instead of another capture service.

WebConfig and AppConfig Changes

In addition to the standard WCF elements, endpoints and bindings must be added for each of the CMIS services, and the Sharepoint List Service. In the examples below only a `basicHttpBinding` is shown. Use of a different type of binding is possible.

Adding The Endpoints

Choose an appropriate set of endpoints for CMIS. Endpoints vary by service and by authentication type. The following is the complete set of CMIS endpoints that must be set, using NTLM authentication. Endpoints are added to the <client> section of <system.serviceModel> in your configuration.

```

web.config Endpoints

<endpoint address="http://you.sp2010server.com/_vti_bin/CMIS/soap/DiscoveryService.svc/ntlm"
behaviorConfiguration="ImpersonationBehavior" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IDiscoveryServicePort"
contract="IDiscoveryServicePort" name="BasicHttpBinding_IDiscoveryServicePort"/>
<endpoint address="http://you.sp2010server.com/_vti_bin/CMIS/soap/VersioningService.svc/ntlm"
behaviorConfiguration="ImpersonationBehavior" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IVersioningServicePort"
contract="IVersioningServicePort" name="BasicHttpBinding_IVersioningServicePort"/>
<endpoint address="http://you.sp2010server.com/_vti_bin/cmisisoap/RepositoryService.svc/ntlm"
behaviorConfiguration="ImpersonationBehavior" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IRepositoryServicePort"
contract="IRepositoryServicePort" name="BasicHttpBinding_IRepositoryServicePort"/>
<endpoint address="http://you.sp2010server.com/_vti_bin/cmisisoap/ObjectService.svc/ntlm"
behaviorConfiguration="ImpersonationBehavior" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IObjectServicePort" contract="IObjectServicePort"
name="BasicHttpBinding_IObjectServicePort"/>
<endpoint address="http://you.sp2010server.com/_vti_bin/cmisisoap/NavigationService.svc/ntlm"
behaviorConfiguration="ImpersonationBehavior" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_INavigationServicePort"
contract="INavigationServicePort" name="BasicHttpBinding_INavigationServicePort"/>
<endpoint address="http://you.sp2010server.com/_vti_bin/CMIS/soap/ACLService.svc/ntlm"
behaviorConfiguration="ImpersonationBehavior" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IACLServicePort" contract="IACLServicePort"
name="BasicHttpBinding_IACLServicePort"/>
<endpoint address="http://you.sp2010server.com/_vti_
bin/CMIS/soap/MultiFilingService.svc/ntlm" behaviorConfiguration="ImpersonationBehavior"
binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IMultiFilingServicePort"
contract="IMultiFilingServicePort" name="BasicHttpBinding_IMultiFilingServicePort" />
<endpoint address="http://you.sp2010server.com/_vti_
bin/CMIS/soap/RelationshipService.svc/ntlm" behaviorConfiguration="ImpersonationBehavior"
binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IRelationshipServicePort"
contract="IRelationshipServicePort" name="BasicHttpBinding_IRelationshipServicePort" />
<endpoint address="http://you.sp2010server.com/_vti_bin/CMIS/soap/PolicyService.svc/ntlm"
behaviorConfiguration="ImpersonationBehavior" binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IPolicyServicePort" contract="IPolicyServicePort"
name="BasicHttpBinding_IPolicyServicePort" />

```

Additional authentication types are anonymous, basic, digest, and kerberos. Any of these can be specified by replacing ntlm in the endpoint address.

You will also need to specify the following endpoint for the Sharepoint Lists Service along with the others.

```

web.config Endpoints

<endpoint address="http://sp2010qa/_vti_bin/Lists.asmx"
behaviorConfiguration="ImpersonationBehavior" binding="basicHttpBinding"
bindingConfiguration="ListsSoap" contract="ListsSoap" name="ListsSoap"/>

```

Adding The Bindings

For each endpoint, specify a corresponding binding. All CMIS and Sharepoint bindings require an MTOM message encoding. The following block shows the binding to pair with the CMIS DiscoveryService endpoint. Bindings are added to the <bindings> section of <system.serviceModel> in your configuration.

web.config Bindings

```
<binding name="BasicHttpBinding_IDiscoveryServicePort" closeTimeout="00:01:00"
openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00" allowCookies="false"
bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard" maxBufferSize="65536"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536" messageEncoding="Mtom"
textEncoding="utf-8" transferMode="Buffered" useDefaultWebProxy="true">
  <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384"/>
  <security mode="TransportCredentialOnly">
    <transport clientCredentialType="Windows"/>
  </security>
</binding>
```

Using this binding as a template, create a binding for each of the following names, which correspond to endpoints.

- BasicHttpBinding_IDiscoveryServicePort
- BasicHttpBinding_IVersioningServicePort
- BasicHttpBinding_IRepositoryServicePort
- BasicHttpBinding_IObjectServicePort
- BasicHttpBinding_INavigationServicePort
- BasicHttpBinding_IACLServicePort
- BasicHttpBinding_IMultiFilingServicePort
- BasicHttpBinding_IRelationshipServicePort
- BasicHttpBinding_IPolicyServicePort

Apart from the `name` attribute, each of these bindings will have the same configuration unless you have a reason to customize. The `maxReceivedMessageSize` attribute may need to be adjusted depending on the amount of content that will be returned for a specific repository.

For the Sharepoint Lists Service endpoint, specify the following binding in addition to the others. This one has been configured for larger data transfer.

web.config Bindings

```
<binding name="ListsSoap" closeTimeout="00:01:00" openTimeout="00:01:00"
receiveTimeout="00:10:00" sendTimeout="00:01:00" allowCookies="false"
bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard" maxBufferSize="16777216"
maxBufferPoolSize="16777216" maxReceivedMessageSize="16777216" messageEncoding="Text"
textEncoding="utf-8" transferMode="Buffered" useDefaultWebProxy="true">
  <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
maxBytesPerRead="4096" maxNameTableCharCount="16384"/>
  <security mode="TransportCredentialOnly">
    <transport clientCredentialType="Windows"/>
  </security>
</binding>
```

Adding The Behaviors

Add the following behavior to the `<behaviors>` section of `<system.serviceModel>` in your configuration.

web.config Behaviors

```
<endpointBehaviors>
  <behavior name="ImpersonationBehavior">
    <clientCredentials>
      <windows allowedImpersonationLevel="Impersonation" />
    </clientCredentials>
  </behavior>
</endpointBehaviors>
```

Set The Location Of The Atala-upload Folder

By default, the client javascript api uploads scanned documents to a folder named `atala-upload-folder` in the root of the website. To change the location where scanned documents are stored, add the following to the `<appSettings>` section of your configuration.

web.config atala-upload path

```
<add key="atala_uploadpath" value="C:\explicit\path\to\upload\directory"/>
```

Authentication For Web Applications

If you are configuring a web application, you will need to make sure the following settings are in the `<system.web>` section of your web.config file.

web.config Settings

```
<authentication mode="Windows"/>
<identity impersonate="true" />
```

If you have advanced authentication requirements in your Sharepoint or IIS environment, you may need to modify or replace these settings to suit your organization.

Authentication

The available authentication options depend upon how Sharepoint and the web application containing the SharepointHandler are deployed.

Due to restrictions in the Windows security model, communicating with Sharepoint using client impersonation is not possible when the Sharepoint server and hosted web application reside on different servers. The delegating IIS server in the middle is not able to forward network requests of the originating client on to the remote Sharepoint server. In this environment, you are limited to the following options to successfully communicate between the client scanning application and Sharepoint.

Specify A User Account In Web.config

Specify the username and password of an ActiveDirectory user in the `<identity>` tag in your web.config. IIS will impersonate the specified user when making requests to the remote Sharepoint server.

web.config Settings

```
<identity impersonate="true" userName="DOMAIN\username" password="password" />
```

Override CreateAuthenticationSession In Your Handler

`AuthenticationSession` objects are used by the SharepointHandler to perform actions before and after making requests to Sharepoint. Overriding `CreateAuthenticationSession` allows you to specify what `AuthenticationSession` object should be used at those points. By default the SharepointHandler uses a

NullAuthenticationSession, which performs no actions. Another available type is the CredentialAuthenticationSession, which is constructed with the domain, username, and password of a Windows account, and instructs IIS to impersonate the specified user. This is similar to specifying an account in web.config, but the impersonation is only used while making Sharepoint requests, rather than being used across the entire website.

Example (C#): Handler with custom authentication

```
public class Handler : SharepointHandler
{
    protected override AuthenticationSession CreateAuthenticationSession()
    {
        return new CredentialAuthenticationSession("DOMAIN", "username", "password");
    }
}
```

It is also possible to write your own types by deriving from AuthenticationSession and implementing the abstract methods Enter and Leave, which setup and undo any changes in authentication. You may want to write your own AuthenticationSession if you need to impersonate a user that is dynamically fetched from your own user store, rather than specifying a single account to use for all requests.

Certificates

All certificates will be accepted when using a "Debug" build; otherwise a valid certificate may be required to successfully connect the handler to Sharepoint. For more information on setting up certificates and certificate stores, see: <http://msdn.microsoft.com/en-us/library/ff648360.aspx>.

Web Scanning Client Reference

The following sections cover all aspects of the WingScan Web Scanning Control. This includes adding the control to a web page, configuration, connecting to server-side handlers, and troubleshooting.

| | |
|--|-----------|
| Initializing the Control on the Client | 31 |
| Connecting to UI Controls | 34 |
| Handling Events | 35 |
| Handling Errors | 38 |
| Setting Scanning Options | 43 |
| Connecting to the Web Document Viewer | 49 |
| Using VirtualReScan (VRS) | 51 |
| Testing Your Application | 52 |
| Troubleshooting Web Scanning Problems | 53 |
| Uninstalling ActiveX Control and Plugin | 56 |
| Client API Reference | 58 |

Initializing the Control on the Client

On the page of your web application that will support scanning, you need to include the web capture javascript, and initialize scanning and upload/import.

Including WebCapture Javascript

Add the needed includes in the `<head/>` section of the document, like this:

```
<script src="jquery-1.7.1.min.js" type="text/javascript"></script>  
<script src="atalaWebCapture.js" type="text/javascript"></script>
```

If you placed the web capture resources in a subfolder under your application, you will need to modify the src attribute to the appropriate relative path.

Initializing

There are two parts of the control that need to be initialized, both can be initialized in the same script tag.

Initializing Scanning

Scanning is initialized with a call to:

```
Atalasoftware.Controls.Capture.WebScanning.initialize({params})
```

This function takes a comma-separated list of arguments including the URL of the handler used on the server, event handlers, scanning options sent to the control, and error handling for the client. All of the arguments are optional except the URL of the server handler.

See [Client API Reference](#) for details.

Initializing the KIC (Kofax Import Connector) Connection

The connection to the KIC server is initialized with a call to:

```
Atalasoftware.Controls.Capture.CaptureService.initialize({params})
```

This needs to be called in addition to the `WebScanning.initialize` function to populate any client UI controls with KIC `contentTypes`, and `contentTypeDescriptions`. It requires a handler argument, and accepts optional custom error handlers. When no selection dropdowns, or other selection UI is desired values for the required `contentType`, and `contentTypeDescriptionName` are also set in the parameter list.

Example

The following example script shows both these objects being initialized:

Code Snippet

```

<script type="text/javascript">
  // Initialize Web Scanning and Web Viewing
  $(function() {
    try {

      Atalasoftware.Controls.Capture.WebScanning.initialize({
        handlerUrl: 'KicWebCaptureHandler.ashx',

        onScanError: function(msg, params) { appendStatus(msg); },
        onScanStarted: function(eventName, eventObj) { appendStatus('Scan Started');
      },
        onScanCompleted: function(eventName, eventObj) { appendStatus('Scan
Completed: ' + eventObj.success); },

        onUploadError: function(msg, params) { appendStatus(msg); },
        onUploadStarted: function(eventName, eventObj) { appendStatus('Upload
Started'); },
        onUploadCompleted: function(eventName, eventObj) {
          appendStatus('Upload Completed: ' + eventObj.success);
          if (eventObj.success) {
            viewer.OpenUrl('atala-capture-upload/' + eventObj.documentFilename);
          }
        },
        scanningOptions: { pixelType: 0 }
      });

      Atalasoftware.Controls.Capture.CaptureService.initialize({
        handlerUrl: 'KicWebCaptureHandler.ashx',
        onError: function(msg, params) { appendStatus(msg + ': ' +
params.statusText); }
      });
    }
    catch (error) {
      //Do something with the error caught. Default is to just go
      //to the javascript error console in the browser.
    }
  });
</script>

```

Example when no contentType, or contentTypeDescription UI is desired:

Code Snippet

```

<script type="text/javascript">
  // Initialize Web Scanning and Web Viewing
  $(function() {
    try {
      Atalasoftware.Controls.Capture.WebScanning.initialize({
        handlerUrl: 'KicWebHandler.ashx',

        onScanError: function(msg, params) { appendStatus(msg); },
        onScanStarted: function(eventName, eventObj) { appendStatus("Scan Started");
      },

      onScanCompleted: function(eventName, eventObj) {
        appendStatus("Scan Completed: " + eventObj.success); },

      onUploadError: function(msg, params) { appendStatus(msg); },
      onUploadStarted: function(eventName, eventObj) { appendStatus("Upload
Started"); },
      onUploadCompleted: function(eventName, eventObj) {
        appendStatus("Upload Completed: " + eventObj.success);
        if (eventObj.success) {
          appendStatus("atala-capture-upload/" + eventObj.documentFilename);
          viewer.OpenUrl("atala-capture-upload/" + eventObj.documentFilename);
          Atalasoftware.Controls.Capture.CaptureService.documentFilename =
eventObj.documentFilename;
        }
      },
      scanningOptions: { pixelType: 1}
    });

    Atalasoftware.Controls.Capture.CaptureService.initialize({
      handlerUrl: 'KicWebHandler.ashx',
      //The required BatchClassName.
      contentType: 'AtalasoftwareEngineering',
      //The ContentTypeDescriptionName must be in the form of
      //'DocumentClassName / FormType'.
      contentTypeDescriptionName: 'PointOfOrigin / ClaimForms',
      onError: function(msg, params) { appendStatus(msg +": " +
params.statusText); },
      onImportCompleted: function(params) { appendStatus(params.id +": " +
params.status ); },
      onTrackStatusReceived: function(params) {appendStatus("Import status: "+
params); }
    });
  }
  catch (error) {
    appendStatus("Thrown error: " + error.description);
  }
});
</script>

```

Connecting to UI Controls

The Web Scanning control automatically finds and connects to UI controls using their `class=""` identifiers, so it is sufficient for you to add, lay out and style the UI controls required by your application, and assign the appropriate classes to those controls.

The four classes are: `atala-scan-button`, `atala-scanner-list`, `atala-content-type-list` and `atala-content-type-document-list`.

Examples:

1. A "Scan" button:

```
<input type="button" class="atala-scan-button" value="Scan" />
```

This button will automatically be enabled when scanning is possible, and disabled otherwise. When the user clicks this button, a scan is initiated with current scanner and document selections.

2. Scanner Device List:

```
<select class="atala-scanner-list" disabled="disabled" name="scannerList" style="width: 194px">
  <option selected="selected">(no scanners available)</option>
</select>
```

This control is loaded with the list of available TWAIN devices, and the current visible selection will be used when a scan is initiated.

If scanning is not possible or there are no scanners available, this control will be disabled.

3. KIC Content Types:

```
<select class="atala-content-type-list" style="width:385px"></select>
```

This control is automatically loaded with the list of available content types provided by the KIC server, and the current visible selection is used when an import is initiated.

If a connection cannot be established to the KIC server, this control is disabled.

4. KIC Content Type Descriptions:

```
<select class="atala-content-type-document-list" style="width:385px"></select>
```

This control is automatically loaded with the list of available content type descriptions as provided by the KIC server, and the current visible selection is used when a scan is initiated.

If a connection cannot be established to the KIC server, this control is disabled.

5. KIC Import button:

```
<input type="button" class="atala-import-button" value="Import" />
```

This button is automatically enabled if KIC import is possible, and is disabled otherwise. When the user clicks it, a KIC import (of the last scanned document) is initiated.

6. KIC Track Import button:

```
<input type="button" class="atala-track-import-button" value="Track Import" />
```

When the user clicks it, the status of the last import is returned.

*One should also note, that any "button" that has a `type="submit"` will create an empty POST that will override any POST or GET that the web scanning control sends.

Handling Events

The Atalasoftware.Controls.WebScanning control has the following events that can be used in the client:

- onScanError
- onScanStarted
- onImageAcquired
- onScanCompleted
- onScanClientReady
- onUploadError
- onUploadStarted
- onUploadCompleted

To use one, some, or all of the events add them to the Atalasoftware.Controls.WebScanning.initialize method's argument list. See [Client API Reference](#).

An example where each event is used:

```

Initializing WebScanning with Event Handlers

try {
    Atalasoftware.Controls.Capture.WebScanning.initialize({
        handlerUrl: 'TestCaptureHandler.ashx',

        onScanError: function(msg, params) { appendStatus(msg); },
        onScanClientReady: function() { appendStatus('Scan-Client Ready'); },
        onScanStarted: function(eventName, eventObj) { appendStatus('Scan Started'); },
        onImageAcquire: function(eventName, imageProxy) { appendStatus('Image Acquired'); },
        onScanCompleted: function(eventName, eventObj) {
            appendStatus('Scan Completed: ' + eventObj.success);
        },
        onUploadError: function(msg, params) { appendStatus(msg); },
        onUploadStarted: function(eventName, eventObj) { appendStatus('Upload Started'); },
        onUploadCompleted: function(eventName, eventObj) {
            appendStatus('Upload Completed: ' + eventObj.success);
            if (eventObj.success) {
                viewer.OpenUrl('atala-capture-upload/' + eventObj.documentFilename);
            }
        }
    });
}
catch (error) {
    appendStatus("WebScanning initialization error: " + error.description);
}

```

Handler: onScanError(msg, params)

See [Handling Errors](#)

Handler: onScanClientReady()

See [Handling Errors](#)

Handler: onScanStarted(eventName, eventObj)

Called when scanning starts.

Note: Always followed by a call to onScanCompleted, even if the scan fails or is aborted.

Handler: **onImageAcquired(eventName, imageProxy)**

This handler will be called during scanning each time an image is received from the scanner and processed by WingScan.

Note: If blank images are being discarded (the `discardBlankPages` scanning option has been set to true), any image that is determined to be 'blank' will be discarded during post-processing. This handler is not called for such images.

The 2nd parameter is a 'proxy' object representing the acquired image, with a limited set of properties and methods that can be used inside the handler.

Note: Do not retain the proxy object outside the `onImageAcquired` handler, it is not valid after the handler returns.

imageProxy properties and methods

ImageProxy.discard

If the handler sets **imageProxy.discard** to true, the image will be discarded when the handler returns. Use this feature if you are uploading or otherwise disposing of each incoming image yourself, and do not want WingScan to collect and upload all the scanned images at the end of the scan job.

ImageProxy.asBase64String(fmt)

This method returns a base-64 encoded file containing the just-received image, in the file-format specified by the `fmt` parameter. The `fmt` parameter must be either the string 'tif' or 'jpg'. Note that 'jpg' won't work if you are receiving B&W images, because JPEG files can only hold grayscale or RGB color images.

This method is useful if you want to store or upload each scanned image separately as it arrives.

Handler: **onScanCompleted(eventName, eventObj)**

Called when scanning ends, successfully or otherwise.

The **eventObj** has a property **success**. If it is true, the scan completed without error.

If **eventObj.success** is false, the scan was not fully successful, and there will be a string with more information in **eventObj.error.message**.

Usually when scanning fails, the `onScanError` handler will have already been called with a specific error message.

Handler: **onUploadStarted(eventName, eventObj)**

Called when an upload begins.

Handler: **onUploadError(msg, params)**

Called when an error is detected during upload to the server.

The `msg` parameter will be one of the following:

- **Atalasoftware.Controls.Capture.Errors.ajax** - could not create/initialize the XMLHttpRequest object.
- **Atalasoftware.Controls.Capture.Errors.serverNotResponding** - connection to the server timed out.

- **Atalsoft.Controls.Capture.Errors.uploadError** - the params object will contain three properties: **responseStatus**, **response**, and **handlerUrl**.

Handler: onUploadCompleted(eventName, eventObj)

Called when an upload completes, whether successfully or not.

If the upload was successful, **eventObj.success** is true, and **eventObj.documentFilename** contains the unqualified name of the file in the upload directory on the server.

If the upload failed for some reason, **eventObj.success** is false. In this case, **onUploadError** will have been called to report the error.

Handling Errors

By default all errors are sent to the javascript console in the browser. However, you can override this by specifying an error-handling function in the parameters to `Atalasoftware.Controls.WebScanning.initialize` and `Atalasoftware.Controls.CaptureService.initialize`. See [Client API Reference](#).

This example shows the basic technique of specifying error-handling functions. There is a longer code example at the end of this section.

```

JavaScript
$(function() {
  try {
    Atalasoftware.Controls.Capture.WebScanning.initialize({
      handlerUrl: 'TestCaptureHandler.ashx',

      onScanError: function(msg, params) { appendStatus(msg); },
      onUploadError: function(msg, params) { appendStatus(msg); }
    });
  }
  catch (error) {
    appendStatus("WebScanning initialization error: " + error.description);
  }
});

function appendStatus(msg) {
  $('#status').append('<p>' + msg + '</p>');
}

```

This will display error messages to a div with id=status.

Handler: `onScanError(msg, params)`

The WingScan service can be initialized with a scan error handler (see the Code Example at the end of this section), and that handler will potentially be called back by WingScan with one of various scanning-related errors.

Note: It is *essential* to a well-functioning web scanning application that you handle at least the **activeX**, **noPlugin** and **oldPlugin** errors.

All WebCapture errors are string members of: `Atalasoftware.Controls.Capture.Errors`

Below are the scanning-related errors currently defined, with an explanation of their cause and recommendation for proper handling.

Errors.badBrowser

`Atalasoftware.Controls.Capture.Errors.badBrowser`

Fired in: Any unsupported browser

During: `Atalasoftware.Controls.Capture.WebScanning.initialize`

Cause: WingScan detected that it is running in a browser it does support. Common reasons WingScan 10.0 might fire this error:

- The browser is not Internet Explorer, Firefox, or Chrome
- The browser is one of the above, but not a supported version e.g. Internet Explorer 6.0
- The operating system is not Windows.

- The browser is not a 32-bit edition. 64-bit browsers are not supported because TWAIN support for 64-bit applications is currently almost non-existent.

How to Handle: We strongly recommend that your application display the msg parameter to the handler, or your own equivalent message. If you also display the value of the params parameter, which will be a string, it would help a technical support specialist identify the browser causing the problem.

Errors.activeX

Atalasoftware.Controls.Capture.Errors.activeX

Fired in: Internet Explorer only

During: Atalasoftware.Controls.Capture.WebScanning.initialize

Cause: WebCapture's initial attempt to start the EZTwainX ActiveX control on the page failed, or the installed ActiveX control is out-of-date.

Background: In normal circumstances, at the same time this error is fired, Internet Explorer will display the [Information Bar](#) or otherwise prompt the user to download & install the ActiveX. That prompt, the subsequent download and installation all happen *concurrently* with javascript execution.

Because the control's .cab file is several MB it can take several seconds to download. Because it is digitally signed by Atalasoftware, it can take the client computer several more seconds to authenticate the file after download. Installation typically takes another few seconds, and then the control is loaded and started, possibly several more seconds. During this time, Internet Explorer displays *no indication* to the user, not even a busy cursor. Neither is any progress information available to javascript code.

How to Handle: We strongly recommend that your application display a message or prompt that encourages the user to approve the download and install dialog(s), and be patient during the download and installation process.

If the user and IE successfully complete the ActiveX install process, WingScan will automatically start the new/updated control and update the UI elements on your page that pertain to scanning. The page does not need to be refreshed and IE does not need to be restarted.

Errors.noTwain

Atalasoftware.Controls.Capture.Errors.noTwain

Fired In: All browsers.

During: Atalasoftware.Controls.Capture.WebScanning.initialize

Cause: Support for the TWAIN protocol itself not found on the client computer.

Background: This error is extremely unlikely to happen on a typical end-user PC running Windows XP, Vista or 7, because retail editions of Windows all include a copy of the TWAIN manager. However, a user on Windows Server 2008 and perhaps some other Server editions can be missing TWAIN which will cause this error. Ref:

[Using scanners in Windows Server 2008 R2 with TWAIN drivers might require the installation of Desktop Experience Pack.](#)

How to Handle: You could just display the error string (the value of the msg parameter of the onScanError handler) or display your own message that TWAIN was not found on the computer.

Errors.noPlugin

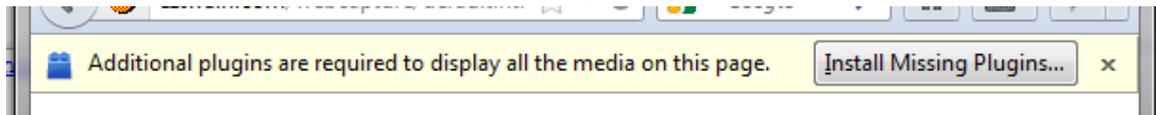
Atalasoftware.Controls.Capture.Errors.noPlugin

Fired In:All browsers other than Internet Explorer

During: Atalasoftware.Controls.Capture.WebScanning.initialize

Cause: The required Web Scanning plugin is *either* not installed or is disabled.

Background: If the plugin is not installed, Firefox will display a notification like this:



On the other hand, if the plugin is installed but disabled, Firefox displays no prompt.

Chrome never displays a prompt.

How to Handle:Your application should encourage the user to follow the prompts if there are any, and provide a download & install link to use if there is no prompt. The `params.filename` contains the unqualified filename of the plugin installer. You are responsible for displaying it as a clickable link with the correct URL to the file. See the code example at the end of this section.

The plugin installation process in Firefox is somewhat tortuous:

1. In the yellow alert bar, click the **Install Missing Plugins...** button.
2. In the resulting **Plugin Finder Service** dialog, which says “No suitable plugins were found”, click the **Manual Install** button.
3. This brings up a warning dialog that says: **Install add-ons only from authors whom you trust.** It will list **Kofax Web Scanning (Kofax, Inc).** Click the **Install Now** button.
4. This unpacks and copies the plugin's files into place, and displays a little pop-up that says “**Kofax Web Scanning will be installed after you restart Firefox**” with a **Restart Now** button.
5. When you restart, the page is reloaded. The plugin is installed and should now run without the `noPlugin` error.

The plugin installation process in Chrome is relatively painless:

1. When the user clicks on the hyperlink to the `.crx` file, a confirmation strip appears at the bottom of the window.
2. When the user OKs that, the plugin package is downloaded and unpacked, and the user is asked if they want to **Add the Kofax Web Scanning Plugin.**
3. If they approve that, the plugin is installed and a small pop-up informs them of this fact.
4. At this point the user must *refresh the page* so WingScan can detect it.

Errors.oldPlugin

Atalasoftware.Controls.Capture.Errors.oldPlugin

Fired In: All browsers other than Internet Explorer

During: Atalasoftware.Controls.Capture.WebScanning.initialize

Cause: The Web Scanning plugin is installed and enabled but WingScan is designed to work with a newer version. For example, WingScan might require plugin version 1.55, but detect that the browser has plugin version 1.42 installed. That would cause this error to be fired during initialization.

How to Handle: Similar to handling **noPlugin** above, but there will never be any prompting by the browser so you *must* present the user with a button or hyperlink to the correct plugin deployment package on your server. The filename of the appropriate download is passed to your error handler as `params.filename`.

Errors.licensingError

`Atalasoftware.Controls.Capture.Errors.licensingError`

Fired In: All browsers

During: Asynchronously, after `Atalasoftware.Controls.Capture.WebScanning.initialize()`

Cause: When queried, the server did not return the expected JSON licensing information in a timely fashion.

How to Handle: If you see this during development, it suggests some server configuration problem, the `handlerUrl` passed to `WebScanning.initialize` isn't right, the server is actually offline or not accessible, or (maybe, even) the licensing isn't right for WingScan on the server.

Assuming you resolve any logical problems during development, if this error occurs after deployment it almost certainly represents a typical "server not responding" error, with all the usual causes.

Other Errors

Other errors are possible, and additional errors may be added in future updates to WingScan. We recommend that you defend against that possibility by displaying the text of the error (the `msg` parameter to the `onScanError` handler) to the user, and offering them as much flexibility as possible - for example, by linking to a troubleshooting & support page that you can revise based on experience.

Handler: onScanClientReady()

This is a handler which can be passed to `WebScanning.initialize` alongside the `onScanError` handler. See the Code Example at the end of this section.

Called In: All browsers.

During: `Atalasoftware.Controls.Capture.WebScanning.initialize` OR at some later time if the scanning control needed to be downloaded and installed.

Cause: The client-side scanning control or plugin has just been successfully initialized and is operational. Note that this does *not* mean that any scanners were detected, working or otherwise, so the **Scan** button is not necessarily enabled.

Background: Alongside the `onScanError` handler, you can provide an `onScanClientReady` handler that will be called when client scanning services have been successfully initialized. Remember that the `onScanClientReady` handler may be called an arbitrarily long time after the `WebCapture.initialize` call.

How to Handle: This handler is a good place to clear any initialization error messages or prompts as discussed above. See the code example at the end of this section.

Code Example – Scan Error Handling

See also: [Initializing the Control on the Client](#).

Javascript

```

function scanErrorHandler(msg, params)
{
    appendStatus(msg);
    switch (msg) {
        case Atalasoftware.Controls.Capture.Errors.badBrowser:
            promptHTML(
                msg + " <br />(" + params + ")");
            break;

        case Atalasoftware.Controls.Capture.Errors.activeX:
            promptText(
                "The ActiveX Scanning Control needs to be installed or updated.\n" +
                "When prompted, please allow the Kofax Scanning Control to install itself.");
            break;

        case Atalasoftware.Controls.Capture.Errors.noPlugin:
            promptHTML(
                "The Kofax Web Scanning plugin is not available. "+
                "Please follow any prompts to install it, or <a href='/" + params.filename + "'>Click
Here</a><br />"+
                "If you are not prompted to install, the plugin may "+
                "be installed but disabled - please enable it.");
            break;

        case Atalasoftware.Controls.Capture.Errors.oldPlugin:
            promptHTML(
                "The Kofax Web Scanning plugin is out of date.<br />"+
                "To download and install the latest version "+
                "<a href='/" + params.filename + "'>Click Here</a>");
            break;

        case Atalasoftware.Controls.Capture.Errors.noTwain:
            promptText(
                "TWAIN is not installed on this computer.\n"+
                "Contact your system administrator.");
            break;

        default:
            promptText(msg);
            break;
    }
}

function scanClientReady() {
    promptText(""); // Clear the prompt box
}

// Initialize Web Scanning and Web Viewing
Atalasoftware.Controls.Capture.WebScanning.initialize({
    // designate error handler:
    onScanError: scanErrorHandler,
    onScanClientReady: scanClientReady,
    // etc...
});

```

Setting Scanning Options

In the `Atalasoft.Controls.WebScanning.initialize` method, one of the parameters that can be passed as an argument is `scanningOptions`. This is an object whose various properties control the scanner and the way images are processed after scanning.

Note that not all settings are supported by all scanners - if you use a setting with a scanner that does not support it, the unsupported setting is simply ignored.

applyVRS

An option to enable or disable VRS post-processing in general.

If you specify `applyVRS: false`, no VRS processing is performed and all VRS processing options are ignored.

If you specify *only* `applyVRS: true`, then by default you effectively get the following settings: `pixelType: 2`, `resultPixelType: 0`, `deskew: true`, `autoRotate: true`, `discardBlankPages: false`

Default value: `true`.

Example

```
Atalasoft.Controls.Capture.WebScanning.initialize({
  handlerUrl: 'TestCaptureHandler.ashx',
  scanningOptions: { applyVRS: false }
});
```

autoRotate

Detects the orientation of the text in an image - right-side up, upside-down, sideways - and rotates the image so the text is upright.

If VRS is disabled, `autoRotate` is always disabled. If VRS is enabled, `autoRotate` is enabled by default but you can disable it with this option.

deskew

Deskew is scanning jargon for 'straighten' - to rotate the scanned image by a few degrees to correct for the paper being scanned slightly crooked. Different from **autoRotate**.

If VRS is disabled, `deskew` is always disabled. If VRS is enabled, `deskew` is enabled by default but you can disable it with this option.

Example

```
Atalasoft.Controls.Capture.WebScanning.initialize(
  scanningOptions: { applyVRS: true, deskew: false }
});
```

disableVRSIfInstalledOnWorkstation

Automatically disable VRS processing if VRS is detected on the client workstation. The idea is that if VRS is detected on the workstation, the user is probably using a VRS-equipped TWAIN driver, so there is no need to apply VRS processing twice to each image.

Default value: `false`.

discardBlankPages

When this option is true, blank images are detected and discarded during scanning.

Note: In duplex scanning, front and back sides of pages are discarded independently.

Chapter 2

Note: No ImageAcquired event is fired for such discarded images.

Default value: false.

Example

```
Atalasoft.Controls.Capture.WebScanning.initialize({
    handlerUrl: 'TestCaptureHandler.ashx',
    scanningOptions: { duplex: 1, discardBlankPages: true }
});
```

dpi

Controls the scanning resolution. It stands for *dots per inch*. It would be very unusual to find a scanner that doesn't support 100, 200 and 300 DPI. 150 DPI is almost as widely supported. Nearly all flatbed scanners can scan anything from 50 DPI to 1200 DPI.

Note: The units of this value are always dots per *inch*, even if the computer, user account or browser are configured for a metric locale.

Default value: 200.

Example

```
Atalasoft.Controls.Capture.WebScanning.initialize({
    handlerUrl: 'TestCaptureHandler.ashx',
    scanningOptions: { dpi: 300} // request 300 DPI scanning
});
```

duplex

Controls duplex/simplex scanning. The possible values are:

0 = Simplex (front side only)

1 = Duplex (both sides)

-1 = Don't care (leave up to scanner)

Default value: 0 (Simplex).

All scanners support simplex scanning. Many scanners with an ADF (Automatic Document Feeder) can scan duplex, but many cannot.

Example

```
Atalasoft.Controls.Capture.WebScanning.initialize({
    handlerUrl: 'TestCaptureHandler.ashx',
    scanningOptions: { duplex: -1 }
});
```

feeder

This option selects between the ADF (Automatic Document Feeder) and the flatbed/glass AKA the *platen*

Valid values are: 0 - Scan from platen, 1 - Scan from feeder, -1 - Don't care (up to scanner or user).

Default value: -1 (Don't care)

Example

```
Atalasoftware.Controls.Capture.WebScanning.initialize({
  handlerUrl: 'TestCaptureHandler.ashx',
  scanningOptions: { feeder: 0 }
});
```

orientation

This parameter tells the scanner the expected orientation of the paper being fed, in the sense of upright (short edge feed) or sideways/landscape (long edge feed).

| Value | Name | Meaning |
|-------|-----------|--|
| -1 | Any | left up to scanner. |
| 0 | Portrait | paper is scanned 'upright' (short edge feed) |
| 1 | Landscape | paper is scanned 'sideways' (long edge feed) |

paperSize

To set the paper size being fed in to the scanner.

Default value: 3 (8.5" x 11")

| Value | Meaning / Dimensions |
|-------|---|
| -1 | Indicates 'no preference' |
| 0 | TWAIN defines this as meaning 'maximum scan area' but many scanners will treat this as 'default' or 'last size selected by the user.' |
| 1 | 210mm x 297mm |
| 2 | 182mm x 257mm (Same as JIS B5) |
| 3 | 8.5" x 11.0" |
| 4 | 8.5" x 14.0" |
| 5 | 148mm x 210mm |
| 6 | 250mm x 353mm (ISO B4) |
| 7 | 125mm x 176mm (ISO B6) |
| 8 | <i>unused</i> |
| 9 | 11.0" x 17.0" |
| 10 | 10.5" x 7.25" |
| 11 | 297mm x 420mm (ISO A3) |
| 12 | 353mm x 500mm (ISO B3) |
| 13 | 105mm x 148mm (ISO A6) |
| 14 | 229mm x 324mm (ISO C4) |
| 15 | 162mm x 229mm (ISO C5) |
| 16 | 114mm x 162mm (ISO C6) |

```

Example
Atalasoft.Controls.Capture.WebScanning.initialize({
    handlerUrl: 'TestCaptureHandler.ashx',
    scanningOptions: { paperSize: 3 }
});
    
```

pixelType

Sets the pixel type for scanning. We recommend using **resultPixelFormat** instead.

| Value | Scan Data Format |
|-------|-----------------------------------|
| 0 | Black and white (1 bit per pixel) |
| 1 | Grayscale (8 bits per pixel) |
| 2 | RGB (24 bits per pixel) |
| 3 | Indexed color (8 bits per pixel) |
| -1 | Don't care |

Default value: 0 (Black & white)

```

Example
Atalasoft.Controls.Capture.WebScanning.initialize({
    handlerUrl: 'TestCaptureHandler.ashx',
    scanningOptions: { pixelType: 0 }
});
    
```

Every scanner capable of scanning paper documents can scan in Black & White (B&W) mode. Almost all scanners can scan grayscale and color. Many scanners, but certainly not all, can scan indexed color.

resultPixelFormat

This specifies the pixel format for images delivered to your application after scanning and post-processing.

This is distinct from the **pixelType** parameter, which controls the pixel format requested from the scanner.

If **resultPixelFormat** is not specified, it defaults to -1.

The pixel format used for scanning is:

1. **pixelType** if specified.
2. otherwise the **pixelType** implied by **resultPixelFormat** if specified, (see tables below).
3. otherwise if **applyVRS** is **true** then Color
4. otherwise: B&W.

| applyVRS:true | | | |
|---------------|------------------|--|--------------|
| Value | Name | Delivered Image | Default Scan |
| -2 | PixelFormat.Auto | 24-bit color or 1-bit B&W, chosen by VRS | Color |
| -1 (default) | PixelFormat.Any | 1-bit images (binarized by VRS) | Color |

| applyVRS:true | | | |
|---------------|-----------------------|------------------------|--------------|
| Value | Name | Delivered Image | Default Scan |
| 0 | PixelFormat.BW | 1-bit B&W images | Color |
| 1 | PixelFormat.Grayscale | 8-bit grayscale images | Grayscale |
| 2 | PixelFormat.Color | 24-bit color images | Color |

| applyVRS:false | | | |
|----------------|-----------------------|------------------------|--------------|
| Value | Name | Delivered Image | Default Scan |
| -2 | PixelFormat.Auto | as scanned | BW |
| -1 (default) | PixelFormat.Any | as-scanned | BW |
| 0 | PixelFormat.BW | 1-bit B&W images | BW |
| 1 | PixelFormat.Grayscale | 8-bit grayscale images | Grayscale |
| 2 | PixelFormat.Color | 24-bit color images | Color |

Note that when VRS is disabled, **resultPixelFormat** can be effectively used in place of **pixelType** to control the scanner.

Example

```
var PixelType = Atalasoft.Controls.Capture.PixelType;
Atalasoft.Controls.Capture.WebScanning.initialize({
    // Deliver color images. Implies scanning color.
    scanningOptions: { resultPixelFormat: PixelType.Color }
});
```

showScannerUI

To show (true), or not show (false) the scanner's user interface during scanning.

Default value: *false*.

Example

```
Atalasoft.Controls.Capture.WebScanning.initialize({
    handlerUrl: 'TestCaptureHandler.ashx',
    scanningOptions: { showScannerUI: true }
});
```

suppressBackgroundColor

Only has effect in Auto Color mode i.e. when **applyVRS** is true and **resultPixelFormat** is -2.

In that mode, if **suppressBackgroundColor** is true, solid-color background in color scans is treated as white. If there is no other color content on a scanned image, the image will be automatically converted to B&W.

This is useful when your scan batch may include invoices and other documents printed on colored paper, which you want converted to B&W, but you also expect some pages with color content which you want to be preserved as color.

Example

```
Atalasoft.Controls.Capture.WebScanning.initialize({
  handlerUrl: 'TestCaptureHandler.ashx',
  scanningOptions: {
    resultPixelFormat: -2,           // detect color & B&W pages automatically
    suppressBackgroundColor: true   // treat solid color background as white
  }
});
```

tiff.jpegCompression

Controls use of JPEG compression when writing color and grayscale images in TIFF format.

Important: Uses the revised TIFF 6 form, not Wang/Microsoft variant - check your downstream processes for compatibility.

When true, JPEG compression is used when writing color or grayscale images to TIFF.

When false, WingScan is free to choose some other compression for color and grayscale images in TIFF. For WingScan 10.0 the default choice is "no compression".

Default value: *false*.

Example

```
Atalasoft.Controls.Capture.WebScanning.initialize({
  handlerUrl: 'TestCaptureHandler.ashx',
  scanningOptions: { tiff: { jpegCompression: true } }
  // Note the nested object-within-object construction
}
```

Connecting to the Web Document Viewer

To display the documents scanned with the WingScan WebScanning Control in the same page, or possibly another page in a web browser simply do the following:

1. Add the [Web Document Viewer](#) resources to the application.
2. Add the following div tag to the same page:

Code Snippet

```
<div>
  <div class="atala-document-toolbar" style="width: 670px;"></div>
  <div class="atala-document-container" style="width: 670px; height: 500px;"></div>
</div>
```

3. In the html/aspx/jsp page add the following script. The values of `serverurl` and `handlerUrl` should be changed to the locations of your WebDocViewer and Capture handlers, respectively.

Code Snippet

```

<script type="text/javascript">
  // Show status and error messages
  function appendStatus(msg)
  {
    $('#status').append('<p>'+msg+'</p>');
  }

  // Initialize Web Scanning and Web Viewing
  $(function() {
    try {
      var viewer = new Atalasoftware.Controls.WebDocumentViewer({
        parent: $('#atala-document-container'),
        toolbarparent: $('#atala-document-toolbar'),
        serverurl: 'WebDocViewer.ashx'
      });

      Atalasoftware.Controls.Capture.WebScanning.initialize({
        handlerUrl: 'TestCaptureHandler.ashx',

        onScanError: function(msg, params) { appendStatus(msg); },
        onScanStarted: function(eventName, eventObj) { appendStatus('Scan
Started'); },
        onScanCompleted: function(eventName, eventObj) { appendStatus('Scan
Completed: ' + eventObj.success); },

        onUploadError: function(msg, params) { appendStatus(msg); },
        onUploadStarted: function(eventName, eventObj) { appendStatus('Upload
Started'); },
        onUploadCompleted: function(eventName, eventObj) {
          appendStatus('Upload Completed: ' + eventObj.success);
          if (eventObj.success) {
            viewer.OpenUrl('atala-capture-upload/' +
eventObj.documentFilename);
          }
        },
        scanningOptions: { pixelType: 0 }
      });

      Atalasoftware.Controls.Capture.CaptureService.initialize({
        handlerUrl: 'TestCaptureHandler.ashx',
        onError: function(msg, params) { appendStatus(msg + ': ' +
params.statusText); }
      });
    }
    catch (error) {
      appendStatus('Thrown error: ' + error.description);
    }
  });
</script>

```

Using VirtualReScan (VRS)

The WingScan Web Scanning Control includes the award-winning VirtualReScan Technology (VRS), a "de Facto Standard" for image processing.

By default, VRS processing is applied to each scanned image: All images are auto-rotated and deskewed, and non-B&W images are converted to B&W ('binarized').

Please note: The specific image processing steps performed by VRS may change in future versions of WingScan.

To turn on or off VRS processing in the client an optional argument must be passed into the `Atalasoftware.Controls.Capture.WebScanning.initialize` method on the page in which the control has been added. See [Client API Reference](#).

Example - Disabling VRS

JavaScript

```
Atalasoftware.Controls.Capture.WebScanning.initialize({
  handlerUrl: 'TestCaptureHandler.ashx',

  onScanError: function(msg, params) { appendStatus(msg); },
  onScanStarted: function(eventName, eventObj) { appendStatus('Scan Started'); },
  onScanCompleted: function(eventName, eventObj) { appendStatus('Scan Completed: ' +
eventObj.success); },

  onUploadError: function(msg, params) { appendStatus(msg); },
  onUploadStarted: function(eventName, eventObj) { appendStatus('Upload Started'); },
  onUploadCompleted: function(eventName, eventObj) {
    appendStatus('Upload Completed: ' + eventObj.success);
    if (eventObj.success) {
      viewer.OpenUrl('atala-capture-upload/' + eventObj.documentFilename);
    }
  },
  scanningOptions: { pixelType: 1, applyVRS: false }
});

Atalasoftware.Controls.Capture.CaptureService.initialize({
  handlerUrl: 'TestCaptureHandler.ashx',
  onError: function(msg, params) { appendStatus(msg + ': ' + params.statusText); }
});
```

In the `scanningOptions` argument, `applyVRS` is set to *false* to turn VRS off in the Web Scanning Control.

Testing Your Application

Obviously you should test scanning, ideally with several scanners. Yes, we try to hide all the scanning issues and make it "just work". Nonetheless, it can be beneficial to learn about the problems your end-users will have setting up and using scanners, to get a sense of the little idiosyncrasies every scanner has, and to understand the physical details of the task you are asking users to carry out.

Test in Internet Explorer, Firefox and Chrome

The client-side ActiveX control and the Firefox/Chrome plugin are separate projects with significant code differences.

Test for Error Conditions.

Scanning and uploading documents is fast and simple when everything works. Your users' efficiency and satisfaction will primarily depend on how you handle errors and failures.

Verify that your application behaves in a reasonable way and guides users successfully when:

1. The browser is unsupported: Safari, or Opera, or a non-Windows OS.
2. All ActiveX is disabled in Internet Explorer (for example, by putting your test site in the Restricted Sites security zone.)
3. The EZTwainX control is completely removed/uninstalled from Internet Explorer ([Uninstalling ActiveX Control and Plugin](#))
4. The EZTwainX control is specifically disabled in IE.
From **Tools** menu - **Manage Add-ons** - Show: **All add-ons** - select **EZTwainX - Disable** button.
5. In Firefox or Chrome, when the plugin is removed and uninstalled ([Uninstalling ActiveX Control and Plugin](#)).
6. In Firefox or Chrome, when the plugin is *disabled* (as described in [Uninstalling ActiveX Control and Plugin](#)).
7. The client PC has no devices in the TWAIN device list.
8. The selected scanner is turned off or disconnected.
9. A scan is canceled mid-scan.
10. WingScan throws an error, especially those documented in [Handling Errors](#).

Troubleshooting Web Scanning Problems

My scanner appears in the list as WIA-something - what does this mean?

Many scanners support Microsoft's proprietary scanner protocol, called [WIA](#). Microsoft Windows performs some magic to make WIA devices also appear as TWAIN devices. However, these pseudo-TWAIN devices are not native TWAIN drivers, and sometimes have important limitations. If you have any problems using a WIA driver through TWAIN, see if the scanner vendor offers a native TWAIN driver.

My scanner appears twice in the scanner list, once with a WIA-prefix and once without - what does this mean?

This means your scanner supports the Microsoft [WIA](#) scanner protocol as well as having a native TWAIN driver. Basically you are seeing two different drivers that can both talk to your scanner. In general we recommend using the native (non-WIA) driver, but you are welcome to try them both and see which one works better for you.

When installing the Firefox plugin, it acts like a download file: The user is prompted to open npWebCapture.xpi

This problem is specific: Your webserver is not serving the npWebCapture.xpi file with MIME type 'application/x-xpinstall'. Reconfigure your web server to serve .xpi files with that MIME type.

[Configure MIME Types \(IIS 6.0\) - Microsoft TechNet](#)

[Configuring MIME Types in IIS 7 - Microsoft TechNet](#)

Scanner does not appear in device list.

Things to check:

- Is the scanner connected and powered on.
- Does the scanner support TWAIN? The popular Fujitsu ScanSnap models do not.
- Is a TWAIN driver for the scanner installed? Most do not auto-install.
- Test the driver+scanner combination outside the browser with IrfanView, see "Scanning fails" below.
- Sometimes, in Internet Explorer: Try moving your website into the trusted zone.

Scanner does not appear in device-list on web page, but is listed in and works with some desktop application

Most likely cause: IE is 'sandboxing' the driver, preventing it from communicating with the scanner.

Try moving your website into the trusted zone.

Scanning fails with "unable to open" or "connection failure"

Scanning fails before scanning any pages

- Is the scanner connected and powered on?
- If there has been a recent crash or error related to scanning?
Try cycling the power on the scanner and then re-try the scan, up to two times.
- Verify that the scanner is working outside the browser, through TWAIN. Note the TWAIN name of the device.

To verify that a scanner has a working TWAIN driver, we sometimes use [IrfanView](#) - this is a free scanning application with TWAIN support.

If IrfanView can scan from your scanner, then you have a working scanner with a working TWAIN driver.

In this case web scanning failures are most likely Internet Explorer 'sandboxing' the scanner driver: Try moving your website into the trusted zone.

If IrfanView *cannot* find and scan from your scanner, then you don't have the basic prerequisite of a working TWAIN scanner.

The ultimate fall-back for this kind of problem is to get support from the scanner vendor.

Uploads fail with '598' status

If uploads fail with '598' status codes, this indicates the client-side code timed-out waiting for the upload to complete. You can increase the [Params.serverTimeout: Integer](#) value, try to speed up your connection, reduce your upload sizes (see below), or (if it's actually the problem) speed up your server.

Uploads fail with JSON parse error or server response status 404

If uploading fails with a JSON parse error or server response 404, once you check that the URL being used for upload is valid and correct, consider whether the upload might be exceeding the server's upload size limit. See [Upload Sizes and Limits](#). Some suggestions:

- Increase the server's upload limit.
- Scan to grayscale instead of color, or better yet to B&W. If you use VRS, we recommend specifying resultPixelFormat instead of pixelType. This allows VRS to scan in color and use sophisticated algorithms to convert to grayscale or B&W. Note that this may cause slower scanning with some scanners.
- If you have licensed VRS, try setting resultPixelFormat: -2 to tell VRS to automatically classify color and non-color images, and convert non-color images to B&W. If you are scanning pages on colored paper, take a look also at suppressBackgroundColor.
- If you are scanning with a DPI higher than 200, experiment with scanning at 200 DPI to see if the results are acceptable.

Documents do not display in viewer after scan & upload

First check the upload: Are the documents being uploaded?

1. Attach handlers for onUploadStarted, onUploadCompleted and onUploadError (see [Client API Reference](#)) - is onUploadCompleted being fired, and not onUploadError?
2. Does each new upload appear in the upload folder on the server? (See notes about the upload folder in [Getting Started with Web Capture](#)).

Second, check the viewer:

1. Is the code to invoke the viewer being called, and is the correct URL being given to the viewer? An alert box is an easy way to check this.
2. Can you enter the URL or its fully-qualified equivalent into a browser manually and get the expected file?

Scan quality is poor

If images or graphics looks bad, could this be because the scans are being converted to black & white? See question below.

Are you setting the resolution? See [Setting Scanning Options](#). A very low resolution - anything below 100 DPI or so - will produce blurry or ... well, 'low-res' images.

As a very rough guide, for black & white scans of text:

- 100 DPI legible but visibly rough & pixelated, like a poor quality fax or a 1970's video game.
- 150 DPI modest fax quality, still some visible defects and pixelation but highly legible in typical on-screen viewing.
- 200 DPI high quality fax. A full page viewed on screen looks good, letters look slightly fuzzy or 'haloed'.
- 300 DPI good quality, minor defects usually visible only under magnification.

All scans are converted to B&W, even my bunny pictures

This is the default behavior for WebCapture: It applies several VRS clean-up operations to each scan, including *binarization*, which converts the image to 1-bit per pixel black & white.

To avoid this, specify the **resultPixelFormat** as grayscale (1) or color (2). If you are using VRS, you can also specify **resultPixelFormat: -2** which tells VRS to automatically classify each image as color or non-color, and to convert only non-color images to B&W.

I ask for duplex scanning, but only front sides are scanned.

It sounds a little silly, but the first thing to check is that 'duplex scanning' is a listed feature of the scanner.

Assuming the scanner claims to support duplex (both sides) scanning, the most common reason for it to fail is using the scanner through a WIA driver (choosing WIA-something in the scanner list). The WIA drivers have historically had problems with duplex scanning.

If a WIA-driver is being used, the solution is to find, install and use a native TWAIN driver for the scanner.

Uninstalling ActiveX Control and Plugin

The Firefox/Chrome Web Scanning plugin

If you just want to diagnose some problem you think the plugin is causing, or force a re-install of the plugin, it might be simpler to *disable* the plugin instead of completely removing it. Disabling a plugin in Firefox or Chrome has most of the effect of removing it (except for the disk space) but is a simpler process for end-users.

Disabling The Web Scanning Plugin

1. Choose **Add-ons** in the Firefox menu, or choose **Settings** from the 'wrench' menu in Chrome.
2. In Firefox, select the **Plugins** tab to see all installed plugins.
3. In Chrome, select the *Extensions* panel on the left side.
4. Find the Atalasoftware **Web Scanning Plugin**. In Firefox click the **Disable** button. In Chrome, uncheck the **Enabled** checkbox.
5. Go back to your scanning page and refresh. Voilà.

Uninstalling The Web Scanning Plugin From Chrome

1. Click the wrench icon top-right and choose Settings from the drop-down menu.
2. Select *Extensions* in the left column
3. Find the Atalasoftware **Web Scanning Plugin**, move the mouse over it and when the garbage can appears to the right, click that.
4. Confirm Removal when prompted.

Uninstalling The Web Scanning Plugin From Firefox

Firefox installs a plugin by copying some files from the .xpi package into a folder in the user's Firefox profile. Firefox does not record plugins in the Windows registry, nor does it maintain any kind of private plugin database. Uninstalling the plugin is simply a matter of deleting those files and restarting Firefox.

1. Find the Firefox **Help** menu and choose **Troubleshooting Information**.
2. Click the **Open Containing Folder** button.
This opens the users *profile folder*, which stores all the user's Firefox state and configuration - including extensions & plugins.
3. Go into the **extensions** subfolder.
4. Look for this folder: **x-webcapture@atalasoftware.com** and delete it.
If you get 'in use' errors, try shutting down all running instances of Firefox.
5. Restart Firefox.
6. To confirm, choose **Add-ons** in the Firefox menu, select the **Plugins** tab, and check that the Atalasoftware **Web Scanning Plugin** does *not* appear.

The Internet Explorer ActiveX control

Uninstalling the EZTwainX ActiveX Control from Internet Explorer

See also: [How to Remove an ActiveX Control in Windows](#) (external link)

There are two ways the EZTwainX control could be installed on a client computer:

1. By a traditional setup process, which would copy the control somewhere under **Program Files** and register it in the registry.
(This case applies because EZTwainX has been distributed for some years as a stand-alone product.)
2. By automatic download from a .cab file referenced by a web page.
(The method used by WingScan.)

It is possible to have one copy of EZTwainX on a system installed by method 1, and one *or more* copies/versions installed by method 2. Internet Explorer seems to be fairly smart about finding and using the appropriate version.

Internet Explorer installs an ActiveX control (if the control is not already installed on the computer) by copying its files into **Windows\Downloaded Program Files**. Sometimes it will place the files at the top level of that folder, sometimes they will be inside a folder with a name like **CONFLICT.1** or **CONFLICT.2**.

To Remove An Automatically Downloaded EZTwainX:

1. Open the **Windows\Downloaded Program Files** folder.
From IE you can do: **Tools | Internet Options | General tab | [History] Settings button | View Objects button**
2. Delete any **eztwainx.ocx** and **eztwainx.inf** files.
Check inside any **CONFLICT** subfolders too.
3. If you get sharing errors or in-use errors, try shutting down all running instances of Internet Explorer (iexplorer.exe).
4. Restart Internet Explorer.
5. To confirm, in IE under **Tools** choose **Manage Add-ons**, in the left panel in the **Show list** choose **All add-ons**. check that **EZTwainX** does *not* appear.

Client API Reference

Atalasoftware.Controls.Capture.WebScanning

This object is responsible for communicating with the client-side scanner, controlling scanning, and uploading scanned documents to the web server.

Atalasoftware.Controls.Capture.WebScanning.initialize(params)

This method must be called to initialize the WebScanning component. The params object must contain a **handlerUrl** property, the other properties are optional.

As a side-effect, **initialize** attempts to initialize TWAIN scanning on the client, and to collect a list of available TWAIN scanners. If it is successful, it will populate a scanner-list control and enable a scan button, provided that they exist with the appropriate classes. See [Connecting to UI Controls](#).

The scanner initialization process is *asynchronous* and may not have finished when the **initialize** function returns. In fact it may never complete, for example if the browser has ActiveX controls disabled, or if the user declines to install and activate the ActiveX or plugin.

params.handlerUrl: string

The URL of the web request handler. This is normally a relative URL, for example: 'TestCaptureHandler.ashx'

Params.serverTimeout: Integer

This is the number of seconds to wait for the server response after starting an upload. After this number of seconds, the upload is considered to have failed, it is canceled, and an error is signaled by calling `onUploadError`.

Default value: 20

Params.onScanClientReady: function()

This handler is called when scanning initialization is complete. Not successful, just complete: Scanning initialization has either succeeded fully, or failed. Normally in case of failure the `onScanError` handler will have been called.

See [Handling Errors](#)

params.onScanError: function(msg, params)

This handler is called when an error occurs during scanning *or scan initialization*.

See [Handling Errors](#)

params.onScanStarted: function(eventName, eventObj)

This handler is called when a scan is started. See [Handling Events](#)

params.onScanCompleted: function(eventName, eventObj)

This handler is called when a scan is completed. See [Handling Events](#)

params.onUploadStarted: function(eventName, eventObj)

This handler is called when a document upload is starting. See [Handling Events](#)

params.onUploadCompleted: function(eventName, eventObj)

This handler is called a document upload has completed. See [Handling Events](#)

params.onUploadError: function(msg, params)

This handler is called when an error occurs during uploading. See [Handling Events](#)

params.scanningOptions: object

This object contains any scanner settings to be used for scanning, as described in [Setting Scanning Options](#).

Atalasoftware.Controls.Capture.WebScanning.scan(options)

Initiates a scan with the specified scanning options (see [Setting Scanning Options](#)).

If you pass in nothing, null or undefined, it uses the scanning options stored in [Atalasoftware.Controls.Capture.WebScanning.scanningOptions](#).

This method is called (with no parameters) when the user clicks the designated scan button. See [Connecting to UI Controls](#)

Atalasoftware.Controls.Capture.WebScanning.scanningOptions

This property holds the current scanning options as described in [Setting Scanning Options](#). These options are used when the user clicks the scan button, or if `Atalasoftware.Controls.Capture.WebScanning.scan()` is called.

Initially this holds the scanningOptions object passed to [Atalasoftware.Controls.Capture.WebScanning.initialize\(params\)](#), but your code can dynamically edit this object to change the settings for a subsequent scan.

Atalasoftware.Controls.Capture.CaptureService

This object is responsible for returning information from KCIC-WS, such as the content-types and content-type document descriptions, and for importing uploaded documents into Kofax Capture.

.initialize(params)

This method must be called to initialize the CcaptureService component. The params object must contain a **handlerUrl**, the other items are optional.

As a side-effect, **initialize** starts a process that attempts to communicate with the KCIC-WS service and obtain the content-type list and content-type document description list. If this process succeeds, it will populate the appropriate controls on the web page, if they exist with the correct classes. See [Connecting to UI Controls](#).

params.handlerUrl: string

The URL, normally relative, of the KCIC-WS extended web request handler on the server.

For example: 'TestCaptureHandler.ashx'

params.onError: function(msg, params)

This handler is called if KCIC-WS returns an error.

Web Document Viewer

The following sections describe setting up and using the WingScan Web Document Viewer.

| | |
|---|-----------|
| Web Document Viewer Overview | 62 |
| Web Document Viewer Guide | 64 |

Web Document Viewer Overview

The WebDocumentViewer is JavaScript based image viewing control that can be created on the client side without the need for a traditional WebServerControl back end. It communicates directly with a WebDocumentRequestHandler on the server side, so there are no page lifecycle problems to deal with.

Getting Started

A WebDocumentViewer only requires a few snippets of HTML and JavaScript on your page, and a separate bare-bones handler.

The WebDocumentViewer doesn't have a Toolbox item to drag onto a form, so you can create the control on any page that you need to use it, without forms. See our [Web Document Viewer Guide](#) for a step-by-step tutorial of setting up a WebDocumentViewer in a new project and deploying it to an IIS server. A complete example of the WebDocumentViewer is also included in the DotImageWebForms demo projects that are installed with DotImage.

JavaScript API

The WebDocumentViewer object that gets created on your page currently only supports a single public method:

OpenUrl (path)

Where 'path' is a relative path to an image on your server. OpenUrl allows you to open a new document in a viewer after it has been constructed. The public API may be extended in future releases as features are added.

Constructor

The WebDocumentViewer constructor can accept different configuration parameters to change the initial behavior of the viewer. The current supported options are:

serverurl : A relative path to your viewer's web handler. Required.

documenturl : A relative path to a document on your server that will be initially displayed. Required.

parent : The jQuery div element in your page that will contain the viewer. Required.

toolbarparent : The jQuery div element in your page that will contain the viewer's toolbar.

fitting : The default method for fitting pages. Acceptable values are Fitting.Width and Fitting.None.

pagespacing : The width (in pixels) between pages.

showpageborder : Whether or not page borders should be displayed. Acceptable values are true and false.

showpagenumber : Whether a page number should be shown on each page. Acceptable values are true and false.

allowannotations : Whether annotations should be enabled. Acceptable values are true and false.

savepath : A relative path where annotation data should be saved on the server. Required for annotation saving.

See the [Web Document Viewer Guide](#) for basic constructor usage with required configuration options.

jQuery UI Styling

The control's toolbar is styled with jQuery UI by default. You can download new themes from <http://jqueryui.com/themeroller/> and include them in your web page after the required javascript and css includes for the viewer. In order to prevent viewer styling from overriding styling on the rest of your page, some jQuery UI styling is further customized by jQuery UI CSS classes prefixed with 'atala-'. These classes can be overridden to change default styling of the control and toolbar.

Troubleshooting

- If images don't show up, and you receive an alert that there was an image error, the license file might not have been found by the handler. Try placing the license file in the bin folder of the Web Site.
- If problems still persist, please contact our Support Team.

Web Document Viewer Guide

Getting Started with the WingScan Web Document Viewer

This guide will take you through setting up a basic web page containing an embedded Web Document Viewer and displaying an initial document in it.

Setting up your project

In Visual Studio, create a new Web Site using the ASP.NET Empty Web Site template.

Immediately, you should open your project's Property Pages to set up the necessary references and project options. Two changes must be made here.

- In the References menu, add a new reference to WingScan WebControls (.NET 2.0). This component was made available when you installed WingScan. Dependencies will automatically be included in your project.
- In the Build menu, change your Target Framework to .NET Framework 3.5. This will force your project to reopen, as well as populate your web.config.

Add project resources

Your project will need a copy of the Web Document Viewer resources, which includes client-side javascript and styles. These resources were installed with WingScan. The default install location for these is in C:\Program Files (x86)\Atalasoftware\WingScan 10.0\bin\2.0\x86\WebResources\WebDocViewer.

Copy the WebDocViewer directory into the root of your project.

We'll also create a default location to store images that will be displayed by the viewer. Create a new directory called Images in the root of your project, and add an image or document of your choice to this directory that you want displayed by default. We'll assume that the image you've added is called Example.tif.

Add a handler for the viewer

The document viewer will communicate with a separate handler on your website.

Add a new Generic Handler to your project. We'll assume that this file is called WebDocViewer.ashx.

Visual Studio adds a default implementation of a web handler. Replace the entire contents of the file with the following piece of code:

```
C#
<%@ WebHandler Language="C#" Class="WebDocViewerHandler" %>
using System;
using System.Web;
using Atalasoftware.Imaging.WebControls;

public class WebDocViewerHandler : WebDocumentRequestHandler
{
}
```

There is no need for further modification to your handler.

Add your web page

In a real deployment, you will want to insert the web document viewer into your own web page, but for this example we will work with a new page.

Add a new Web Form to your project. We'll assume that this file is called Default.aspx. Visual Studio will automatically create a codebehind for this file called Default.aspx.cs, but you will not need to change it.

A Web Document Viewer needs three chunks of code to load resources, create a viewing area, and initialize that area.

To load the necessary resources for creating web document viewer objects, add the following lines of HTML in your document's head.

Some of the resources included contain explicit version numbers which may change in future releases. Make sure the version numbers referenced in your `src` attributes match the files in the WebResources directories.

HTML

```
<script src="WebDocViewer/jquery-1.7.1.min.js" type="text/javascript"></script>
<script src="WebDocViewer/jquery.easing.1.3.js" type="text/javascript"></script>
<script src="WebDocViewer/jquery-ui-1.8.14.custom.min.js" type="text/javascript"></script>
<script src="WebDocViewer/atalaWebDocumentViewer.js" type="text/javascript"></script>
<link href="WebDocViewer/atalaWebDocumentViewer.css" rel="stylesheet" type="text/css" />
```

Next, add the following HTML into your document's body to create the document viewing area. The div tags can be customized. In this example, the height and width have been constrained.

HTML

```
<div id="_toolbar1" class="atala-document-toolbar" style="width: 670px;"></div>
<div id="_container1" class="atala-document-container" style="width: 670px; height:
500px;"></div>
```

Finally, add the following chunk of JavaScript for initializing your viewer. The constructor accepts various configuration options that affect the viewer's behavior or initial state. A minimum configuration is provided to tell the viewer where it should create the viewer (pointing to the div tags we added), where its web handler is located, and what image should be displayed initially.

JavaScript

```
<script type="text/javascript" language="javascript">
  var _docUrl = 'Images/Example.tif';
  var _serverUrl = 'WebDocViewer.ashx';
  var _viewer = new Atalasoftware.Controls.WebDocumentViewer({
    'parent': $('#_container1'),           // parent container to put the viewer in
    'toolbarparent': $('#_toolbar1'),     // parent container to put the viewer toolbar in
    'serverurl': _serverUrl,             // server handler url to send image requests to
    'documenturl': _docUrl               // document url relative to the server handler
  url
  });
</script>
```

Deploying to IIS

At this point your web site is ready to build and use. If you are deploying to IIS, you will need to make sure the following items are taken care of.

- Copy your project to a path within your IIS document root.
- Get a server license for WingScan, and put it in a location where it can be found. The best place to put your license file is in your project's Bin directory.

Chapter 2

- In IIS Manager, convert your project directory to an Application and assign it to an Application Pool.
- Check the settings of the Application Pool you used, and make sure Enable 32-Bit Applications is set to True. This will ensure your application is consistent with the assemblies you used.

At this point, you should be able to navigate to your page in a browser, and see a loaded document.