IMPORTANT: Disable if not using

First point is this: if you are not planning on using the File Upload feature in WebDocumentViewer (WDV) you should follow best practices and completely disable the feature by handling the FileUpload event server side and explicitly setting e.Cancel = true;

Please see: HOWTO: Completely Disable Upload Feature in WebDocumentViewer

Upload files with Web Document Viewer

Web Document Viewer (WDV) supports uploading files on a server. It can be done using WDV JavaScript API or WDV UI.

Enabling file upload

File upload is disabled by default. By setting an 'upload' configuration section to any value, including the 'empty object,' you can enable it.

Minimal config to enable upload with default parameters and uploading to FileUploads directory:

NOTE: The uploadpath value from client side should be verified in the FileUpload event handler server side. Just accepting user input for directories or file names is not a safe or secure practice

```
ar viewer = new Atalasoft.Controls.WebDocumentViewer({ //... other viewer settings upload: {
uploadpath: '/FileUploads'} });
```

Config to enable upload with specified upload path

```
ar viewer = new Atalasoft.Controls.WebDocumentViewer({ //... other viewer settings upload: {
uploadpath: 'folder\\subfolder' }, //... });
```

Using this section, you not only enable/disable upload feature but also configure it, to perform files filtering for upload on a client-side. Files can be filtered by size, extension or

MIME-type. The viewer performs minimal filetypes checks, and for best practice/secure file filtering the server-side API should be used. (Note the preceding link was for use in .NET Framework, for .NET 6/ 7/ 8, use WebDocumentViewerCallbacks API)

There are also several flags that allow a developer to enable or disable multiple files upload using WDV UI which includes configuration of maximum active parallel file upload requests; to enable or disable for an end-user possibility to add files for upload using drag-and-drop mechanism.

The last thing is 'enabled' flag. It allows to the developer disable upload feature even with if upload section is presented in WDV config.

NOTE: client side config and enabling/disabling is a convenience - any secure implementation of file upload must inspect/verify all user input on the server side API.

Config section with all flags mentioned

```
ar viewer = new Atalasoft.Controls.WebDocumentViewer({ //... other viewer settings upload: { enabled: false, // disable configured upload uploadpath: 'Upload/Viewer', // path where upload files should be saved allowedfiletypes: '.jpg,image/tiff', // Allow to upload files with 'jpg' extension and 'image/tiff' MIME-type allowedmaxfilesize: 10 * 1024 * 1024, // Allow to upload files not bigger than 10 MB allowmultiplefiles: true, // Allow to perform upload of several files simultaneously through UI allowdragdrop: true, // Allow to add files for upload using Drag-and-Drop filesuploadconcurrency: 2 // Only two files upload AJAX requests can run simultaneously closeuiafterupload: false // Control will require user action to close it after upload. } });
```

Upload API

WDV provides API to perform files upload programmatically, this includes methods and events on both client and server APIs.

Client-side JS API

The client-side API for upload includes new methods to actually upload files - uploadFile and uploadFiles. These methods only upload files to the specified folder without any additional filtering from viewer config. To perform this filtering before the upload, a new method was added filterFilesForUpload. By separating these two operations, the developer can collect all

files for upload and validate them if needed, before sending any requests and upload operation start.

To track the upload operation status and control it new events were introduced: fileaddedtoupload, uploadstarted, uploadfinished, fileuploadstarted, fileuploading, fileuploadfinished, fileuploaderror. All these events can be used for additional end-user notification about process progress and gives an ability to abort operation at any moment.

This feature also supports server requests customization from beforehandlerrequest event, new type for such requests is a fileupload type.

Clientside events for file upload

- **fileaddedtoupload -**

  Triggers when the file added to upload files through UI controls.
- **fileuploaderror -**

  Triggers when file upload has failed or was canceled.
- **fileuploadfinished -**

  Triggers when file upload has finished successfully.
- **fileuploading -**

  Triggers during file upload process. Can be used to track upload progress.
- **fileuploadstarted -**

  Triggers when file upload is started

Clientside Functions for File Upload

- **filterFilesForUpload(files, filteredFilesopt, callbackopt)** -> {Array.<File>}

  Filters files for upload using the settings from config upload section.

  Filters files that should be uploaded using the settings from config upload section. This includes filtering by size, by type and even by name in order to find out files for upload that have same names. It can be useful, because all events in WDV related to upload use filename as a key, thus you can find duplicates and upload uch files in separate uploadFiles method calls.

> This method is fully optional and even if some files failed to pass this filtering, they still can be uploaded to server

Server-side .NET events

On the server-side, the relevant events for file upload are in WebDocumentRequestHandler. These are FileUpload, FileUploaded and FileUploadResponseSend since this feature also supports response customization as others features do. The sample below is demonstrates how using these events and server-side files filtering and saving in custom destinations can be implemented

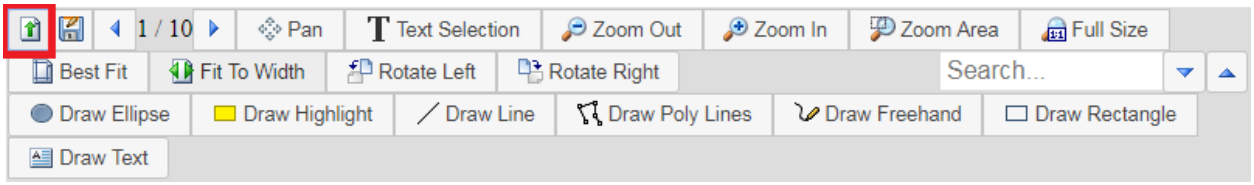
```
ublic class MyWdv : WebDocumentRequestHandler { public MyWdv() { FileUpload +=
MyWdv_FileUpload; FileUploaded += MyWdv_FileUploaded; } private void MyWdv_FileUpload(object
sender, FileUploadEventArgs e) { // Files that should be uploaded in wrong folder // or have
incorrect extension should be rejected if (e.SaveFolder.Contains("StopWord") ||
e.FileName.EndsWith(".exe")) e.Cancel = true; // Some files, for instance, we want to save in
MemoryStream to save to database // for example and not on file system. if
(e.SaveFolder.Contains("database") { e.DestinationStream = new MemoryStream(); } else { // OK
we're NOT sending to database so we will do the file stream here // If you want, you can set
e.DestinationStream to a FileStream right here string newFileName = Guid.NewGuid().ToString()
+ "_" + e.FileName; // this will be passed back to the client as the relative file path to
the uploaded file // they can access it as the filepath in the fileuploadfinished event
client-side e.DestinationName = "/uploads/" + newFileName; // FileStream needs a full path -
in .NET framework ashx handler we get this from HttpContext.Current.Request.MapPath string
fullPath = Path.Combine(HttpContext.Current.Request.MapPath("/uploads/"), newFileName ); //
IMPORTANT: we just open and set it here // It will be written to and then the FileUploaded
event will fire and you can access it then if you need to further // but if all you neeeded
was to redirect the file you're done e.DestinationStream = new FileStream(fullPath,
FileMode.Create FileAccess.ReadWrite, FileShare.Read); } private void
MyWdv_FileUploaded(object sender, FileUploadedEventArgs e) { // This is how you can take the
incoming and write to a file if (e.Destination.GetType() == typeof(MemoryStream)) { //
remember the stream may not be "rewound" so do that first e.DestinationStream.Seek(0,
SeekOrigin.Begin); // your code here to shove e.DestinationStream (a MemoryStream) to a web
servce? // or perhaps call e.DestinationStream.ToArray() for a byte[] you wil send to a
database? // NOTE you can also set e.DestinationName from here // so for instance you have
code that takes the file and saves it to a database and then returns the record ID back to
the client side e.DestinationName = YourCodeToWriteToDbAndReturnRecordId(
e.DestinationStream.ToArray() ); } //Free memory if needed e.DestinationStream?.Dispose(); }
}
```

Again, please note that if you are not using the file upload feature it is best practice to explicitly disable it. Please see:

HOWTO: Completely Disable Upload Feature in WebDocumentViewer

Upload UI

On the UI side, when upload is enabled, a new button appears on a WDV toolbar in the top-left corner. Using this button end user can upload files through UI. Also, new API methods became available to the developer.



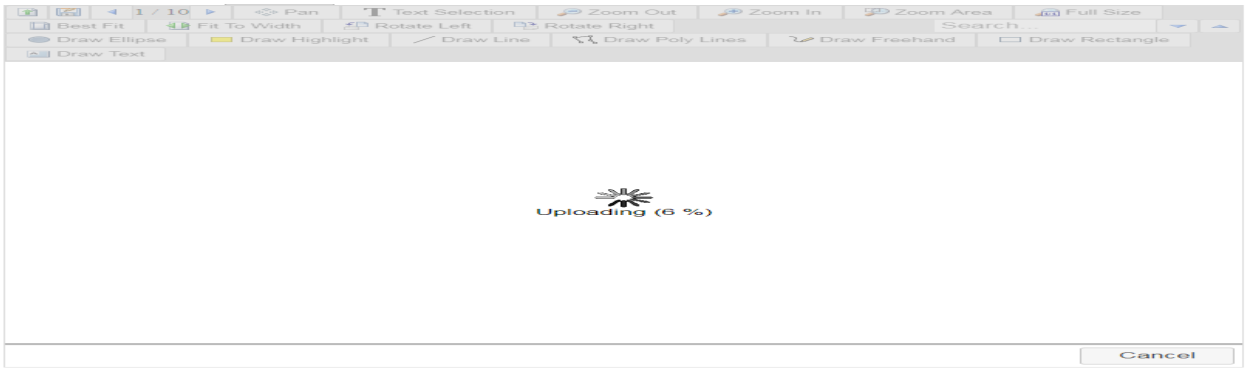
The default scenario with file upload is to perform upload using only open file dialog from a browser. The open file dialog behavior can be customized by configuration properties config.upload.allowmultiplefiles and config.upload.allowedfiletypes.

Upload control

Upload control is a new container for show upload UI and upload progress to end users, it also provides a possibility to handle files drag-and-drop operation in the case when WDV is configured to support it.

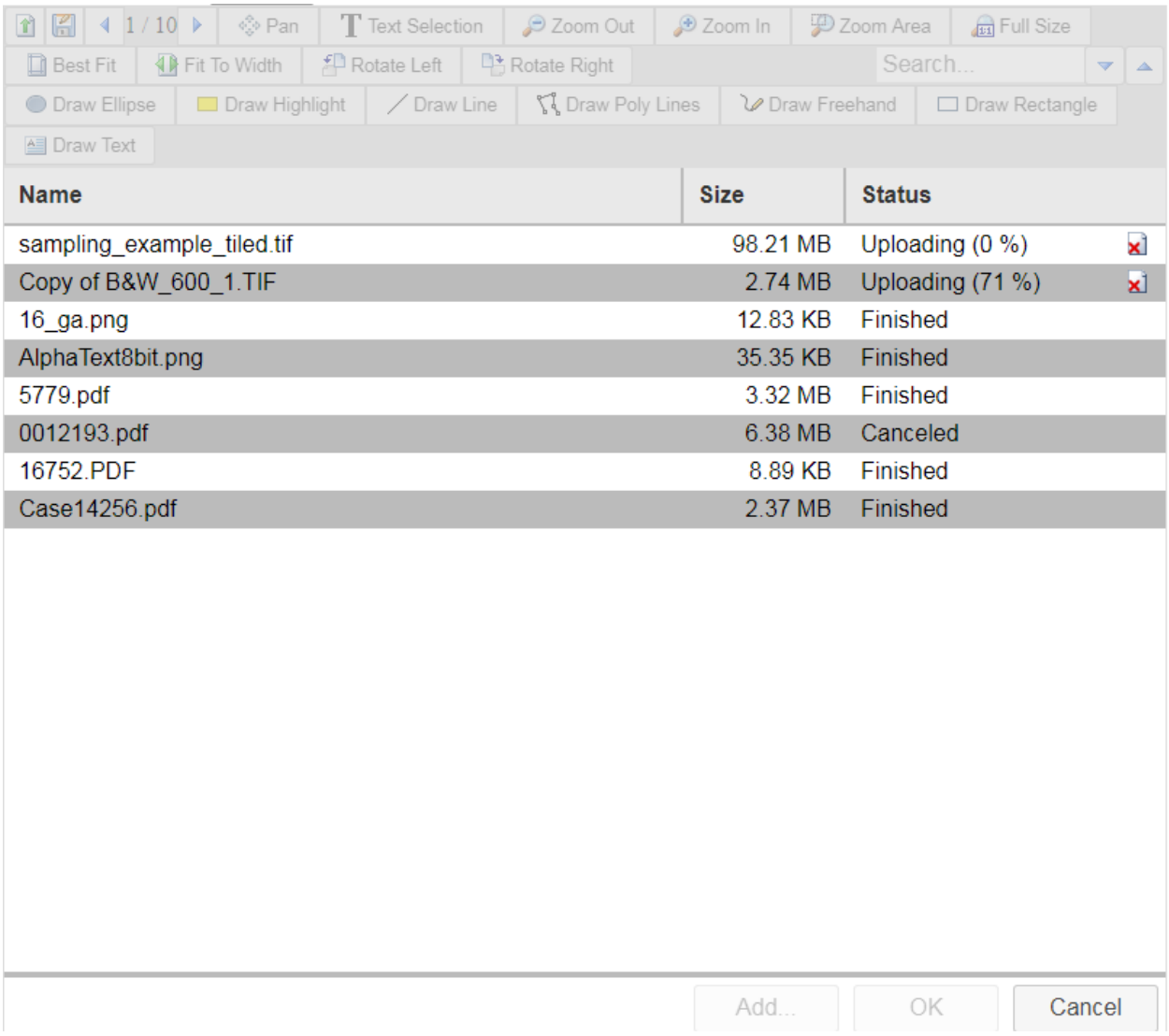
Single file

When a web application is set up to perform an upload of a single file only, then when the user finished selecting files, WDV control is replaced by an upload control, as demonstrated below. This control shows upload progress and provides an ability to cancel the operation at any time, by clicking Cancel button or any thumbnail in WebDocumentThumbnailer (WDT).



Multiple files

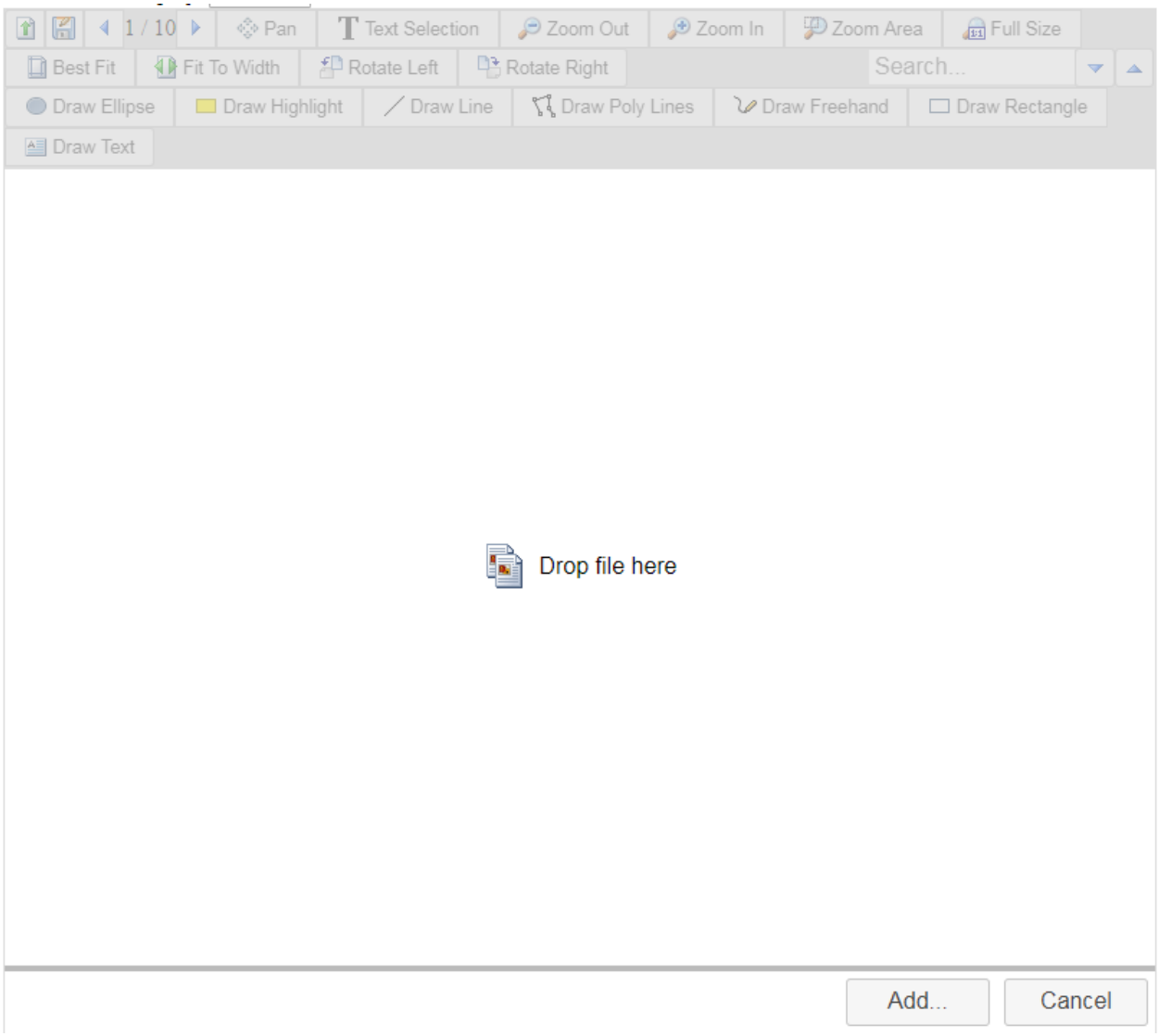
In a case when the application is set up to support multiple files upload, the upload control represents a table with files ready to upload brief information about them. It also provides a possibility to cancel upload for each file individually, even if upload operation has started.



Drag-and-drop

When the developer enables drag-and-drop support for WDV upload feature, then upload control changes its behavior. First of all it starts to support drop files into it, and even

perform some checking to notify user whether or not current dragdata can be used for upload. The other change is a new state, where upload control shows nothing except the notification to drop files into it. Of course the end-user still can add files using open file dialog by open it on click "Add.." button.



UI customization

Upload control supports theming using jQuery-UI themes like also a other viewer parts, but also it provides set of CSS classes to customize it more accurately. Below is the list of using CSS classes for upload control customization:

```
* Buttons related classes. */ .atala-upload-buttons /* Represents the class for internal
buttons container. */ .atala-upload-button /* Represents the base class for upload buttons
Add, Ok, Cancel. */ .atala-upload-close-button /* Represents the class for Close button. */
.atala-upload-cancel-button /* Represents the class for Cancel button. */
.atala-upload-ok-button /* Represents the class for OK button. */ .atala-upload-add-button /*
Represents the class for Add button. */ /* Drag-and-Drop related classes. */
.atala-upload-drag-and-drop /* Represents the class for span element in D&D message. */
.atala-upload-drag-and-drop-image /* Represents the class for image element in D&D message.
*/ /* Upload progress classes. */ .atala-upload-progress-file /* Represents the class for
span element in file upload progress message for single file only. */
.atala-upload-progress-file-image /* Represents the class for image elemen in file upload
progress mesage for single file only. */ /* Control elements structure classes. */
.atala-upload-control /* Represents the class for top-level element in upload control. */
.atala-upload-flex-area /* Represents the class for second-level control container with table
layout. */ .atala-upload-flex-area-non-table /* Represents the class for second-level control
container with non-table layout. */ .atala-upload-flex-buttons /* Represents the class for
second-level upload control container. It's using for buttons container. */ /* File(-s)
upload controls. */ .atala-upload-single-file /* Represents the class for single file upload
container, including drag-and-drop. */ .atala-upload-single-file-no-drag-drop /* Represents
the class for single file upload progress container. */ .atala-upload-multiple-files /*
Represents the class for files table. */ .atala-upload-multiple-files-drag-and-drop /*
Represents the class for drag-and-drop area when multiple files upload enabled. */ /* Files
table classes. */ .atala-upload-files-header-row /* Represents the class for header row in
files table. */ .atala-upload-files-header /* Represents the base class for header cell in
files table. */ .atala-upload-files-header-name /* Represents the class for Name header cell
in files table. */ .atala-upload-files-header-size /* Represents the class for Size header
cell in files table. */ .atala-upload-files-header-status /* Represents the class for Status
header cell in files table. */ .atala-upload-files-row /* Represents the class for file row
in files table. */ .atala-upload-files-cell /* Represents the base class for file cell in
files table. */ .atala-upload-files-cell-size /* Represents the class for files cell Size in
files table. */ .atala-upload-files-cell-status /* Represents the class for files cell Status
in files table. */ .atala-upload-files-cancel-button /* Represents the class for Cancel
button in Status cells. */ /* Miscellaneous. */ .atala-upload-text-element /* Represents the
shared class for all elements in upload control with text. */
```

Last Update

2026-11-11 - TD - (Moved Images)