HOWTO: Enable PdfMultiprocessing In WDV (new in 11.5)

New in 11.5, we've added PDF Multiprocessing.

Background

The PdfDecoder is not "thread safe" and thus has always had locking code to prevent it from running multiple instances at once. This has led to performance issues and bottlenecking in some instances. New in 11.5, we've created a PdfMultiprocessing. It works by creating "silos" where PdfDecoder runs encapsulated /independently - each on a separate core.

Performance impact may be affected as follows:

- Documents with vector content will experience performance improvements.
- High page-count documents with rasterized content may experience performance improvements.
- High page-count documents with vector content, especially if the vector quantity per page is small, may not show substantial performance improvement.
- Some PDF processing use case types may experience a degradation in performance as a result of parallel orchestration and system resource constraints. These use case types include the following:
    - Using the multiprocessing feature for single page PDF documents.
    - Using the multiprocessing feature for PDF documents without any graphics.
    - Using the multiprocessing feature in one-core CPU environment.

Easy-Mode

If you're using WDV "Out of the Box", all you need to do is enable the WDV (and if in use, WDT) multiprocessing: true config option

```
viewer = new Atalasoft.Controls.WebDocumentViewer({ parent: $('#_containerViewer'),
toolbarparent: $('#_toolbar1'), serverurl: _serverUrl, multiprocessing: true,
allowannotations: true, forcepagefit: true, }); _thumbs = new
Atalasoft.Controls.WebDocumentThumbnailer({ parent: $('#_containerThumbs'), serverurl:
_serverUrl, // server handler url to send image requests to documenturl: _docUrl, // +
_docFile, // document url relative to the server handler url allowannotations: true, viewer:
_viewer, multiprocessing: true, });
```

Custom Handling

If you have a custom WebDocumentRequestHandler (or custom middleware for .NET 6+) handling, then you need to do a bit more work.

In order to use the Pdf MultiProcessing, you need to add code in the handler/back end. You will still need to determine if the incoming image is a PDF or not, so that you can either handle the image processing normally (non PDF) or direct through the PdfDecoderMultiprocessor.

IsPdf

You need a way to know if a file is a PDF or not. The problem is if you use RegisteredDecoders.GetDecoder and check if the returned decoder is a PdfDecoder, the problem is that the check ends up still using PdfDecoder - thus running into the single thread bottleneck.

So, we can use PdfDoc.Examiner.ExaminerResults

```
tatic bool IsPdf(Stream inStream) { bool isPdf = false; if (inStream != null) {
inStream.Seek(0, SeekOrigin.Begin); try { //ExaminerResults res =
ExaminerResults.FromStream(inStream, string.Empty, string.Empty); ExaminerResults res =
ExaminerResults.FromStream(inStream); if (res != null) { isPdf = res.IsPdf; } } catch
(Exception ex) { // log maybe? } inStream.Seek(0, SeekOrigin.Begin); } return isPdf; }
```

DocumentInfoRequested

The DocumentInfoRequested event event requires at minimum, to provide PageCount and PageSize, where the PageCount is the total number of pages in the target document, and the PageSize is a System.Drawing.Size of the first page of the document

We can use the Document class to get these. However, if the file is NOT a Pdf, we can't use Document, so we have to do a bit of if/else.

```
rivate void WebDocumentViewerHandler_DocumentInfoRequested(object sender,
DocumentInfoRequestedEventArgs e) { string fullPath =
```

```
HttpContext.Current.Request.MapPath(e.FilePath); int count = 0; Size pageSize; using
(FileStream inStream = new FileStream(fullPath, FileMode.Open, FileAccess.Read,
FileShare.Read)) { if (IsPdf(inStream)) { Document doc = new Document(inStream); count =
doc.Pages.Count; pageSize = doc.GetPageSize(0, (int)_pdfDecoderMultiprocSettings.Resolution);
} else { count = RegisteredDecoders.GetImageInfo(inStream).FrameCount; inStream.Seek(0,
SeekOrigin.Begin); using (AtalaImage img = new Atalasoft.Imaging.AtalaImage(inStream, 0,
null)) { pageSize = img.Size; } } } e.PageCount = count; e.PageSize = pageSize; }
```

ImageRequested

Where the WebDocumentViewer really does its work is the ImageRequested. Here we once again need to determine if the document is a PDF or if it's not and handle accordingly.

```
rivate void WebDocumentViewerHandler_ImageRequested(object sender, ImageRequestedEventArgs e)
{ string fullPath = HttpContext.Current.Request.MapPath(e.FilePath); using (FileStream stm =
new FileStream(fullPath, FileMode.Open, FileAccess.Read, FileShare.Read)) { if (IsPdf(stm)) {
e.Image = PdfDecoderMultiprocessor.NewAtalaImage(_pdfDecoderMultiprocSettings, stm,
e.FrameIndex); } else { e.Image = new Atalasoft.Imaging.AtalaImage(stm, e.FrameIndex, null);
} } }
```