INFO: Explaining our Different Viewer Controls

Atalasoft provides multiple different viewers, each meeting specific needs, but it can be, at times, overwhelming to figure out what viewer is needed or may be best

Also, when providing support to you with issues repeated to viewers, it is critical that we know exactly which viewer is being used as there are many and subtle differences and if we are thinking you're using viewer X but you are using viewer Y, it will make for a confusing experience for both you and for support

First we need to categorize very generally: There are different viewers for Windows Forms (WinForms), WPF, and for Web, and within each of those there are multiple options

Windows Forms Applications

Our Windows Forms viewers are our most mature and also have the biggest selection the hierarchy is as follows

ThumbnailView

Allows the viewing and navigating (when properly tied to a main viewer (see below) of multi-page images or multiple separate images via small "thumbnail" images that are interactive in multiple ways (depending on how you use them) for click navigation, drag/drop reordering etc..)

FolderThumbnailView

This is a special case of a **ThumbnailView**. Generally **ThumbnailView** is used to show and navigate the multiple pages of a single multi-page document like a TIFF or PDF though it can be used to present disparate single images as a "virtual single document". But the **FolderThumbnailView** is designed to specifically be set to monitor the contents of a single directory and provide access to the first page of each file as a thumbnail to interact with

ImageViewer

This is the most basic individual image viewer. It can be used to provide access to view a single frame of a single image. It has lots of tools for things like tracking where the mouse is on an image, zooming, scrolling etc.. but it is primarily meant only to be used when just viewing... if you will be planning on modifying the image in the viewer with Image Commands, consider moving up to a "higher level" viewer such as **WorkspaceViewer**, **AnnotateViewer**, **DocumentViewer** or **DocumentAnnotationViewer** (see below)

WorkspaceViewer

This is an **ImageViewer** with extra features geared toward when you plan on applying various changes and image commands to the image you're working on. The most critical additions are the ApplyCommand feature and the Undo/Redo capabilities that come with it

With ImageViewer if you wish to apply a change to the image, you would need to directly manipulate the AtalaImage object exposed by viewer.Image and then replace viewer.Image with the new one and viewer.Refresh(); The process might go like this:

```
f (imageViewer1.Image != null) { RotateCommand rotate = new RotateCommand(90); AtalaImage
updatedImage = rotate.Apply(imageViewer1.Image).Image; if (!rotate.InPlaceProcessing) {
imageViewer1.Image.Dispose(); } imageViewer1.Image = updatedImage; imageViewer1.Refresh(); }
```

To do the same with **WorkspaceViewer**, the same operation is just:

```
f (workspaceViewer1.Image != null) { RotateCommand rotate = new RotateCommand(90);
WorkspaceViewer1.ApplyCommand(rotate, "optional Undo/redo caption here"); }
```

WorkspaceViewer also adds a concept of **WorkspaceViewer**.Images which is an *ImageCollection*.

However, PLEASE DO NOT USE EITHER

Long story short: the Open method when provided with a multi-page document such as PDF or TIFF will open all frames in the internal Images *ImageCollection*. This is very convenient for developers, but long history has shown that it's incredibly bad in terms of memory management. Each *AtalaImage* object needs a contiguous block of memory Height (pixels) * Width (pixels) * BitDepth (1, 4, 8, 16, 24, 32, 64 based on PixelFormat) That contiguous block is inefficient and can lead to "badly played game of Tetris" in your application's memory space - in 32 bit apps this causes frequent issues with System.OutOfMemoryException.

Best practice is NOT to use **WorkspaceViewer**.Images or **WorkspaceViewer**.Open() and instead work with ApplyCommand() and directly with **WorkspaceViewer**.Image only.. or if you need to use Open on multi-page documents, consider using **DocumentViewer** or **DocumentAnnotationViewer** depending on if you need annotations or not. (this also applies to **AnnotateViewer**)

AnnotateViewer

AnnotateViewer is a WorkspaceViewer which incorporates an AnnotationController and implements our Annotations framework to allow for drawing, viewing, burning, and manipulating Atalasoft Annotations.

Since **AnnotateViewer** uses **WorkspaceViewer** "under the hood" it also has the concept of **WorkspaceViewer**.Images which is an *ImageCollection*.

However, just as with **WorkspaceViewer**, PLEASE DO NOT USE EITHER

Long story short: the Open method when provided with a multi-page document such as PDF or TIFF will open all frames in the internal Images *ImageCollection*. This is very convenient for developers, but long history has shown that it's incredibly bad in terms of memory management. Each *AtalaImage* object needs a contiguous block of memory Height (pixels) * Width (pixels) * BitDepth (1, 4, 8, 16, 24, 32, 64 based on *PixelFormat*) That contiguous block is inefficient and can lead to "badly played game of Tetris" in your application's memory space - in 32 bit apps this causes frequent issues with System.OutOfMemoryException.

Best practice is NOT to use **AnnotateViewer**.Images or **AnnotateViewer**.Open() and instead work with ApplyCommand() and directly with **AnnotateViewer**.Image only.. or if you need to use Open on multi-page documents, consider using **DocumentViewer** or **DocumentAnnotationViewer** depending on if you need annotations or not. (this also applies to **WorkspaceViewer**)

DocumentViewer

Generally, its rare indeed to use the **ImageViewer** / **WorkspaceViewer** / **AnnotateViewer** which show a single page in full size alone without a **ThumbnailView** to navigate (though it's not impossible). However, the aforementioned viewer controls can allow for some very seriously unsustainable memory usage. Internally if you use these viewers included Open methods (see the lengthy warnings above in **WorkspaceViewer** and **AnnotateViewer** descriptions for details).

DocumentViewer is a CustomControl where we've taken a SplitContainer and placed a **ThumbnailView** in one pane and a **WorkspaceViewer** in the other. We also have implemented a large amount of glue code to ensure best practices for memory management, so that when you use **DocumentViewer**, you can use its **DocumentViewer**.Open() method to open multi-page documents with large numbers of pages and not cause System.OutOfMemoryException

The "cost" to this is that we also somewhat "welded the hood shut". **DocumentViewer** exposes ThumbnailControlProperties in its **DocumentViewer**.ThumbnailControl and exposes the underlying ImageControlProperties under **DocumentViewer**.ImageControl. However, it is critical to understand that you are getting access only to properties and not the individual underlying viewer events. This is done deliberately to avoid interfering with the "glue code" we use to implement the best practices memory management. There are some tasks which are possible with a **ThumbnailView** and **WorkspaceViewer** independently which are not readily available using **DocumentViewer**. There is a way to "dig in" and "extract the underlying controls" so you can tie in to these hidden events and methods and properties. If you find that you really want to use **DocumentViewer** except this one or two events are making it so you're considering backing down to separate controls, please stop and consider creating a support case to ask for our assistance. We may be able to help with the "secret sauce" but you will need to hear us tell you why what we're about to show you is "at your own risk".

DocumentAnnotationViewer

As you've probably guessed, **DocumentAnnotationViewer** is **DocumentViewer** but instead of **WorkspaceViewer**, we use **AnnotateViewer** so that you get a combined control that has thumbnails, image viewing and processing AND annotation support. This is our flagship WinForms viewer control.

Like **DocumentViewer**, in **DocumentAnnotationViewer**, the "cost" to this is that we also somewhat "welded the hood shut". **DocumentAnnotationViewer** exposes ThumbnailControlProperties in its **DocumentAnnotationViewer**.ThumbnailControl and exposes the underlying ImageControlProperties under **DocumentAnnotationViewer**.ImageControl. And under **DocumentAnnotationViewer**.Annotations, you will find access to the

AnnotationController used by the AnnotateViewer embedded within.

In addition to the other issues described with regard to these property accessors in **DocumentViewer** above, you need to be aware that the way **DocumentAnnotationViewer** works is that the current **DocumentViewer**.Annotations consist only of the single *Layer* for the currently selected page in the **DocumentAnnotationViewer**. As you change pages using the ThumbnailView or **DocumentAnnotationViewer**.SelectThumbnail() method, the *AnnotationController* current layer is cached and the new page's layer is loaded.

Strategies for how to iterate all images / annotations can get a little complicated but the excellent AdvancedScanToFile sample application does a great job of providing an example of how to work with this.

As with **DocumentViewer**, if you find yourself fighting with the control because there are one or two events, properties or methods missing that are making it so you're considering backing down to separate controls, please stop and consider creating a support case to ask for our assistance. We may be able to help with the "secret sauce" but you will need to hear us tell you why what we're about to show you is "at your own risk".

WPF Applications

First, a short note: We do not have any form of ThumbnailView class for WPF the way we do for WinForms. There is an article on how you may wish to "roll your own" - it is provided as a convenience as-is but there is no supported WpfThumbnailView in Atalasoft

AtalaImageViewer (WPF Image Viewer)

This is (similar to **WorkspaceViewer** above) the WPF single Image viewer control. It most closely resembles **WorkspaceViewer** on the WinForms side. We have a really helpful sample application: WpfDemo

AtalaAnnotationViewer (WPF Annotation Viewer)

This is (similar to AnnotateViewer above) the WPF Single Image viewer control with Annotation Support. We have a really helpful sample application: WPF Annotations Demo

Web Applications

Modern HTML5

We need to differentiate first.. what do we mean by "Modern HTML5"

Basically, Older web development with Microsoft was using WebForms. This is a technology that has grown very long in the tooth. It was useful in getting many great apps developed and working but it's not kept up well with modern standards and needs. By comparison, our "Modern HTML5" controls are redesigned from the ground up to be more responsive (or at least be able to be used in a responsive app given a bit of correct CSS magic), use "Lazy Loading" and implement "continuous scrolling" while being compatible with modern HTML, CSS and jQuery standards. Furthermore, our "Modern HTML5" controls can be used in new .NET 6 Web applications (MS does not plan on implementing WebForms in .NET Core (.NET 6 and up)

These modern controls rely on WebDocumentRequestHandler (derived from ASHX Generic Handler) for .NET Framework implementation, or WebDocumentViewerCallbacks middleware for .NET 6 for the back end . They use Ajax requests with JSON payloads to communicate with the server back end, and are fully asynchronous. These controls CAN NOT be used in "headless" (client only) applications, they require being hosted on a working IIS server back end.

You MUST NOT cause postback to any page containing these viewers - they function as Single Page Application (SPA) type design.

WebDocumentViewer

This is a standalone web viewer that uses Continuous scrolling (horizontal or vertical configurable) and Lazy Loading to avoid memory issues. It has support for Annotations, viewing, insert, delete, reorder pages and even some operations such as rotation built in. It can also be configured to work in "single page mode" if you want to simulate the legacy controllers feel.

Please see:

INFO: WebDocumentViewer Whitepaper - Getting Started With Web Viewing

INFO:WDV (and WebCapture) In .NET 6 (.NET Core) Whitepaper - Getting Started

WebDocumentThumbnailer

Unlike the legacy viewers where the **WebImageViewer** / **WebAnnotationViewer** (see below) are single image and pretty much must be used with a WebThumbnailViewer (again, see below), **WebDocumentViewer** can work completely on its own using continuous scrolling, but can be optionally tied to a **WebDocumentThumbnailer** (and in fact you can even tie a single **WebDocumentViewer** to multiple **WebDocumentThumbnailer** viewers so that whichever thumb viewer is active is controlling the main viewer.

However, this is optional you do not need to use this control

It is actually a special case of a **WebDocumentViewer**

Using the two together requires a bit of understanding of how they interact (example: when you have just a **WebDocumentViewer** you would use its openUrl(..) method to open a new document, but if you;re working with tied viewers, you would actually use **WebDocumentThumbnailer**.openUrl() to open the image and let the tied viewer just handle it. But if you want to save annotations, you'd do that from the WebDocumentViewer and not from the thumbnail viewer etc.. It takes a bit of getting used to.

Legacy Web Controls (do not use for new projects)

As already mentioned above, "Legacy Controls" are for the old and deprecated WebForms development that had its heyday a decade ago.

Atalasoft still maintains these controls for critical bug fixes, but we are not adding any new features and eventually, these controls will be retired.

It is strongly advised that no new development be done with these controls, (we understand that maintenance and continued support may be required and we'll continue to support them as long as it is possible). Customers are strongly encouraged to consider migrating to the modern HTML 5 controls detailed above at their earliest opportunity.

The **WebImageViewer** and **WebAnnotationViewer** use special custom Ajax requests called "Remote Invoke" and special server side attribute methods [RemoteInvokable] for handling in the code-behind. You MUST NOT cause postback to any page containing these viewers - they function as Single Page Application (SPA) type design.

WebThumbnailViewer

Since the **WebImageViewer** and **WebAnnotationViewer** (see more below) are single image only controls, they are pretty much useless without a **WebThumbnailViewer** to use for navigation. Thus, you will almost never be using one of those without also using a **WebThumbnailViewer**.

WebImageViewer

This is a single image image viewer for WebForms, It relies on a **WebThumbnailViewer** for page navigation. It uses a concept called "RemoteInvoke" to make server side calls to code-behind.

WebAnnotationViewer

This is basically a WebImageViewer (as described above) with added WebAnnotationController added to it

Additional Reading

FAQ: Support for ASP.NET Core / .NET Core / .NET 5 / .NET 6