VirtualViewer® HTML5 V5.1 for Java Administrator Guide

With this release of VirtualViewer® 5.1, our documentation has been modified to make it easier to search as well as more up to date. The documentation for this release has been divided into multiple parts:

- Administrator Guide (this document)
- User functionality and interface guide
- Release notes
- Callback, Client and Java Content Handler API references
- Transition guide to VirtualViewer 5.x from previous versions
- Snowbound supported file format references
- Snowbound OSS documentation
- Snowbound TIFF tags references

Samples documents for testing may be found in the `\Sample-documents` directory of the build. For questions or comments about the documentation, please email documentation@snowbound.com.

An online version of this manual contains information on the latest updates to VirtualViewer 5.1. To find the most recent version of this manual, please download the most recent version from our website at www.snowbound.com/support/manuals.html.

Copyright Information

Part Number: DOC 3.1-VV Java 5.1
Revision: 1
VirtualViewer® HTML5 for Java Release Number: 5.1
Published Date: October 2019
Published by: Snowbound Software Corporation.
309 Waverley Oaks Road
Suite 401

Waltham, MA 02452 USA phone: 1-617-607-2000

Sales: 1-617-607-2010
Fax: 617-607-2002

Contents

About Snowbound Software

For over two decades, Snowbound Software has been the independent leader in document viewing and conversion technology. It plays an integral role in enhancing and speeding document processing for the Fortune 2000. Snowbound excels in providing customers with powerful solutions for capturing, viewing, processing, and archiving hundreds of different document and image types. Thanks to its pure HTML5 technology and multi-environment support (including Java and Windows), Snowbound's products operate across all popular platforms and can be easily integrated into new or existing enterprise content management systems. Nine of the 10 largest banks in the United States (seven of 10 in the world), as well as some of the biggest healthcare providers, government agencies, and insurance companies rely on Snowbound for their mission-critical needs.

Important Phone Numbers and Links

For the most current information about Snowbound and our products, please contact Snowbound Sales at 1-617-607-2010 or questions@snowbound.com.

You may also contact us at http://register.snowbound.com/MQL-contactUs-Website-2017.html.

For sales inquiries, please visit https://mylivechat.com/chatnoscript.aspx?HCCID=17729140.

Release Notes and Product Manuals:

The most current version of this manual along with other documents is available on Snowbound's website: https://www.snowbound.com/support/manuals

Snowbound Target Markets

Snowbound's two flagship products—VirtualViewer® HTML5 (a pure HTML5 document viewer) and RasterMaster® SDK (document/image conversion library)—help organizations and companies across a variety of industries meet their document viewing and conversion needs:

- Medical: Patient record management

- Insurance: Insurance & health insurance claim processing
- Finance: Mortgage processing & financial statements
- Shipping: Full array of shipping documents
- Legal: Claims, briefs, and other court documents

VirtualViewer® HTML5

Easy-to-Use in Any Environment

VirtualViewer® HTML5 is equipped with powerful and sophisticated features and functionality.

True cross-platform support: VirtualViewer® HTML5 is a universal viewer that operates seamlessly on any platform with both a pure Java solution with Java-based server components or a .NET solution.

No Downloads: No application download or client-side installation is required, making it a trouble-free solution for users as well as IT administrators.

Localized UI: The viewer's intelligent localization capabilities auto-detect browser settings and display in the proper language.

High-speed viewing: With advanced server processing, the viewer delivers an extremely high-speed response.

Seamless Integration into ECM Applications: VirtualViewer® HTML5 integrates into existing back end repositories and homegrown applications. Snowbound also offers a variety of out of the box ECM connectors (Alfresco, IBM FileNet, and Open Text/Documentum) with seamless integration.

One Quick & Easy 10 Minute Installation

Installation of VirtualViewer® HTML5 takes less than 10 minutes for POCs on any desktop, laptop, and virtual machine. After the quick and easy install, VirtualViewer® HTML5 is then backed by Snowbound's award-winning and responsive support team. Snowbound's skilled network of system integrators can further enhance the benefits of VirtualViewer® HTML5 with custom integration to your existing system.

Technical Information

Snowbound provides the option of either a 100% Java or a .NET (64-bit) server component. The viewer operates in all modern browsers (Microsoft Edge, Firefox, Chrome, Safari, Microsoft Internet Explorer 11 and mobile browsers).

Server options:

- UNIX servers including Linux, Sun, IBM, HP, Mac
- Windows servers including Server 2016, 2012, 2010, 2008 and 2007. Server 2019 coming soon.

RasterMaster® SDK

RasterMaster® is the industry's leading document/image conversion and imaging library for Java and .NET. It is continually enhanced with new functionality and formats and was developed by Snowbound's experts who have nearly a hundred years of combined imaging expertise.

High-Speed File Conversion

RasterMaster® is the fastest file conversion SDK on the market. Users can quickly convert files on the fly for viewing or batch convert large amounts of document types. Special features, including conversion via Byte Array is also available for high performance applications.

Extensive Format Support

AFP, DWG, JPEG, MO:DCA, PDF, MS Office, TIFF, SVG, PNG, and hundreds more document types are supported. Convert any format to PDF or TIFF to ensure universal compatibility. RasterMaster® also includes both PDF/A and SVG output support, enabling long term archiving and high resolution viewing.

Technical Information

RasterMaster® is available for multiple platforms, including Java and .NET:

- Java: for all computing platforms, including Unix, Linux, Windows, and Mac
- NET (x64): for Windows native applications, including Server 2016, 2012, 2010, 2008, and 2007

Responsive Support

All of Snowbound's products are backed by responsive support. Our expert, responsive internal support team is available to answer your questions and help you install our HTML5 viewer and conversion SDK. A support portal is also available 24x7 for questions and information at https://snowboundsupport.force.com/SupportPortal/CommunityLogin.

Important information

For the latest information, please refer to the Release Notes (releasenotes.md) in your product build directory.

Release Notes and manuals may also be found on the Snowbound website at http://www.snowbound.com/support/manuals.

- Please be advised the previously named “default content handler” and now called the “sample content handler” is actually intended to be used for Proof of Concept efforts but is not a complete connector.
- It is recommended that customers upgrade as soon as possible to the latest release of VirtualViewer (typically offered quarterly). The product is rapidly evolving with new features as well as fixes.
- Snowbound recommends the use of the SVG output format from the server to the browser whenever possible for reducing data size and improving performance, particularly when working with large spreadsheets.
- When working with large spreadsheets, it may be advantageous to try the file breakup option so you’re not working with extremely large downloaded documents that might affect performance.
- VirtualViewer for Java now supports only JRE 1.7+. Previous JRE versions are no longer supported or tested except under special arrangements. It is expected that support will shift to JRE 1.8 in 2019.
- For Windows products, .NET framework versions 4.5.2 and up are now supported
- Web.xml changes: The following parameters in web.xml have been removed:
    - `defaultByteSize`
    - `tiffByteSize`
    - `jpegByteSize`

Getting started

This section explains the system requirements and how to install and verify VirtualViewer® HTML5 on your system. Snowbound Software's VirtualViewer HTML5 for Java viewer works with the latest Java and AJAX technology to create a true zero footprint viewing solution.

This section will aid you with setting up and working with the package included in your zip file, virtualviewer.zip. This zip file installs all of the VirtualViewer HTML5 for Java components.

Licensing

VirtualViewer® HTML5 for Java is delivered as a .zip file including the virtualviewer.zip installation package. The package may vary depending on your version.

Your options are enabled through a SnowboundLicense.jar which contains a license .xml file.

Your SnowboundLicense.jar is included in your delivery in the virtualviewer\WEB-INF\lib directory. You do not need to take any further action.

If you order a new option, Snowbound will provide a new SnowboundLicense.jar to replace the one we provided in your original delivery.

System requirements

The system requirements to successfully run VirtualViewer HTML5 for Java are listed below.

Content server

VirtualViewer HTML5 for Java requires the VirtualViewer Java Content Server in order to function. The VirtualViewer Java Content Server is included in the virtualviewer.zip package.

Validation minimum requirements

The following are the validation minimum requirements:

	Minimum requirements
Processor	64bit
Speed	2.4 GHz dual core
Ram	16GB
Available memory	6GB
SSD or HD	250GB

Performance minimum requirements

The following are the performance minimum server requirements:

	Minimum requirements
Processor	64bit
Speed	3.2 GHz quad core
Ram	32GB
Available memory	16GB
SSD or HD	250GB

Performance recommended requirements

The following are the high performance recommended requirements:

	Recommended requirements
Processor	64bit
Speed	3.4 GHz quad core with hyper-threading
Ram	64GB
Available memory	32GB
SSD or HD	250GB

Servlet container

VirtualViewer HTML5 for Java requires a J2SE or J2EE servlet container to run. We recommended the Apache Tomcat and IBM Websphere application servers.

Other servlet containers such as Weblogic or Jboss may be supported with additional effort. Please contact Snowbound for detailed support information for these and any other special-case systems.

Server Java version

VirtualViewer HTML5 for Java JRE of 1.8 or higher is highly recommend, though JRE 1.7 will be supported through 2019.

Client browser versions

We recommend Firefox, Google Chrome, Microsoft Edge, and Safari for client browser use.

Internet Explorer 11 can be used in most circumstances, but due to age, incompatibility and lower performance, it is not recommended. VirtualViewer may also work with other browsers (such as Opera), but no testing is done to ensure compatibility or performance.

Note: If you are using Internet Explorer 11, VirtualViewer will look, perform, and behave better if it is running outside of compatibility mode. For best performance, please configure Internet Explorer to use normal mode when using VirtualViewer. Quirks mode in Internet Explorer is not supported.

Determining memory requirements

The guidelines below are meant to help you determine the amount of memory required to meet the needs of your organizations and users. Please note that memory requirements can vary wildly depending on the system, file, and organizational structure, and as such, these guidelines should not be considered the final word on the matter, but rather a starting point.

Determining memory required to display documents

The amount of memory required to display a document may be significantly larger than the size of the document that is stored on disk. Just like a road map, the document is folded up and compressed when it is stored. In order to see the document, it must be unfolded (decompressed) and spread out so you can see the whole map. The map takes up much more room when open for viewing. The same is true of online documents.

When a document is open, a black and white letter size page at 300 dpi takes roughly 1MB of memory to display and a color page takes 25MB.

The amount of memory required to view documents varies depending on the size of the documents you are processing and the number of documents you are processing at any one time. The amount of memory needed increases as:

- You go from black and white, to grayscale, to color documents (bits per pixel increases).
- You go from compressed to uncompressed document formats (lossy compression to raw image data).
- You go from low resolution to high resolution documents (dots per inch / - quality increases).
- You go from small index card size images to large blueprint size images (number of pixels increases).

Generally, higher quality documents require more memory to process. Snowbound Software does not have a one-size-fits-all recommendation for memory because our customers have such a variety of documents and different tolerances for the level of output quality. However, you can try doubling the memory available to see if that

resolves the issue. Keep increasing memory until you stop getting out of memory errors. If you hit a physical or financial limit on memory, then you can do the following:

- Decrease the number of documents you have open at any one time.
- Decrease the quality of the images requested by decreasing bits per pixel, the resolution, or the size.

To calculate the amount of memory required for an image, you will need to know the size of the image in pixels and the number of bits per pixel in the image (black and white=1, grayscale=8, color=24). If you do not know the height or width in pixels, but you do know the size in inches and the dpi (dots per inch) of the image, then you can calculate the size in pixels as *(width_in_inches * dots_per_inch) = width_in_pixels*.

To calculate the amount of memory (in bytes), multiply the height, width and number of bits per pixel. Then, divide by 8 to convert from bits to bytes. In other words:

*(height_in_pixels * width_in_pixels * (bits_per_pixel / 8) = image_size_in_ bytes*

This table lists examples of memory requirements based on image sizes:

Image size	Required memory
24-bit per pixel, 640 x 480 image	640 * 480 * (24 / 8) = 921600 bytes
1-bit per pixel, 8.5" x 11", image at 300 dpi (2550 pixels by 3300 pixels)	2550 * 3300 * (1 / 8) = 1051875 bytes
24-bit per pixel, 8.5" x 11", image at 300 dpi (2550 pixels by 3300 pixels)	2550 * 3300 * (24 / 8) = 25245000bytes (25 megabytes)

Determining memory requirements based on the number of users and pages viewed

To calculate the amount of memory needed based on the number of users and potential pages viewed at any given time, you can use the general formula:

*The number of concurrent users * size per page in MB * 5 pages in view = required memory*

The table below offers some examples:

Users pages	Required memory
Black and white page (100 dpi) .1MB per page x 5 pages = .5MB x 1000 users	500 mb =~ 0.5GB
Black and white page (300 dpi) 1MB per page x 5 pages = 5MB x 1000 users	5000 mb =~ 5GB
Color pages (300 dpi) 25MB per page x 5 pages = 125MB x 1000 users	125000 mb = ~122GB

Installing VirtualViewer HTML5

To install VirtualViewer HTML5 for Java, follow the steps below:

1. Extract the virtualviewer.zip file to a directory.

2. The extracted .zip file includes the virtualviewer.war file.

3. Save the virtualviewer.war file to the location where you want to install it. Please note that the application needs to be added to a web server before it can be run. Open the virtualviewer.war file in an archive utility such as 7-zip.

4. In the /virtualviewer directory, you will see the extracted files for VirtualViewer HTML5 for Java. If you change the default directory from /virtualviewer, please see Changing the Default Directory for more information about how to successfully change the default directory.

5. Find the web application (webapps) directory where you want to install the files. For example: C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps

   In Tomcat 8.0, documents will load using absolute path and not rel- ative path. Use the following example for the path for Tomcat 8: C:\Program Files\Apache Software Foundation\Tomcat8.0\webapps\virtualviewer\Sample-Documents

6. From the extracted zip directory, copy the virtualviewer directory as a new subdirectory under your webapps directory. If you are using a web server other than Tomcat this may be a different location.

   Note: For other web servers, you may need to take web application deployment steps that are specific to that type of web server. If you have trouble installing VirtualViewer HTML5 for Java, try exploding the .war file. Uninstall the application. Extract the contents of the .war file and deploy using the root directory /virtualviewer.

7. Verify that the VirtualViewer HTML5 for Java content server is running by looking at the web server logs and search for VirtualViewer. You should see the start up message. If you do not see VirtualViewer, then search for snow which may be in an error message. If you still do not get a result, then please refer to the web application deployment documentation for your web server for information on troubleshooting web application deployment and start up issues.

The default port for VirtualViewer is **8080**. To configure a different port, set the codebaseparameter in the web.xml file located in virtualviewer\WEB-INF to the appropriate value. The default parameter is provided below:

```
<init-param>
<param-name>codebase</param-name>
<param-value>http://localhost:8080/virtualviewer</param-value>
</init-param>
```

After completing these steps, we recommend restarting Tomcat or your web application to ensure that your changes have taken effect.

Verifying your installation

Once all components have been installed, VirtualViewer® HTML5 for Java will start up from any supported browser. No client components are needed on the client machine.

To start VirtualViewer HTML5 for Java, open your *.html file in a browser. In most cases, this will be index.html. Ensure that your web service (for example, Tomcat) is running when you open the viewer in your browser.

Viewing sample documents

Snowbound Software provides sample documents along with the VirtualViewer HTML5 for Java installation to get you started. The sample files are located in the Sample-Documents subdirectory.

The web.xml file delivered with VirtualViewer HTML5 for Java located in virtualviewer\WEB-INF specifies the Sample-Documents subdirectory as the default location of the sample files in the filePath parameter.

To view the sample documents, enter the URL for your server into a browser and specify the document name after documentId with the document name as shown below:

```
http://[your server:port]/virtualviewer/index.html?documentId=[someFile.tiff]
```

So for example, if you wanted to view a permitting document and were running Tomcat locally on port **8080** (the default port for VirtualViewer), you might enter http://localhost:8080/virtualviewer/index.html?documentId=BostonPermit.tif into your browser.

If you are able to see all of the documents that came in the Sample-Documents directory of your VirtualViewer HTML5 for Java installation, then you have successfully installed it.

To view other documents, specify the filename after the documentId in the URL.

If you are not able to see the documents in the viewer, please see the Troubleshooting page. If you are still not able to see the documents, please file a ticket with Snowbound Support at support.snowbound.com.

Viewing your documents

Once you've verified the sample documents, you can move on to verifying that your documents appear correctly by placing them in the Sample-Documents directory and then specifying the document's file name after the documentId in the URL.

For example, if you want to display the file named test.tif, add that file to your Sample-Documents directory and test.tif after documentId as shown in the following example:

```
http://localhost:8080/virtualviewer/index.html?documentId=test.tif
```

The documentId should be a filename if the sample content handler is used. Otherwise, it can be whatever the custom content handler expects for a documentId. For more information about custom content handlers, please see Working with the Content Handler.

Capacity Planning

VirualViewer HTML5 for Java performs well for your number of users, you may want to do some capacity planning.

Please see the following algorithm for calculating memory for 200 users. For this calculation, we assume five pages will be active in VirtualViewer HTML5 for Java at any one time (one page being displayed, four thumbnails).

*users * pages * page-seize = memory required for 200 users.*

For example:

- For 200 concurrent users viewing 5 1-bit color pages, the required Java heap capacity is:
  *200 users * 5 pages * 1 mb /pg = 1,000 mB = 1GB.*

- For 200 concurrent users viewing 5 24-bit color pages, the required Java heap capacity is:
  *200 users * 5 pages * 24 mb /pg = 24,000 mB = 24GB.*

For CPU size, generally, the larger the CPU and number of cores, the faster the response time. We expect VirtualViewer HTML5 for Java to use all available resources to display the documents as quickly as possible. It will peg the CPU for short periods during heavy use.

Default Configuration Maximizes Performance

Please note that the default configuration for VirtualViewer HTML5 for Java is set to maximize performance. The default settings are the following:

- The bit depth settings for vector formats such as PDF and Word are set to 1. **Note**: With the bit depth set at 1, color formats will display as black and white. To view these files in color, set the bit depth to 24.
- The DPI settings for vector formats such as PDF and Word are 200. To increase the quality of an image, set the DPI to a higher value such as 400.
- The default format is set to `TIFF_FAX_G4`. If you are trying to view another format in color, set the format parameter to the format type.
-

To improve performance and the speed of loading documents in VirtualViewer Java Content Server, try setting the values of the following parameters in the web.xml file as shown below:

```
<param-name>documentCacheSize</param-name>

<param-value>1024000</param-value>

<param-name>wordBitDepth</param-name>

<param-value>1</param-value>

<param-name>wordDPI</param-name>

<param-value>100</param-value>

<param-name>wordFormat</param-name>

<param-value>JPEG</param-value>

<param-name>pdfBitDepth</param-name>
```

```
    <param-value>1</param-value>

    <param-name>pdfDPI</param-name>

    <param-value>100</param-value>

    <param-name>pdfFormat</param-name>

    <param-value>JPEG</param-value>

    <param-name>xlsBitDepth</param-name>

    <param-value>1</param-value>

    <param-name>xlsDPI</param-name>

    <param-value>100</param-value>

    <param-value>xlsFormat</param-value>

    <param-value>JPEG</param-value>
```

> **Note**: Increasing the value of the `documentCacheSize` parameter will improve performance on the client, but will require the server to keep more content in memory and thereby decreasing performance. It is important to find the right balance between the two by performance tuning the cache size during testing.

Recommended JRE Memory Settings

The amount of memory required to display a document may be significantly larger than the size of the document that is stored on disk. Just like a road map, the document is folded up and compressed when it is stored. In order to see the document, it must be unfolded (decompressed) and spread out so you can see the whole map. The map takes up much more room when open for viewing. The same is true of online documents. When a document is open, a black and white letter size page at 300 dpi takes roughly 1MB of memory to display and a color page takes 25MB.

The amount of memory required to view documents varies depending on the size of the documents you are processing and the number of documents you are processing at any one time. The amount of memory needed increases as:

- You go from black and white, to grayscale, to color documents (bits per pixel increases).

- You go from compressed to uncompressed document formats (lossy compression to raw image data).
- You go from low resolution to high resolution documents (dots per inch / quality increases).
- You go from small index card size images to large blueprint size images (number of pixels increases).

Generally, higher quality documents require more memory to process. Snowbound Software does not have a one-size-fits-all recommendation for memory because our customers have such a variety of documents and different tolerances for the level of output quality. However, you can try doubling the memory available to see if that resolves the issue. Keep increasing memory until you stop getting out of memory errors.

If you hit a physical or financial limit on memory, then you can do the following:

- Decrease the number of documents you have open at any one time.
- Decrease the quality of the images requested by decreasing bits per pixel, the resolution, or the size.

To calculate the amount of memory required for an image, you will need to know the size of the image in pixels and the number of bits per pixel in the image (black and white=1, grayscale=8, color=24). If you do not know the height or width in pixels, but you do know the size in inches and the dpi (dots per inch) of the image, then you can calculate the size in pixels as *(width_in_ inches * dots_per_inch) = width_in_pixels*.

To calculate the amount of memory (in bytes), multiply the height, width and number of bits per pixel. Then, divide by 8 to convert from bits to bytes. See the following example:

*(height_in_pixels * width_in_pixels * (bits_per_pixel/ 8)) = image_size_in_ bytes*

The table below lists examples of memory requirements based on image sizes:

Image Size	**Required Memory**
24-bit per pixel, 640 x 480 image	640 * 480 * (24 / 8) = 921600 bytes
1-bit per pixel, 8.5" x 11" image at 300 dpi (2550 pixels by 3300 pixels)	2550 * 3300 * (1 / 8) = 1051875 bytes
24-bit per pixel, 8.5" x 11" image at 300 dpi (2550 pixels by 3300 pixels)	2550 * 3300 * (24 / 8) = 25245000 bytes (25 megabytes)

Determining memory requirements based on the number of users and pages viewed

To calculate the amount of memory needed based on the number of users and potential pages viewed at any given time, you can use the general formula:

*The number of concurrent users * size per page in MB * 5 pages in view = required memory*

The table below offers some examples:

Users pages	Required memory
Black and white page (100 dpi) .1MB per page x 5 pages = .5MB x 1000 users	500 mb =~ 0.5GB
Black and white page (300 dpi) 1MB per page x 5 pages = 5MB x 1000 users	5000 mb =~ 5GB
Color pages (300 dpi) 25MB per page x 5 pages = 125MB x 1000 users	125000 mb = ~122GB

Caching to Improve Performance

The document cache keeps documents that VirtualViewer has displayed in server memory so that they do not have to be re-rendered the next time they are viewed. This enhances performance but consumes memory on the server. VirtualViewer lets you determine and configure this trade-off between speed vs. memory consumption. Overall cache use can be limited at the web server level.

When caching is enabled, the VirtualViewer content server caches the entire document in memory. The HTML5 server caches the pages it receives from the content server. If the content server and HTML5 server are on the same machine (this is common), they will use the same cache.

The caching configuration parameters mentioned below are in the `RetrievalServlet` portion of the content server's web.xml file. These are documented in Servlet Tags for web.xml.

Do you need caching at all?

If your users never view the same page twice, set the `documentCacheSize` parameter to 0 to turn off document caching.

```
<init-param>
   <param-name>documentCacheSize</param-name>
   <param-value>0</param-value>
</init-param>
```

Sizing the Cache if You Need It

If your users view the same pages frequently, calculate the number of these pages that should be cached away for faster viewing. This will be `numberOfPagesToCache`.

Next, determine how much memory will be used to cache each page, `sizeOfPageInBytes`. To calculate the `sizeOfPageInBytes` value you will need to know:

- Your page size in inches.
- If the pages are in black & white or in color. Black & white pages use 1-bit per pixel (bpp). Color uses 24-bpp.
- The desired resolution in dots per inch (DPI). 100 DPI is fine if the document will not be zoomed or printed. A higher DPI may be required if users are zooming in to look at details.

The size can be calculated using the following formula:

*(height_in_pixels * width_in_pixels * bits_per_pixel) / (8 bits per byte) = image_size_in_bytes*

The value is in bytes and describes the uncompressed size of the page, so it may look rather large.

Below are some examples:

One black and white 8.5x11 inch page at 100 DPI = (8.5 inches 300 dpi * 11 inches * 300 dpi) * (1 bit per pixel / 8 bits per byte) = 116875 bytes or 0.1MB per page for* ***sizeOfPageInBytes****.*

One black and white 8.5x11 inch page at 300 DPI = (8.5 inches 300 dpi *11 inches * 300 dpi) * (1 bit per pixel / 8 bits per byte) = 1051875 bytes or roughly 1 MB per page.*

One color 8.5x11 inch page at 300dpi is 25245000 or roughly 25MB per page.

Now, multiply *numberOfPagesToCache* * *sizeOfPageInBytes*. Set the `documentCacheSize` to the calculated cache size to turn on document caching.

If this number is larger than the memory you have available on the system, you can adjust things like the document bit depth, resolution, or number of pages cached.

For example, a cache size of 1024000 (1GB) will hold 40 full 24-bit color pages at 300 DPI or 1,000 black and white pages at 300 DPI or 9,187 black and white pages at 100 DPI.

Cache maintenance

If your users modify and save the documents being viewed, set the `clearCacheOnSave` parameter to true (the default) so that older versions of the page are not displayed.

```
<init-param>
<param-name>clearCacheOnSave</param-name>
<param-value>true</param-value>
</init-param>
```

The cache is primarily maintained by your application server. The disk cache will be cleared of all temp files at VirtualViewer startup.

If you start seeing Out of Memory (-1) errors, then you may need to resize your cache as described above.

Revalidate cache method called for every page

This is a short timespan cache to store answers from `validateCache` for each session/user. Every x minutes the cache will be deleted for each user (with storing and retrieval handled separately). This provides performance benefits to some users.

For whatever the specified window is (zero will check every time) we will cache the validation for that amount of time based on `sessionId`, `documentId` and HTTP action (`GET` or `PUT`). Once that time elapses, we will revalidate.

The time span value applies to both storage and retrieval.

The validation cache is defined in ehcache.xml with the document cache in a section for "vvValidationCache". By default, validations will expire after five minutes, although that is configurable in ehcache.xml.

When Does the Cache Get Cleared?

The cache gets cleared when the user clicks the **Save** button.

The parameter clearCacheOnSave in the web.xml must be set to true and the documentCacheSize in the web.xml must be greater than 0 to allow for caching.

To clear the Documentum caches if using Documentum, use the following JavaScript method to clear the diskCache for the current session:

```
VirtualViewer.invalidateServerSessionDiskCache();
```

Use the following JavaScript method to clear the diskCache for all sessions:

```
VirtualViewer.invalidateAllServerDiskCaches();
```

> **Note**: The above assumes that useSessionDiskCache is not set to false in the web.xml.

When the Cache Size Is Reached

If you do not use the Save Document functionality, does the cache recycle on its own or does the cache get overwritten once the cache size is reached?

The cache size is determined by the documentCacheSize parameter set in web.xml. If the document is too big, it is not cached. The application will revert to using the getDocumentContent method in the content handler instead of retrieving the documents from the cache.

The documentCacheSize parameter limits the maximum size of a document (compressed) allowed to be put in the cache. The setting does not limit the overall cache size. The document size is logged if caching is on, log- level=finest and the document is too big to fit in the cache.

If the document is larger than `documentCacheSize` (in bytes), it will not enter the cache and VirtualViewer will log something such as:

For key: documentId=MyBig.TIF, data size(2710838) > capacity (1024000) (Not caching)

When this happens, VirtualViewer will call `getDocumentContent` for the document every time it renders the main display or thumbnail since the document is not in cache. Calling `getDocumentContent` multiple times for the same document is usually a significant and unnecessary performance hit which is why you should make the document cache big enough to hold your largest documents.

When the document is within the `documentCacheSize` limit, it is allowed in the cache. If the cache is already full, the cache will remove the oldest document(s) until there is enough room to put the new document into the cache.

This is known as First In First Out (FIFO). The documents are identified by id, not by content. There is no pooling of documents with the same content and different ids.

The `documentCacheSize` parameter defines the maximum size of the heap allocated to VirtualViewer to use for document caching. If the `-Xmx` set in the web server for VirtualViewer is less, the cache will be limited to the smaller value. VirtualViewer may use any remaining heap memory for rendering pages and other operations.

Monitoring the Cache Size

Is there any way to monitor the cache size and have an alert when the cache size is about to be reached?

The system logs will say that the application cannot cache the document because the document exceeded the cache size. The application will retrieve the document with the content handler method `getDocumentContent` instead of retrieving from the cache.

Cache Setting in Tomcat

Tomcat has a setting to limit how much cache it can consume. Set this limit to stop the cache from growing at that threshold. Keep in mind it does not clear cache. It just limits it so there is no build up and eventual OOM exceptions.

In the Tomcat application server, find the Tomcat Monitor application. Open the Tomcat Monitor and then click on the **Java** tab. At the bottom, you will see the settings for Maximum memory pool. Use this to control not only how much is cached by one single application, but how much Tomcat will cache collectively. The value here is you can

control how much VirtualViewer bytes are cached as well as all other applications. Thus, managing memory at a much higher level.

Caching and security

Snowbound Software has no mechanism to selectively remove cached content. However, a Cache Validation interface is provided so you can customize when cached content is permitted to be retrieved. You can implement the `validateCache` method to use your authentication system to validate that the current user is authorized to access the cached page content.

Working with the Content Handler

What is the Content Handler?

The VirtualViewer HTML5 for Java content handler is a Java class that the servlet will call on to perform various actions concerning the retrieval and storage of content.

VirtualViewer HTML5 for Java comes with a sample file content handler that connects VirtualViewer HTML5 for Java to your file system. The sample content handler that connects to the server's File System is provided as a starting point to integrate VirtualViewer HTML5 for Java with your document storage.

Additional out-of-the-box content handlers are being added as well as custom variations. Support for Alfresco, Pega Systems, FileNet P8 and Documentum Webtop are all available in various forms. You can create your own custom connector or use Snowbound Professional Services to create a custom content handler for you.

By default, the VirtualViewer HTML5 for Java servlet will use the sample content handler that Snowbound Software provides. `SampleContentHandler.java`, reads and writes to a file system location. You can find this sample content handler at virtualviewer\Sample-Code\Java Content Handler. It displays files from the .\sample-documents directory.

The sample content handler is not intended for production use. We encourage you to use this as a starting point for writing your own custom content handler to integrate VirtualViewer HTML5 for Java into back-end systems. You should create your own content handler to serve up documents from locations that work for your company as well as to add error handling and more robustness for handling requests from multiple users.

How the Content Handler Works

Whenever VirtualViewer HTML5 for Java requests a document, the servlet will first check the cache to see if the document is present. If it is not, it then calls into the content handler for the document. The order of action is as follows:

1. `getDocumentContent`
2. `getAnnotationNames`
3. `getAnnotationContent` (once for each layer name returned by getAnnotationNames)
4. `getBookmarkContent`

Whenever the user chooses to save the document by choosing **Save Document**,

VirtualViewer HTML5 for Java passes the appropriate data to the servlet, which calls the content handler method `saveDocumentComponents`.

Inside `saveDocumentComponents` the following methods should be called separately when the appropriate data has changed:

- `saveDocumentContent`
- `saveAnnotationContent`
- `saveBookmarkContent`

Other methods within the content handler are called by various functions in VirtualViewer HTML5 for Java.

Defining a Custom Content Handler

The Content Handler APIS are now documented in our Javdocs located on our website at https://docs.snowbound.com/virtualviewer/5.1.0/java/content-handler-api/. Also useful are the sample files found in in the VirtualViewer .war file under \Sample Code\Java Content Handler\SampleContentHandler.java in your build directory.

The document content handler/connector that VirtualViewer will use is set using the `contentHandlerClass` parameter in the application's web.xml file. Many customers create a custom content handler class that integrates with their document management and security systems.

Please see the following example for specifying a custom content handler:

```
<init-param>

<param-name>contentHandlerClass</param-name>

<param-value>com.snowbound.VirtualViewer.custom.MyContentHandler

</param-value>

</init-param>

<param-name>contentHandlerClass</param-name>

<param-value>com.mycompany.viewer.DocumentConnector</param-value>
```

VirtualViewer would then look for and invoke your custom content handler at ./WEB-INF/classes/com/mycompany/viewer/DocumentConnector.class.

> **Note**: The `filecontentHandler` will handle byte arrays, but it does not support input streams.

To use the sample/default File Connector specify:

```
<param-name>contentHandlerClass</param-name>

<param-value>com.snowbound.VirtualViewer.FileContentHandler</paramvalue>

<param name="filePath" value="C:/MyDocuments"/> // folder on the server

<param name='startFile' value='myLogo.gif'> // will display
C:/MyDocuments/myLogo.gif in the client at start up
```

The source code for the sample file content handler is provided as a starting point for your own custom connector. The code is located in the VirtualViewer .war file under \Sample Code\Java Content Handler\SampleContentHandler.java.

To use the default URL connector (no longer recommended):

```
<param-name>contentHandlerClass</param-name>

<param-value>com.snowbound.VirtualViewer.FileAndURLRetriever</paramvalue>

<param-name>baseURL</param-name>

<param-value>http://www.mycompany.com/</param-value>

<param name='startFile' value='myLogo.gif'> // will display
http://www.mycompany.com/myLogo.gif in the client at start up
```

`VirtualViewerContentHandlerInterface`

This interface defines methods for retrieving content for VirtualViewer HTML5 for Java.

Most of the methods take in a single input parameter, which is an instance of the class `ContentHandlerInput`, an extension of `java.util.Hashtable` which contains the data that is required to implement each method.

Likewise, most of the methods return a single value, which is an instance of the class `ContentHandlerResult`, also and an extension of `java.util.Hashtable` which contains the data required to complete the method.

Authentication

The authentication is added in the content handler in the get and save method implementations, including `getDocumentContent`, `validateCache`, and `saveDocumentContent`.

VirtualViewer passes along any cookies that are associated with the html page. This mechanism is how we support single sign on (SSO). Additionally, if the HTTP session ID is not part of the cookie string, then VirtualViewer HTML5 for Java will automatically add the HTTP session ID to the cookie string. This way, the content handler has the information it needs to verify the current user is authorized to view or save the current document.

If you need to pass more authentication information to your custom content handler, you can use the ClientInstanceId to pass it encrypted in whatever way you like. You have to decrypt it before using it.

When customizing the VirtualViewer content handler to connect to your document storage, you may need to request or store authentication tokens as part of the process.

You can store the tokens in the session object within the content handler. Use the `HttpServletRequest` session object in the content handler to achieve this.

The user can get a handle to `HttpServletRequest` session object in the content handler by using this line of code:

```
HttpServletRequest request = (HttpServletRequest)
input.get(ContentHandlerInput.KEY_HTTP_SERVLET_REQUEST);
```

The user can then get or set session attributes:

```
request.getSession().getAttribute(arg0);

request.getSession().setAttribute(arg0, arg1)
```

Single Sign On (SSO)

Single sign on (SSO) related information can be stored in the `ClientInstanceId` parameter or more often in the HTTP session ID. VirtualViewer HTML5 for Java should

pass along any cookies that are associated with the HTML page that contains VirtualViewer HTML5 for Java.

This mechanism is how we support single sign on (SSO). Additionally, if the HTTP session ID is not part of the cookie string, then VirtualViewer HTML5 for Java will automatically add the HTTP session ID to the cookie string.

Getting Document Content

The examples below show various ways of getting document content. Please also look at the Javadoc content (https://docs.snowbound.com/virtualviewer/5.1.0/java/content-handler-api/) for more information.

Document content as a byte array

Use the following example to get document content as a byte array from a file and adding to `ContentHandlerResult.KEY_DOCUMENT_CONTENT`:

```
* This constant key should be used to store the document content as a
* byte array.
* public final static String KEY_DOCUMENT_CONTENT = "KEY_DOCUMENT_CONTENT";


public ContentHandlerResult getDocumentContent(ContentHandlerInput input)
throws FlexSnapsSIAPIException
{
String clientInstanceID = input.getClientInstanceID();
String key = input.getDocumentId();
String Logger.getInstance().log("FileContentHandler.getDocumentContent(" +
key + ")");
String fullFilePath = gFilePath + URLDecoder.decode(key);
File file = new File(fullFilePath);
ContentHandlerResult result = new ContentHandlerResult();
try
{
result.put(ContentHandlerResult.KEY_DOCUMENT_CONTENT,
ClientServerIO.getFileBytes(file));
}
catch (FileNotFoundException fnfe)
{
```

```
/* Removing stack trace here, as it was unnecessary */
Logger.getInstance().log(Logger.SEVERE, fnfe.getMessage());
throw new FlexSnapSIAPIException("Document not found: "
+ ClientServerIO.makeXssSafe(key));
}
catch (Exception e)
{
return null;
}
result.put(ContentHandlerResult.KEY_DOCUMENT_DISPLAY_NAME, key);
return result;
}
```

Document content as a file

Use the following example to get document content as a file and adding to `ContentHandlerResult.KEY_DOCUMENT_FILE`:

```
* This constant key should be used to store the document content as
* java.io.File object.
* public final static String KEY_DOCUMENT_FILE = "KEY_DOCUMENT_FILE";
* This is used for very large files to help load quickly... this is
* read only.
*/


Large file example


public ContentHandlerResult getDocumentContent(ContentHandlerInput input)


throws FlexSnapSIAPIException
{
String clientInstanceId = input.getClientInstanceId();
String key = input.getDocumentId();
String fullFilePath = gFilePath + URLDecoder.decode(key);
File file = new File(fullFilePath);
ContentHandlerResult result = new ContentHandlerResult();
try
```

```
{
result.put(ContentHandlerResult.KEY_DOCUMENT_FILE, file);
catch (Exception e)
{
return null;
}
result.put(ContentHandlerResult.KEY_DOCUMENT_DISPLAY_NAME, key); return
result;
}
```

Document content as a vector

Use the following example to get document content as a vector and adding to `ContentHandlerResult.KEY_DOCUMENT_CONTENT_ELEMENTS`:

```
* This constant key should be used to store the content of the various
* elements of the document if the document is made of more than one file.
* If this key is not null, then the KEY_DOCUMENT_ CONTENT will be ignored.

public final static String KEY_DOCUMENT_CONTENT_ELEMENTS = "KEY_
DOCUMENT_CONTENT_ELEMENTS";

*/

filenet example:

public ContentHandlerResult getDocumentContent(ContentHandlerInput input)

throws VirtualViewerAPIException

{

gLogger.log("FileNetContentHandler.getDocumentContent"); String documentId
= input.getDocumentId(); gLogger.log("documentId is " + documentId);

Document filenetDocument = getFilenetDocument(input); Vector vectorOfBytes
= getFilenetContentBytes(documentId, filenetDocument);

ContentHandlerResult result = new ContentHandlerResult(); try

{
```

```
result.put(ContentHandlerResult.KEY_DOCUMENT_CONTENT_ELEMENTS,
vectorOfBytes); result.put(ContentHandlerResult.KEY_DOCUMENT_DISPLAY_NAME,
filenetDocument.getName());

}

catch (Exception e)

{

e.printStackTrace();

}

return result;

}
```

CacheValidator

This interface defines a method that will be called when a document is requested that is in the cache to determine whether or not the cache may be used to retrieve the document, or the normal content handler sequence must be called.

The document cache speeds up access to documents by saving the rendering the first time a document is viewed. When it is viewed for the second time, the rendering can be fetched from the document cache and re-used.

When multiple users are viewing documents, documents that should be secured may end up in the document cache. To prevent a user that does not have permission from viewing a high security document, use the cache validator to check the user's permission before allowing a document to be fetched from the cache for that user.

The cache validator can also be used to prevent high security documents from being stored in the cache.

To use this feature, your custom content handler must implement `com.snowbound.contenthandler.interfaces.CacheValidator` in addition to `VirtualViewerContentHandlerInterface`.

`CacheValidator` method detail

`validateCache`: Determines whether or not the specified cache put or get is allowed.

```
public ContentHandlerResult validateCache (ContentHandlerInput input)

throws

com.snowbound.contenthandler.VirtualViewerAPIException
```

Parameters

A `ContentHandlerInput` object containing the following data:

Key	Type	Description
"KEY_CLIENT_ INSTANCE_ID"	String	Value of the `clientInstanceId` parameter.
“KEY_DOCUMENT_ID”	String	The name or ID of the document.
“KEY_ANNOTATION_ID”		Either `ContentHandlerInput.VALUE_CACHE_GET` or `ContentHandlerInput.VALUE_CACHE_ PUT.`

Returns

A `ContentHandlerResult` object with the following key/value pairs: `ContentHandlerResult.KEY_USE_OF_CACHE_ALLOWED` either `Boolean.FALSE` or `Boolean.TRUE`.

Event Notification and Handling

`eventNotification`

The VirtualViewer HTML5 for Java client sends event notifications to the VirtualViewer HTML5 for Java's content handler on the server whenever the user does something that triggers an audited event. Event triggers include opening a document or going to another page in the document.

```
public ContentHandlerResult eventNotification (ContentHandlerInput input)
```

```
throws

VirtualViewerAPIException
```

Implement this content handler method to receive event notifications.

```
<param name="enableEventNotifications" value="true">
```

The following auditing events trigger event notification:

- Page request
- Save annotation
- Save document
- Print
- Export
- Document close

The sample file content handler logs these events to the web server's log. For example, these events might appear in a TomCat web server log:

```
[exec] 04-30-2012 16:00:13 FileCoNtentHandler.eventNotification

[exec] 04-30-2012 16:00:13 Key: KEY_EVENT_PAGE_

REQUESTED_NUMBER, value: 0

[exec] 04-30-2012 16:00:13 Key: KEY_DOCUMENT_ID,

value: 6-Pages.tif

[exec] 04-30-2012 16:00:13 Key: cacheBuster, value:

0.7112829455936928

[exec] 04-30-2012 16:00:13 Key: KEY_EVENT, value:

VALUE_EVENT_PAGE_REQUESTED

[exec] 04-30-2012 16:00:13 FileCon-

tentHandler.eventNotification

[exec] 04-30-2012 16:00:13 Key: KEY_EVENT_PAGE_

REQUESTED_NUMBER, value: 0
```

```
[exec] 04-30-2012 16:00:13 Key: KEY_DOCUMENT_ID,

value: 6-Pages.tif

[exec] 04-30-2012 16:00:13 Key: cacheBuster, value:

0.1953655708607337

[exec] 04-30-2012 16:00:13 Key: KEY_EVENT, value: VALUE_EVENT_PAGE_REQUESTED
```

The server `loglevel` must be set to **`Info`** in order to see the event notifications. The `logLevel` must be set to **`Finest`** to see all the Key event details in the log file.

You can change the information being logged and the required `loglevel` by modifying the `eventNotification` method in the sample `FileContentHandler`.

The `eventNotification` method in the content handler can be customized to meet your needs. For example, you could add code to send a message to an audit logging system for certain events or change the log messages to match your company's standard format.

Parameters

The `ContentHandlerInput` hash table will contain a variety of elements depending on the type of event being logged, all values are strings:

Key	Type	Description
`"KEY_EVENT"`	String	One of the `VALUE_EVENT_*` values
`"VALUE_EVENT_PAGE_REQUESTED"`	String	The event being logged is a page request.
`"KEY_EVENT_PAGE_REQUESTED_NUMBER"`	String	The page number requested (zero-based)
`"KEY_DOCUMENT_DISPLAY_NAME"`	String	Getting document content as a byte array from a file and adding to `ContentHandlerResult.KEY_DOCUMENT_CONTENT`
`"VALUE_EVENT_SAVE_ANNOTATION"`	String	The event being logged is a save annotation request.

"VALUE_EVENT_SAVE_ANNOTATION_LAYER_NAME_BASE"	String	The base name of the keys containing the layer names. There will be one of these for each layer. For example: "KEY_EVENT_SAVE_ANNOTATION_LAYER_NAME_BASE0"
"VALUE_EVENT_PRINT"	String	The event being logged is a print request.
"KEY_EVENT_PRINT_PAGE_NUMBERS"	String	The page range being printed, in the format '0-4'.
"VALUE_EVENT_EXPORT"	String	The event being logged is a document export request.
"KEY_ANNOTATION_ID"	String	The name of the annotation layer.

Key/Value pairs passed on page request

Key/Value
Key = ContentHandlerResult.KEY_EVENT Value = ContentHandlerResult.VALUE_EVENT_PAGE_REQUESTED
Key = ContentHandlerResult.KEY_DOCUMENT_ID Value = <documentId>
Key = ContentHandlerResult.KEY_CLIENT_INSTANCE_ID Value = <clientInstanceId>
Key = ContentHandlerResult.KEY_EVENT_PAGE_REQUESTED Value = <page number>

Key/Value pairs passed on annotation save

Key/Value
Key = ContentHandlerResult.KEY_EVENT Value = ContentHandlerResult.VALUE_EVENT_SAVE_ANNOTATION
Key = ContentHandlerResult.KEY_DOCUMENT_ID Value = <documentId>
Key = ContentHandlerResult.KEY_CLIENT_INSTANCE_ID

Value = <clientInstanceId>

Key/Value pairs passed on print

Key/Value
Key = ContentHandlerResult.KEY_EVENT Value = ContentHandlerResult.VALUE_EVENT_PRINT
Key = ContentHandlerResult.KEY_DOCUMENT_ID Value = <documentId>
Key = ContentHandlerResult.KEY_CLIENT_INSTANCE_ID Value = <clientInstanceId>
Key = ContentHandlerResult.KEY_EVENT_PAGE_REQUESTED Value = <page number>

Key/Value pairs passed on export

Key/Value
Key = ContentHandlerResult.KEY_EVENT Value = ContentHandlerResult.VALUE_EVENT_EXPORT
Key = ContentHandlerResult.KEY_DOCUMENT_ID Value = <documentId>
Key = ContentHandlerResult.KEY_CLIENT_INSTANCE_ID Value = <clientInstanceId>
Key = ContentHandlerResult.KEY_EVENT_EXPORT_FORMAT_NAME Value = <format>

Key/Value pairs passed on document close

Key/Value
Key = ContentHandlerResult.KEY_EVENT Value = ContentHandlerResult.VALUE_EVENT_CLOSE_ DOCUMENT
Key = ContentHandlerResult.KEY_DOCUMENT_ID Value = <documentId>
Key = ContentHandlerResult.KEY_CLIENT_INSTANCE_ID Value = <clientInstanceId>

Key/Value pairs passed when retrieved from the internal cache

Key/Value
Key = ContentHandlerResult.KEY_EVENT Value = ContentHandlerResult.VALUE_EVENT_DOCUMENT_RETRIEVED_FROM_CACHE
Key = ContentHandlerResult.KEY_DOCUMENT_ID\ Value = <documentId>
Key = ContentHandlerResult.KEY_CLIENT_INSTANCE_ID Value = <clientInstanceId>

Use the following sample `eventNotification` method in a custom content handler to make use of the new event for cache retrieval:

```
public ContentHandlerResult eventNotification

(ContentHandlerInput input)

throws

VirtualViewerAPIException

String eventType = (String) input.get("KEY_EVENT");
```

```
if (eventType.equals("VALUE_EVENT_DOCUMENT_RETRIEVED_FROM_CACHE")) {

String documentId = input.getDocumentId(); System.out.println("Document
retrieved from cache: " + documentId); // do getDocumentContent related
tasks here because getDocumentContent will not be called

}

return ContentHandlerResult.VOID;
```

Returns

A `ContentHandlerResult object` or `null`. The return value is currently ignored.

Event notification rotate page

Use the following example to capture the page rotation event as part of the Event Notification feature set.

```
var pageEvent = function() {

if(vvConfig.enableEventNotification === false) {
return;

}

var uri = new URI(vvConfig.servletPath);
uri.addQuery("action", "eventNotification"); uri.addQuery("KEY_EVENT",
"VALUE_EVENT_PAGE_REQUESTED");
uri.addQuery("KEY_DOCUMENT_ID", virtualViewer.getDocumentId());
uri.addQuery("clientInstanceId", virtualViewer.getClientInstanceId());
uri.addQuery("KEY_EVENT_PAGE_REQUESTED_NUMBER", pageNumber);
```

Extracting Parameters from ContentHandlerInput

The interface for extracting result parameters from a `ContentHandlerResult` can be found in the JAVAdocs at https://docs.snowbound.com/virtualviewer/5.1.0/java/content-handler-api/.

How to return an error for display in the client

There are two ways to return error messages to the client. The method that works with all operations is to throw a `VirtualViewerAPIException`. For example:

```
if (currentSecLevel.equals("0"))

{throw new VirtualViewerAPIException("Security violation detected");
```

For Send and Save operations you may return an error message through `ContentHandlerResult.ERROR_MESSAGE` as shown in the following example:

```
if (currentSecLevel.equals("0")) {

ContentHandlerResult failResult = new ContentHandlerResult();
failResult.put(ContentHandlerResult.ERROR_MESSAGE, "Security violation
detected");

failResult.put(ContentHandlerResult.KEY_DOCUMENT_DISPLAY_NAME, "Security
error");

return failResult;

}
```

Content Handler method documentation

For information regarding specific method APIs, please see the javadocs for your version of VirtualViewer at https://docs.snowbound.com/virtualviewer/.

Document Repository Specific Information

Alfresco

Alfresco Quickshare Support added in 4.10

A logged-in user can create or close a public quickshare link through Alfresco. If a logged-in user accesses that quickshare link, they can see and modify the document through VirtualViewer as normal; if a guest or unauthenticated user accesses the

quickshare link, they will be able to view the document but not save any modifications to the document (or annotations, notes, etc).

Currently the unauthenticated user gets the full VirtualViewer UI with a readable error message if they take any action that tries to save the document. In the future they should get a limited UI that doesn't present those options.

Alfresco Watermark Support added in 4.10

Added support for Watermarks in the Alfresco version of VirtualViewer allowing saving watermarks back into the Alfresco repository. All supported features or functions remain.

Other information available in separate documents.

IBM Filenet P8

Filenet F_CREATOR Tag Support Added

Other information available in separate documents

OpenText Documentum

Information available in separate documents. Contact Snowbound sales.

Pega Systems

Support included in Version 8.1

Configuration guide

Toolbar Configuration

The file user-config/toolbar-config.js now defines everything about the toolbars. Customer admins can go in and manage this instead of modifying index.html directly: VirtualViewer code goes into this configuration file and creates a toolbar button for every configuration entry that it sees.

Toolbar button configuration

There are three parts to toolbar-config.js. First, we have the two big lists: `imageToolbarButtons` for the top toolbar and `annotationToolbarButtons` for the left-hand toolbar. Their names should categorically not be modified, particularly not in the final return statement in toolbar-config.js.

Take, for instance, `imageToolbarButtons`. It looks like this:

```
var imageToolbarButtons = {
"buttonKey": {button configuration),
"anotherButtonKey": {button configuration}
};
```

Each "`buttonKey`" is a unique, descriptive key for the toolbar button. Unique because it's defining the button in the configuration, and also because it's used in the HTML. It should be letters only, and it is good practice for customers to put a prefix on this key. For instance, Big Company, Inc may call a key "`bcMyKey`."

This is the same structure as the annotation toolbar list of buttons. Each button key should also be unique between the two lists--don't put "`vvMyNewButton`" as a key in the image toolbar configuration and in the annotation toolbar configuration.

Now we can look at the button configuration. In the curly brackets, there are a few items:

- `localizeKey`: This is a string referring to a locale file, where the text that appears in the title--what appears in the tooltip and in the dropdowns--is stored. If a customer is creating a whole new button that doesn't have an entry in a locale file, this field can be left blank, but the "name" field *must* be filled in or the button

will appear blank in dropdowns.

- `name`: This is a string with the name of a toolbar button. VirtualViewer buttons do not use this property, since they use localized strings, but customers without an entry in a locale file must put the name of their toolbar button in this field. This should be something short and descriptive.
- `clickHandler`: This is the function that will be called when a user clicks on the toolbar button.
- `iconImage`: This is a string representing a URL to an icon image. VirtualViewer buttons use CSS files to assign icon images, but customers can list an image here and it will be used.
- `addSeparatorAfter`: This is a boolean value. If a group of buttons is too long, it may be necessary to add a bit of whitespace after a button to visually break up the group when the toolbar is fully expanded. If this is true, the viewer will pop some white space after the button. It is not necessary to specify "false" on this item.
- `groupId`: This is a string, and it refers to a group key. Groups are also configured in toolbar-config.js. If a button is in group A, it will be placed in group A in the order it appears--if the Sticky Note button is before the Rubber Stamp button in the annotation toolbar list, and they both belong to group A, the Sticky Note button will appear before the Rubber Stamp button in the UI. If an item is in the annotation toolbar list, it should be assigned to an annotation toolbar group. Additionally, this is optional--if a button has no `groupId`, it will essentially be in an "uncategorized" list. It will appear at the end of the toolbar and will not be included in a dropdown. (See the Layer Manager and Watermarks buttons as default examples.)

Toolbar button group configuration

The final part of toolbar-config.js is the group configuration list, called `toolbarButtonLogicalGroups`. This list looks very similar to the button configuration lists. What it will do is define all the buckets that the toolbar buttons can go into.

Again, each group has a unique string key followed by a configuration object:

```
var toolbarButtonLogicalGroups = {
   "myGroupKey": { configuration },
   "myOtherGroupKey": { configuration }
};
```

The configuration items are as follows:

- `localizeKey`: This is a string referring to a locale file, where the text that appears in the tooltip of the dropdown button is stored. This can be left blank.
- `groupTitle`: Like "name" for the buttons, this is a string with the name of the group in it. For instance, "Preferences" or "Edit" or "File." The localize key will handle this, but a customer without a locale entry could use "groupTitle".
- `iconImage`: This is a string representing a URL to an icon image. VirtualViewer buttons use CSS files to assign icon images, but customers can list an image here and it will be used.
- `annotationToolbar`: This is a boolean flag that chooses where to put the group. If true, the group is an annotation toolbar group; if false or absent, the group is in the top toolbar. Annotation toolbar buttons should only be placed in annotation groups, and similarly, image toolbar buttons should only be placed in image toolbar groups.

Changes in VirtualViewer 5.x

New strings in vv-en.json:

- `utilityToolbar.fileGroup:` File
- `utilityToolbar.zoomGroup:` Zoom
- `utilityToolbar.pagesGroup:` Pages
- `utilityToolbar.pageManipulationGroup:` Page Manipulation
- `utilityToolbar.infoGroup:` Info and Settings
- `annToolbar.textAndStampsGroup:` Text and Stamps
- `annToolbar.markupGroup:` Markup
- `annToolbar.shapesGroup:` Shapes

Feature-specific Configuration

Username public API

This was added so that the user can programatically add a username to their instance of VirtualViewer if they so desired. The user can still use the dialog box in the User Preferences.

API: `virtualViewer.setUsername(string);`

Document Properties

Use the following API methods to manage the display of document properties:

Showing the Information for a document:

```
VirtualViewer.prototype.showImageInfo = function()
```

Hiding the Information for a document:

```
VirtualViewer.prototype.hideImageInfo = function()
```

Toggling the Information for a document:

```
VirtualViewer.prototype.toggleImageInfo = function()
```

The Pages and Documents Panel

The Pages and Documents panel provides a convenient way to:

- Navigate to any page in a document in the Pages panel.
- Select another document to view from the multiple Documents panel.
- Create a new document by dragging and dropping pages from another document.

However, this convenience does have a price. VirtualViewer HTML5 for Java performance degrades because it is processing every page in the document Pages panel and/or the first page of every document in the Documents panel. If you want to speed up performance, you may want to disable or hide the Pages and Documents panel by setting the `showThumbnailPanel` parameter to false in the config.js file as shown in the example below:

```
var showThumbnailPanel = false;
```

Document Notes Panel

The Document Notes Panel allows you to add notes that are relevant to the active document that you are currently working with. It includes the ability to view, create, edit, and delete notes.

The `getDocumentNotes(string)` function in webviewer.js in the js directory will change the note's author to whatever name is specified in the string. The string will replace "User Unknown" with whatever string is entered in this function.

To set the user name in the Document Notes panel, set:

```
virtualViewer.getDocumentNotes("[User's Name]"),
```

For example, if you want to set the user name as Fred:

```
virtualViewer.getDocumentNotes("Fred"),
```

The time stamp is set by the server time for the computer of the user who created the note. The time stamp changes for the server time for the computer of the user when edited.

The Magnifier

The Magnifier functionality is defined by the `magnifierDefaults` parameter in the config.js file as shown below with the default values:

```
magnifierDefaults: { zoomPercent: 150, width: 300, height:150, x:200, y:100 }
```

The `toggleMagnifier()` method to close the Magnifier can be mapped to a shortcut key.

The API call `virtualViewer.setMagnifierPosition(X, Y)` overrides the defaults and will allow coordinates to be passed.

Configuring email documents

To display the Email Document button, set the values for the `emailDefaults` parameter in the config.js file as in the example below:

Set the `prepopulatedForm` parameter to the email address that you want to configure in the config.js file as in the example below:

```
var prepopulateFrom = prepopulatedEmail@domain.com;
```

Set the parameters below in the web.xml file to set the values for your email system.

```
<init-param>
    <param-name>smtpServer</param-name>
    <param-value>...</param-value>
</init-param>

<init-param>
     <param-name>smtpUsername</param-name>
     <param-value>...</param-value>
</init-param>

<init-param>
     <param-name>smtpPassword</param-name>
     <param-value>...</param-value>
</init-param>
```

Text Annotations

The default appearance for a text annotation looks like a yellow sticky note. If you prefer a different look, the `annotationDefaults` config.js configuration parameter sets the default and is customizable.

Please see the following example:

```
// Default appearance options for annotations

annotationDefaults: {

   lineColor: "FE0000",
   lineWidth: 3,
   fillColor: "FE0000",
   stickyFillColor: "FCEFA1", // yellowish
   stickyMargin: 10, // also need to adjust .vvStickyNote in webviewer.css
   highlightFillCOlor: "FCEFA1"
   redactionFillColor: "000000",
   redactionOpacity: 0.5, textString: "Text",
   fontFace: "Arial", fontSize: 14, fontBold: false,
   fontItalic: false,
   fontStrike: false, // for future use
   fontUnderline: false, // for future use
   fontColor: "000000"
}
```

The system administrator has the ability to set the following default values for annotations:

- Line color
- Line width
- Fill color
- Sticky note fill color
- Sticky note margin
- Text string
- Font face
- Font size
- Font bold
- Font Italic
- Font strike

Configuring the Annotations Checkbox

To set the Annotations checkbox in the Export dialog box, set the `exportBurnAnnotations` parameter to true in the config.js file as in the example below:

```
var exportBurnAnnotations = true;
```

Resize text annotations

In **config.js**, set the `autoResizeTextAnnotations` parameter to `true` to dynamically resize annotations. The default value is `false`.

Annotation Commenting

Annotation commenting is display only. Export, Send, Email and Print will not display the annotation comments on the pages.

In config.js, set the `enableAnnotationCommenting` parameter to `true` to enable annotation commenting. Set the parameter to `false` to disable annotation commenting.

Annotation Navigation

The annotation navigation buttons are enabled by setting the `showAnnNavToggle` config.js parameter to `true`. The default value is `false`.

Annotation Redaction Tags

Annotation redaction tagging assigns a categorical value to individual annotations or redaction. To configure your predefined list of annotation redaction tags, add the strings for your tags to the `annotationTags` array in the config.js file:

```
annotationTags: ["Confidential","Redaction","Social Security","Credit
Info"],
```

In config.js, set the `searchRedactionTags` parameter to `true` to turn on search redaction tags. The default value is `true`.

Note: On the Include Annotations dialog box, the `Text` and `Non-Text` options are hidden from the Export Document, Email Document, and SaveAs Document sub-option by default.

To re-enable the `Text` and `NonText` option, change their respective entries from "`display: none`" to "`display :! Important`" in dialog.css.

```
div#vvExportOptionsAnnotationsTypeCheckboxes { display: block
!important;
}
div#vvEmailOptionsAnnotationsTypeCheckboxes { display: block !important;
}
div#vvSaveAsOptionsAnnotationsTypeCheckboxes { display: block
!important;
}
```

Text Rubber Stamp Annotations

The Text Rubber Stamp functionality is enabled when the `enableTextRubberStamp` parameter is set to true and the config.js file contains one or more defined Rubber Stamps.

The system will allow for a limited number of Rubber Stamps with the upper limit of available Rubber Stamps set at ten. To disable this functionality, set the `textRubberStamps` parameter to false in the config.js file as in the example below:

```
var enableTextRubberStamp = false;
```

The system administrator has the ability to set the following pre-defined font characteristics for Rubber Stamps:

```
Font Face: (Helvetica, Times New Roman, Arial, Courier, Courier New)
Font Size: (Any valid integer in range of 2-176)
Font Color: (Any valid HTML color code, specified in hexadecimal)
Font Attributes: (Normal/Bold/Italic)
```

Please see the following example for how we configure the two Rubber Stamps "Approved" and "Denied":

```
var textRubberStamps = [

   { textString: "Approved",
     fontFace: "Times New Roman",
     fontSize: 30,
     fontBold: true,
     fontItalic: true,
     fontColor: "00FF00" }

   { textString: "Denied", fontColor: "FF0000" }

];;
```

Any font characteristics not defined by the system administrator will use the following default system characteristics:

```
{ fontFace: "Arial",
  fontSize: 12,
  fontColor: "FF0000",
  fontAttributes: "Normal" }
```

Image Rubber Stamp Annotations

An Image Rubber Stamp is an image annotation from a pre-defined list. You can define a list of pre-configured/custom Image Rubber Stamps through the `enableRubberStamp` parameter in the web.xml file.

If the `enableRubberStamp` parameter is set to true and one or more Rubber Stamps are defined, then clicking on the **Text Edit** annotation toolbar button will produce the rubber stamp text menu.

> **Note:** If the `enableRubberStamp` parameter is set to `false`, then clicking the **Text Edit** annotation button allows you to select only **Add New Text** to add a text annotation.

The Image Rubber Stamp functionality is defined by the `enableSingleClickImageRubberStamp` parameter in the web.xml file as shown below in a

comma-separated list of names which will be used to pull the individual stamp configurations out of the web.xml:

```
<init-param>
   <param-name>testStamp1,testStamp2</param-name>
   <param-value>8.5</param-value>
</init-param>
```

Set the comma-separated list of names which will be used to pull the individual stamp configurations out of the web.xml. You use these names in the param-name tags as shown in this example:

```
<init-param>
   <param-name>testStamp1</param-name>
   <param-value>
         This is the First Test,300,175,
         http://www.sample.com/sites/sample.com/files/images/Sample.png
   </param- value>
</init-param>

<init-param>
   <param-name>testStamp2<</param-name>
   <param-value>
         This is the Second Test,600,300,http://www.sample.com/sample.pngg
   </param-value>
</init-param>
```

The `param-value` tags are comma separated as follows:

```
<param-value>displayName,stampWidth,stampHeight,stampURL</param-value>
```

- `displayName` -The text that will show up in the pop-up menu in the UI to describe the stamp.
- `stampWidth`/`stampHeight` - The dimensions used when
- `enableSingleClickImageRubberStamp`is enabled in config.js.
- `stampURL` - The URL to the stamp in question. This will be downloaded on servlet startup, converted to PNG, and stored in memory.

Set the `enableSingleClickImageRubberStamp` parameter to true in the config.js file to draw the bounding box when adding a rubber stamp to the image with a single click .It will be sized according to the dimensions specified in web.xml. If false, it will behave like any other annotation.

```
<init-param>
   <param-name>enableSingleClickImageRubberStamp</param-name>
   <param-value>true</param-value>
</init-param>
```

Dynamically Load Custom Image Stamps via Javascript API

The list of available image stamps configured in config.js and can be modified dynamically via a Javascript API. These stamps are loaded with a provided URI, rather than by assuming a stamp's presence and naming convention in the Resources directory.

> **Important!** The web.xml initial parameter `customImageRubberStamps` has been deprecated in favor of the new config.js `initialStamps` list and its related stamp Javascript API.

The configuration item `initialStamps` in config.js holds a list of image stamp objects. An image stamp object consists of a display name for a stamp, the `stampTitle` attribute, and the URI of the stamp image in the `url` attribute. Optional attributes `width` and `height` allow for default dimensions other than the stamp image's native dimensions.

These stamp images may be stored in any accessible location, whether that be locally in the resources/stamps directory within VirtualViewer, or elsewhere on the web or your organization's filesystem.

Several of the functions below take image stamp objects. An image stamp object has four attributes `url`, `stampTitle`, `width`, and `height`. Only `url` and `stampTitle` are required. An example object, from config.example.js, would look like:

```
{
stampTitle: "Approved",
url: "./resources/stamps/Approved.png", width: 535,
height: 293
}
```

Functions that accept image stamp objects include:

- `clearImageStamps` will remove all the stamp options from the Image Rubber Stamp dropdown list in the toolbar.
- `createImageStamp` will add a single image stamp to the Image Rubber Stamp dropdown on the toolbar.
- `url {string}` The URI of the stamp's image.
- `stampTitle {string}` The displayed title of the stamp. This will appear in the list in the toolbar.
- `options {object}` Dimension options for the stamp. The attributes of this object are listed below as options.attribute.
- `options.width {number}` The width of the stamp image. If not provided, the native width of the stamp image will be used.
- `options.height {number}` The height of the stamp image. If not provided, the native height of the stamp image will be used.
- `createImageStampArray` will add a whole array of image stamps to the viewer. The provided stamps will append to the existing list of stamps. A stamp object has mandatory URL and title properties, and optional width and height properties.
- `stampList` {array of image stamps} An array of stamp objects, that require `url` and `stampTitle` properties to be valid, as described above.
- `createImageStampArrayFromFunction` will initialize the viewer's list of image stamps. This takes a callback function as a parameter, and calls that function to get the actual list of image stamps that will be used in the viewer.
- `fn {function}` This function is called with no parameters. It is expected to return an array of image stamp objects, as described above. The array returned by this function will be used to fill the stamp list on the toolbar.

Consolidate Annotation Layers

In config.js, set the `enableCrop` parameter to `true` to enable consolidate annotation layers. Set the parameter to false to disable consolidate annotation layers. This parameter is set to `false` by default.

Substitute Image Thumbnails

You can add a substitute box instead of image thumbnails. This improves performance because image thumbnails do not need to be created.

In config.js, set the `doNotLoadPageThumbs` parameter to `true` to display substitute boxes instead of image thumbnails. The default value is `false`.

If the `doNotLoadPageThumbs` parameter is set to `true`, VirtualViewer will not request thumbnail images. Instead, VirtualViewer displays a box with the page number. Select the substitute thumbnail box as you would an image thumbnail.

```
doNotLoadPageThumbs: true,
```

Use the `thumbPageLabel` string to set the page thumbnail tooltip. Please note that it is important to include the trailing space in `"Page "`:

```
"thumbPageLabel": "Page "
```

Search

The Search tab is enabled by default. The Search tab is enabled by setting the `showSearch` config.js parameter to `true`. Set the parameter to `false` to disable the text searching tab. Please see the example below:

```
var showSearch= true;
```

To determine whether or not text searches should be case sensitive, set the `searchCaseSensitive` config.js parameter to `true` or `false` depending if you want case sensitivity turned on or off. The default value is `false`. See the example below:

```
var searchCaseSensitive = false;
```

Search default colors

You can configure the default colors for the first and second search match by setting the values for the `searchDefaultColor` and the `searchSelectedColor` in the vvDefines.js file found in the js directory as shown below:

```
vvDefines = {
   searchDefaultColor: "rgba(255,78,0,0.2)",
   searchSelectedColor: "rgba(255,255,0,0.2)"
   }
```

OCR Configuration

This is a new OCR option in VirtualViewer. The OCR function allows searching text in an image document (TIFF or PNG initially) as well as selecting text in the VV client after the document has been OCRed.

To OCR a document in the VV client, a user must search for text in a non-text document to get the OCR prompt. The OCRed result is cached; while that result is cached, the user can search for and select text without a further OCR prompt.

The two new parameters necessary to enable OCR are in web.xml (web.config for .NET):

- `enableOcr`: Enable OCR for searching and text extraction. Must have a valid OCR configuration and licensing to function correctly. Defaults to `false`.
- `tesseractDataPath`: Absolute or relative path to Tesseract OCR Engine's training data. If using packed WARs in Tomcat, this needs to be changed to an external unpacked folder. Defaults to "`/tessdata`".

Page manipulation

Disabling Page Manipulations

Page manipulations are enabled by default. To disable page manipulations, the `pageManipulations` parameter must be set to false in the config.js file. This disables the Page Manipulations menu in VirtualViewer HTML5 for Java and enables the Save Annotations menu choice in the File menu.

To disable it, set the `pageManipulations` parameter to false in the config.js file as shown in the example below:

```
var pageManipulations = false;
```

Rotate Specific Pages

Use the following APIs to rotate specific pages:

- `virtualViewer.rotatePageBy(pageNumber, angle)` rotates the current page 0, 90, 180 or 270 (positive or negative) degrees from it's current state.

  So, you call this twice with 90 degrees as the parameter, the final image will be rotated by 180. It returns true if the page is rotated successfully. Otherwise, it throws an error.

- `virtualViewer.rotatePageTo(pageNumber, angle)` rotates the document 0, 90, 180 or 270 degrees absolutely.

  Thus, if you call this twice with 90 degrees as the parameter, the final image will only be rotated by 90 degrees only. It returns true if the page is rotated successfully. Otherwise, it throws an error.

Disabling Copy to New Document

The Copy to New Document functionality is enabled by default. To disable it, set the `pageManipulationsNewDocumentMenu` parameter to false in the config.js file as shown in the example below:

```
var pageManipulationsNewDocumentMenu = false;
```

Split screen

In config.js, set the `splitScreen` parameter to true to enable the Split Screen View feature. If the `splitScreen` parameter is set to `false`, the Split Screen View feature is disabled. The default value is `true`.

```
splitScreen: true,
```

In config.js, set the `screenSizes` parameter to the screen size for panel 1 and panel 2. The first value sets the size of screen panel 1. The second value sets the size of screen panel 2. If the first value is set to 50, the first screen panel is set to 50% of the viewer. If the second value is set 50, the second screen panel is set to 50% of the viewer.

```
screenSizes: [ 50, 50],
```

As images are added by selecting Split Image, each new document request replaces the existing document in the lower panel.

Defining keyboard shortcuts

Keyboard shortcuts are defined in config.js in the hotkeys section as shown below:

```
hotkeys: {
   zoomIn: 'ctrl++,ctrl+=',
   zoomOut:'ctrl+-,ctrl+_',
   exportDocument: 'ctrl+shift+e',
   printDocument: 'ctrl+shift+p',
   lastPage: 'end',
   firstPage: 'home',
   previousPage: 'ctrl+shift+pageup',
   nextPage: 'ctrl+shift+pagedown',
   rotateCounter: 'ctrl+shift+l',
   rotateClock: 'ctrl+shift+r', s
   howKeyboardHints: 'ctrl+/',
   toggleThumbnailPanel: 'ctrl+shift+t',
   fitHeight: 'ctrl+shift+j',
   fitWidth: 'ctrl+shift+w',
   fitWindow: 'ctrl+shift+q',
   panLeft: 'left',
   panRight: 'right',
   panUp: 'up',
   panDown: 'down',
   thumbPageDown: 'ctrl+end',
   thumbPageUp: 'ctrl+shift+end',
   copyText: 'ctrl+shift+c',
   searchText: 'ctrl+shift+f',
   enterSelectTextMode: 'ctrl+shift+insert',
   toggleTextSelectionMode: 'ctrl+shift+d',
   toggleImageInfo: 'ctrl+shift+u'
},
```

Localization

VirtualViewer HTML5 for Java localization supports auto detecting the language settings the user's browser is configured to use. It then looks for a localization file in that language. If a localization file for the corresponding language exists, it will be used to display terms throughout the UI in that language.

For more information on setting language preferences in a browser, please see http://www.w3.org/International/questions/qa-lang-priorities.en.php.

Localization Files

Localized files are stored in the/virtualviewer/resources/locale/ directory.

The english file, named vv-en.json, is located in that directory and can be used as a reference when translating to other languages.

The naming of the localized files should follow the syntax of vv-en.json, replacing "en" with the two-letter code of the language used for the appropriate translation. The two-letter codes follow the ISO 639 code values.

Please visit the following links for additional resources on language codes:

- http://www.loc.gov/standards/iso639-2/php/code_list.php
- http://en.wikipedia.org/wiki/List_of_ISO_639-2_codes

Converting Terms

The terms that are displayed in vv-zz.json using all caps represent where the language specific replacements should be placed.

Each term includes a replacement text for the `alt value` and the `title value`, although these are most likely going to match each other.

The `alt` and the `title` values represent the displayed text that is shown if the image fails to load or when the user hovers the mouse over the image. It can describe the icon, or in the case of VirtualViewer HTML5 for Java, what action is associated with the corresponding icon.

Supporting Accents/Special Characters

To support the translation of terms to languages that use accents or special characters, these accents/special characters must first be converted to Unicode before including it in the translation file. You may also translate the entire string to Unicode, rather than just the accent- s/special characters.

Please visit the following links for additional resources to convert text to Unicode:

- http://www.pinyin.info/tools/converter/chars2uninumbers.html
- http://tokira.net/unicode/index.php

EXAMPLE: Create a French Language Translation File

In order to illustrate how you might create a translation file, we'll walk through an example translating a term in a French language translation file.

The two letter code for French is "fr", so we will copy vv-zz.json and rename the new file **vv-fr.json**.

Now that we've created our translation file, we'll create a translation for the **Rotate Left** button.

In vv-fr.json, look for the corresponding value:

```
"rotateLeft": {
       "alt": "ROTATELEFT.ALT", "title": "ROTATELEFT.TITLE"
              },
```

The French translation for "Rotate Left" is "Rotation à Gauche". In this case, we will use the same value for both the alt and title values.

Converting the accents/special characters in this translation into Unicode results in "Rotation À Gauch." Now, replace the converted result in vv-fr.json as shown below:

Using the converted results in vv-fr.json with:

```
"rotateLeft": {

       "alt": "Rotation&#192; Gauche", "title": "Rotation&#192; Gauche"

               },
```

Now, when a user's browser is set to display the French language, "Rotate Left" will appear as "Rotation à Gauce."

Force a Specific Language

If you do not wish to use the language settings auto-detection, you can force override the UI to use a specified translation.

This setting is controlled via a setting `localizeOptions` in vvDefines.js as shown below. The vvDefines.js file is located in the /virtualviewer/js/ directory.

```
// Remove the backslashes (//) before the word language and replace
// the values zz with the letter codes of the language file you want to
// force.
//
// Have the translation file available for reference.

localizeOptions: {
   //language: "zz",
   pathPrefix: "resources/locale"
},
```

Advanced customization

This section describes how to set up and work with the advanced features in VirtualViewer® HTML5.

Virtual Documents

This section describes how to work with virtual documents.

A virtual document is a collection of any combination of documents or pages of documents displayed as a single multi-page document with a single set of thumbnails. The pages can be from documents of different file format types such as AFP, Word, or PDF. The virtual document is viewed and regarded as any normal document would be.

Please note the following:

- Exporting to a .tif may require significant resources especially if con- verting to 24-bit color.
- Virtual Documents export only to .tif because the virtual documents may be a mix of multiple formats.
- If you are viewing all pages in a single document, you should not use Vir- tual Documents.
- Document Notes is not supported in Virtual Documents.

Loading Virtual Documents

To pass a number of documents to the viewer, the value of a `documentId` can start with a special identifier, followed by a string of a comma-separated list of `documentIds`. The list is issued to create the virtual document. The `documentIds` are listed in the order in which the documents are to be compiled for viewing.

Virtual Document Syntax

The special identifier is the string `VirtualDocument`: which is then followed by any number of `documentIds`. The syntax can be used any time a normal `documentId` could be used. A `documentId` in the comma-separated list may be specified in the following manner.

File Name	Description

ABC.tif	This specifies that all pages of the document should be included.
ABC.tif[2]	This specifies that only a single page from the document should be included.
ABC.tif[1-3]	This specifies that a range of pages from the document should be included.

> **Note:** To include non-consecutive pages from a single document, you need to specify the document each time in the virtual document string.

Displaying a Virtual Document

Virtual documents are displayed based upon parameters in the URL. For example, suppose that you have three documents, `ABC.tif`, `EFG.pdf`, and `IJK.doc`, each with three pages.

You can create a virtual document from the entirety of all three documents:

```
http://localhost:8080/virtualviewer/index.html?documentId=VirtualDocument:AB
C.tif,EFG.pdf,IJK.doc
```

Based upon the parameters described above, we know that we are displaying the entire contents of the original documents in our virtual document. If we wanted to include (for example) the full texts of ABC.tif and IJK.doc but only page two of EFG.pdf, we would enter:

```
http://localhost:8080/virtualviewer/index.html?documentId=VirtualDocument:AB
C.tif,EFG.pdf[2],IJK.doc
```

Finally, if we wanted to include a range of pages from ABC.tif, the full EFG.pdf, and only page 3 from LJK.doc, we would enter:

```
http://localhost:8080/virtualviewer/index.html?
documentId=VirtualDocument:ABC.tif[1-2],EFG.pdf,LJK.doc[3]
```

Virtual Documents: Save Document As

When a user prints, exports, emails, or uses **Save Document As** with a virtual document, the resulting document will reflect what the user sees on their screen at the time of execution. Please note that `sendDocument` is not supported in virtual documents. A work around is to send a virtual document with **Save Document As**. **Save Document A**s has better functionality than `sendDocument`.

The `loadVirtualDocumentAnnotations` and `saveVirtualDocumentAnnotations` web.xml parameters enables virtual documents to read annotations from the source document and to save annotations created on the virtual documents back to the source document. The default values for both parameters are set to **`false`**.

Printing Virtual Documents

To print a virtual document, select the **Print** button. 

Special annotation layers

You can save Snowbound and FileNet annotations. See the sections below for more information on configuring Snowbound and FileNet annotations.

Support for Daeja annotation saving is in development. Please check with Snowbound for updates if you are interested in this feature.

Enabling Snowbound annotations

To save annotations in the Snowbound XML format, add the `annotationOutputFormat` parameter with the value set to Snowbound to the servlet web.xml files as shown in the example below:

```
<init-param>
   <param-name>annotationOutputFormat</param-name>
   <param-value>Snowbound</param-value>
</init-param>
```

If VirtualViewer HTML5 for Java is configured to save Snowbound annotations, then any existing annotations that are in the FileNet format are read in as read-only and are not

able to be edited or deleted. Edit controls are disabled for annotation layers that are not editable.

This means that, for example, the menu items for the layer will be visible, but grayed-out in menus such as **Select Layer**. When you right-click an annotation to edit it, the pop-up menu will simply not appear.

Snowbound annotation supported configurations

Supported server configurations

Parameter Name	Value	File Location
`annotationOutputFormat`	Snowbound	Web.xml

> **Note**: Snowbound annotations can be used with any configuration of non-required annotation parameters.

Supported client configurations

None.

Configuring FileNet annotations

To save annotations in the FileNet XML format, add the `annotationOutputFormat` parameter with the value set to FileNet to the servlet web.xml files as shown in the example below:

```
<init-param>
   <param-name>annotationOutputFormat</param-name>
   <param-value> FileNet</param-value>
</init-param>
```

You will also need to set the `oneLayerPerAnnotation` parameter the config.js file to **`true`** as shown below:

```
var oneLayerPerAnnotation = true;
```

FileNet annotation supported configurations

Supported server configurations

Parameter Name	Value	File Location
annotationOutputFormat	filenet	Web.xml

> **Note**: Snowbound annotations can be used with any configuration of non-required annotation parameters.

Supported client configurations

Parameter Name	Value	File Location
base64EncodeAnnotations	False	Config.js
oneLayerPerAnnotation	True	Config.js

Special Annotation Mapping

The table below shows the FileNet annotation and its analogous Snowbound annotation:

FileNet Annotation	Snowbound Annotation
v1-Rectangle	SANN_FILLED_RECT
Arrow	SANN_ARROW
Stamp	SANN_EDIT
Text	SANN_EDIT
Transparent Text	SANN_EDIT (Not transparent)
v1-Oval	SANN_FILLED_ELLIPSE

v1-Highlight Polygon	SANN_FILLED_POLYGON
Pen	SANN_FREEHAND
Freehand Line	SANN_FREEHAND
Highlight Rectangle	SANN_HIGHLIGHT_RECT
v1-Line	SANN_LINE
v1-Open Polygon	SANN_POLYGON
Closed Polygon	SANN_POLYGON
StickyNote	SANN_POSTIT

Annotations Security: Watermarks and Redactions

The implementation of security for annotations allows each layer to have a permission level assigned to it. This permission level is not inherent in the layer and is only defined when the layer is retrieved by the content handler.

In order to assign a permission level to an annotation layer, the content handler must be implemented or extended and the `getAnnotationProperties` method used.

The Annotation Security Model

The security model is such that when reading annotation layers, various levels of permissions for viewing and working with annotation layers may be specified.

The model currently accounts for nine levels on a per layer basis.

> **Note**: Redaction annotations are only considered redactions when they are burned in and saved as an image format file such as TIFF format.

Permission Levels

Each successive level includes the functionality of previous levels. This allow each annotation layer to carry a set of permissions. These permissions allows the layer to be passed in with several different levels of permissions such as read only or edit.

If you are storing the annotations as layers (XML files) with a redaction permission level, then you will be able to present them to the users in the viewer as burned in, but they will not actually be burned into the source document.

Permission	Level	Actions Permitted
PERM_HIDDEN	Hidden	The layer is passed to the client but not displayed.
PERM_PRINT_ WATERMARK	Print Watermark	The user does not see the layer, but it will be burned in for printing.
PERM_VIEW_ WATERMARK	View Watermark	The user may view the layer but may not hide the layer.
PERM_VIEW	View	The user may view or hide the layer.
PERM_PRINT	Print	The user may print the layer.
PERM_CREATE	Create	The user may add an object to the layer
PERM_EDIT	Edit	The user may also edit an object on the layer, and edit layer properties.
PERM_DELETE	Delete	The user may also delete an object on the layer, and delete the layer

Level Definitions

Permission	Definition
Hidden	If a layer is indicated as having the Hidden permission, the information about the layer will be passed, so that changes done by Page Manipulation will be applied when the annotations are saved. The layer is not displayed to the user even if manipulations are applied.
Print Watermark	If a layer is indicated as having the Print Watermark permission, it shall be passed as a normal layer, but will not be shown to the user. When the document is printed, any layer with Print Watermark permission will be applied to the image before printing.
View Watermark	If a layer is indicated as having the View Watermark permission, it shall be passed as a normal layer. However, the user will not be allowed to show or hide the layer, or manipulate the layer in any way. This layer will never be printed.

View	If a layer is indicated as having the View permission, it shall be passed as a normal layer. The user will be able to hide or show the layer. The user will not be able to add an object, edit an object, delete an object, print the layer, rename the layer, or delete the layer..
Print	If a layer is indicated as having the Print permission, it shall be passed as a normal layer. The user will be able to hide or show the layer, print the layer. The user will not be able to add an object, edit an object, delete an object, or rename or delete the layer.
Create	If a layer is indicated as having the Create per- mission, it shall be passed as a normal layer. The user will be able to hide or show the layer, print the layer, or add an object to the layer. The user will not be able to edit an object, delete an object, edit the layer properties, or delete the layer.
Edit	If a layer is indicated as having the Edit permission, it shall be passed as a normal layer. The user will be able to hide or show the layer, add an object, edit an object, or edit the layer properties. The user will not be able to delete objects or delete the layer.
Delete	If a layer is indicated as having the Delete per- mission, it shall be passed as a normal layer. The user will have full rights to perform any operation on the layer.

Securely retrieving annotation layers

When loading a document, annotation layers will need to be retrieved and have the correct permission level set. The process of loading an annotation layer is as follows:

For each `annotationKey` returned by `getAnnotationNames` the following method will be called.

```
public Hashtable getAnnotationProperties (clientInstanceId, documentKey,
annotationKey)
```

This method returns a hash table with the following expected key/value pairs for that annotation layer.

Key/Value Pairs

- The `permissionLevel` will determine how the layer is handled. If no value is set, an exception will occur.

- The `redactionFlag` determines if the layer has **Mark Layer As Redaction** selected in the client. If no value is set, an exception will occur.
- If the `permissionLevel` is set to `PERM_REDACTION`, the value of `redactionFlag` is moot since the client does not receive that layer as an annotation layer.
- If `getAnnotationProperties` returns null, an exception will occur. This prevents cases where a layer should have strict permissions but for some reason no permission level gets set.

Saving Redaction Layers

If a layer has **Mark Layer As Redaction** selected, when choosing **Save Annotations**, VirtualViewer HTML5 for Java will pass both the `permissionLevel` and the `redactionFlag` to the `saveAnnotationContent` method in a hash table:

```
public void saveAnnotationContent(ContentHandlerInput input)
saveAnnotationContent(ContentHandlerInput input)

(String clientInstanceId, String documentId, String annotationKey, byte []
data, Hashtable annProperties)
```

Printing Layers

When printing a document, the user may choose to print with or without annotations. Only visible layers with a Print permission level or higher in the Image Panel will print.

A layer which has been given a `permissionLevel` of **`PERM_REDACTION`** shall always print as part of the image, (since it has been burned into the image), even if the user chose to print without annotations.

DWG Layer Support

You can toggle DWG layers in and out of view in VirtualViewer. DWG layers are typically called referenced design layers. Layers include schematics or diagrams of blueprints that are embedded in the DWG file and laid over the image at view time.

The DWG option for VirtualViewer HTML5 Java requires Microsoft Visual C++ 2015 Redistributable installed on your computer.

You can either install this through Microsoft or use the vcredist_x64.exe that is included with your Snowbound license.

Follow the steps below to use the **Layer Manager**:

1. Load a **DWG** or **blueprints** file that contains layers.
2. Select the **Layer Manager** .

   The Layer Manager dialog box displays a list of all the DWG layers on the Image Layers tab. The left side of the tab displays the layer name. The right side of the tab displays a visibility button to toggle the visibility of the single layer.

3. From the Layer Manager dialog box, choose which DWG layers to take in and out of view.
4. Select the **Visibility** button to view or hide the image layer. A check on the visibility button indicates that the image layer is hidden.
5. Select **OK** to display the changes that you made in the Layer Manager.

The DWG file format is only available for 64-bit Windows systems. Cropping a DWG page with layers is not supported.

Selecting **Export**, **Print**, **Save As**, and **Email** includes all DWG layers. There is no option to choose specific layers for each function to carry out.

Page Manipulations carry over all DWG layers on that page.

Virtual Documents consisting of DWG pages allow you to take the layers in and out of view on a page.

DWG XREF Support

You can load a DWG file that contains references to other files (xrefs). Those related drawings are attached and displayed along with the DWG file. This feature assumes that the xrefs are in the same location as the original DWG file.

Set a valid directory in the `tmpDir` key in web.xml.

If the content handler returns any external reference files, they are saved in the [your temp directory]/[document ID] directory. Make sure this directory is accessible to VirtualViewer to read and write. External reference files are saved into these directories.

Use the following key in the Content Handler `result` class:

```
KEY_EXTERNAL_REFERENCE_CONTENT_ELEMENTS
```

This key returns the vector of `ExternalReference` objects defined in the `clientcontentserver` package.

To implement external references in your content handler, include references to the `ExternalReference` class in your code.

Watermark JSON Files

Watermarks for a document are stored in a json file. Like annotations, the file will be `documentkey + suffix`, e.g., "6-Pages-1.tif.watermarks.json".

The .watermarks.json file is a list of json objects, so it has the format:

```
[ { myJsonData }, { myOtherJsonData }]
```

Watermark properties

Each individual watermark is a json object. Each will have the following properties:

Property	Type	Description
`Transparency`	Boolean	If true, the watermark will be transparent; if false, it will be a solid color.
`adminCreated`	Boolean	If false, any user can manage any aspect of the watermark. If true, admin restrictions will apply (as described below).
`Text`	string	This is the text that will appear on the watermark. Multiline watermarks are supported. This is done under the hood in the watermarks dialog, but if a user is manually entering json, they

		should enter a newline character ("\n") where a line break should be.
`allPages`	Boolean	If this is set, the watermark will appear on every page of a document.
`Pages`	array of page indices, zero-indexed	For instance, to place a watermark on only page one, this property would contain [0]. This is a key difference between watermarks and annotations. Watermarks are intended to repeat across pages, so an identical watermark will have multiple pages it applies to.
`widthAtTenPx`	integer	This is a read-only value used by VirtualViewer to calculate the dimensions of the watermark, representing how wide the watermark is when the font is 10 pixels high.
`Stretch`	double	This defines how far across the page the watermark will stretch. Set to 1.0, the watermark will go across 100% of the page (minus some margin space). Set to 0.5, 50% of the page. The UI allows only a small set of percentages. Diagonal watermarks will always stretch 100% across the diagonal.
`Format`	json sub-object	A json sub-object that has font and color information as follows: ▪ font: A font name, for instance "Arial".color: ▪ We currently support only one color, so "000000" would be stored here.
`Position`	json sub-object	This is another sub-json object, that defines where the watermark will be placed on the page. There are two defining properties in here: the vertical placement of the watermark (top of the page, middle of the page, or the bottom of the page) and the direction of the text. While these options may open up further, the direction options are currently left-to-right text or diagonal text. The two options combine so that, for instance, top vertical placement & diagonal direction produce a watermark stretching from the top-left to bottom-right corner--while bottom vertical placement & diagonal direction will go from bottom-left to top-right: ▪ vertical: Use 0 for top, 1 for center, and 2 for bottom. ▪ direction: Use 0 for left-to-right text, and 2 for diagonal text.

Watermark.json file sample

```
[{"widthAtTenPx":19,"transparency":true,"adminCreated":false,"text":"bugs","
allPage s":true,"pages":[],"stretchPercent":0.5,"format":{"font":"Times New
Roman","color":"000000"},"position":{"vertical":0,"direction":0}},{"widthAtT
enPx":86,"t
ransparency":true,"adminCreated":false,"text":"second%20watermark","allPages
":fal se,"pages":[0],"stretchPercent":1,"format":{"font":"Times New
Roman","color":"000000"},"position":{"vertical":2,"direction":2}},{"widthAtT
enPx":62,"t
ransparency":false,"adminCreated":false,"text":"sdadafsadfgsafd","allPages":
false,"p ages":[0],"stretchPercent":1,"format":{"font":"Times New
Roman","color":"000000"},"position":{"vertical":2,"direction":0}}]
```

Tips and troubleshooting

This sections describes tips and troubleshooting solutions to resolve issues that some users have experienced with VirtualViewer® HTML5.

Tips

Changing the Default Directory

VirtualViewer HTML5 for Java is delivered with virtualviewer/ as the default directory.

If you would like to change this directory, you need to change the pointers in the files below to reflect that change.

For example, let's suppose we want to change the directory from virtualviewer/ to a directory called displaytools/.

First, in **config.js**, we would change:

```
servletPath: "/virtualviewer/AjaxServlet"
```

to:

```
servletPath: "/displaytools/AjaxServlet"
```

In **web.xml** (located in the WEB-INF sub-directory), we would change:

```
<init-param>
   <param-name>codebase</param-name>
   <param-value>http://localhost:8080/virtualviewer</param-value>
</init-param>
```

to:

```
<init-param>
   <param-name>codebase</param-name>
   <param-value>http://localhost:8080/displaytools</param-value>
</init-param>
```

Changing the default directory for image rubber stamps

In web.xml, the default image rubber stamp parameters need to have their path changed as shown in the example below, or else the viewer will spin and not load any documents.

To take the "Approved" default stamp as an example, you would need to change the parameter as follows:

```
<init-param>
<param-name>Approved</param-name>
   <param-value>
         Approved,535,293,http://localhost:8080/virtualviewer/resources/sta
         mps/Approved.png
   </param-value>
</init-param>
```

to:

```
<init-param>
<param-name>Approved</param-name>
   <param-value>
         Approved,535,293,http://localhost:8080/[new/path/to/your/resource/
         folder]/Approved.png
   </param-value>
</init-param>
```

Documents Slow to Load in Multiple Documents Mode

Performance may be affected, and documents may take several minutes to load, if the `multipleDocMode` parameter is set to `availableDocuments` and the directory specified in the `filePath` configuration parameter has several hundred files (The default `filePath` value is `=".\sample-documents"`).

To avoid this issue, set the `multipleDocMode` parameter to `specifiedDocuments`. The default setting for the `multipleDocMode` parameter is now `specifiedDocuments`.

Improving Performance or Quality

One of the differences between raster and vector formats is that raster formats have specific DPI (dots per inch) and bit depths. Vector formats aren't inherently black and white or color, and while they typically have sizing in inches, there is nothing that says what DPI or bit depth to use when rendered as a raster image.

When the content server pulls out a page from a vector format document, it must render that page to a certain DPI and bit depth, as well as save that image as some format to be passed to the client for display. (Sending an SVG version of the document for display retain the vector format and DPI is not necessary.) The particular settings are determined on a per format basis by three servlet parameters.

To improve the performance, you can save your files as black and white or grayscale. For example, if you are converting a PDF document, you can save the document in the TIFF_G4_FAX file format. This will make the file size smaller and improve performance.

Please note that there is always a trade off between performance and quality. To improve performance, the quality of the image may be less. This is true whenever working with any imaging software. Please note that depending on the operating system and configuration, certain unusual or corrupt documents or files can cause the software to crash. Potentially, in some unusual circumstances, files may not be rendered identically to the creator application and may not format correctly or miss information.

Setting the Bit Depth (`xxxBitDepth`)

This parameter determines what bit depth to use when converting the vector page. Valid settings for this format are 1 (for black & white, smaller) or 24 (for color, bigger). If any pages of the vector document might be in color, then the setting of 24 should be used, since there is no way to tell if a page might or might not contain color vector objects.

The example below shows how to set the bit depth parameters in the web.xml file. For a list of web.xml parameters, please see Servlet Tags for web.xml.

```
<init-param>
   <param-name>docxBitDepth</param-name>
   <param-value>24</param-value>
</init-param>
```

The available bit depth parameters are shown in the table below:

Parameter Name	Description
bitDepth	The default bits per pixel for decompression of ormats not specified with individual parameters.
docxBitDepth	The bit depth to use for Word 2007 documents. Valid values are 1 or 24.
iocaBitDepth	The bit depth to use when decompressing IOCA pages. Valid values are 1 or 24.
modcaBitDepth	The bit depth to use when decompressing MO:DCA pages. Valid values are 1 or 24.
pclBitDepth	The bit depth to use when decompressing PCL pages. Valid values are 1 or 24.
pdfBitDepth	The bit depth to use when decompressing PDF pages. Valid values are 1 or 24.
pptBitDepth	The bit depth to use when decompressing PPT pages. Valid values are 1 or 24.
wordBitDepth	The bit depth to use when decompressing Word pages. Valid values are 1 or 24.
xlsBitDepth	The bit depth to use when decompressing XLS pages. Valid values are 1 or 24.

Setting the DPI (xxxDPI)

This parameter determines how many DPI (dots per inch) should be used when converting a vector page. Typical settings for this parameter are 150, 200, or 300. The higher the DPI, the higher the quality of the image, but also the bigger the size, which means more processing on the server and larger page sizes across the network.

The optimal setting for this varies by format, but 200 is usually good for black & white documents or text, and 300 for color images and more detailed documents. Even higher numbers can be used (400, 600) but it can seriously affect speed of processing and available resources.

The example below shows how to set the DPI parameters in the web.xml file. For a list of of web.xml parameters, please see Servlet Tags for web.xml.

```
<init-param>
   <param-name>docxDPI</param-name>
   <param-value>200</param-value>
</init-param>
```

The available DPI parameters are shown in the table below:

Parameter Name	Description
docxDPI	The Dots Per Inch to use for Word 2007 documents.
iocaDIP	The Dots Per Inch to use when decompressing IOCA pages.
modcaDPI	The Dots Per Inch to use when decompressing MO:DCA pages
pclDPI	The Dots Per Inch to use when decompressing PCL
pdfDPI	The Dots Per Inch to use when decompressing PDF pages.
pptDPI	The Dots Per Inch to use when decompressing PPT pages.
wordDOI	The Dots Per Inch to use when decompressing Word pages.
xlsDPI	The Dots Per Inch to use when decompressing XLS pages.

Setting the Format (xxxFormat)

This parameter determines which format the vector page will be rendered to for sending to the client. Valid values for this parameter are TIFF_G4_FAX (black & white, best for text documents, small size), JPEG (color, good for images, lesser quality for text, small size), TIFF_LZW (color or greyscale, good for documents with text and color elements), or PNG (color, better for text than JPEG, not as small).

By adjusting these parameters in various combinations, you can find the best settings for your environment, documents, and user load.

The example below shows how to set the format parameters in the web.xml file. For a list of web.xml parameters, please see Servlet Tags for web.xml.

```
<init-param>
   <param-name>docxFormat</param-name>
   <param-value>TIFF_LZW</param-value>
</init-param>
```

The available format parameters are shown in the table below:

Parameter Name	Description
docxFormat	The format to convert Word 2007 documents to. Valid values should are TIFF_G4, JPEG, TIFF_ LZW, PNG.
icoaFormat	The format to convert IOCA pages to. Valid values are TIFF_G4_FAX, JPEG, TIFF_LZW, PNG.
modcaFormat	The format to convert MO:DCA pages to. Valid val- ues are TIFF_G4_FAX, JPEG, TIFF_LZW, PNG.
pclFormat	The format to convert PCL pages to. Valid values are TIFF_G4_FAX, JPEG, TIFF_LZW, PNG
pdfFormat	The format to convert PDF pages to. Valid values are TIFF_G4_FAX, JPEG, TIFF_LZW, PNG.
pptFormat	The format to convert PPT pages to. Valid values are TIFF_G4_FAX, JPEG, TIFF_LZW, PNG
wordFormat	The format to convert Word pages to. Valid values are TIFF_G4_FAX, JPEG, TIFF_LZW, PNG. The bit depth to use when decompressing XLS pages. Valid values are 1 or 24.
xlsFormat	The format to convert XLS pages to. Valid values are TIFF_G4_FAX, JPEG, TIFF_LZW, PNG.

The full list of format server parameters and their usage is in Servlet Tags for web.xml.

Setting Office 2007 - 2010 Documents to Display Color Output

To display color output in Office 2007 - 2010 documents, set the `xlsxBitDepth` and `docxBitDepth` parameters to 24 and the `xlsxDPI` and `docxDPI` parameters to 200 as shown in the following example:

```
<init-param>
      <param-name>xlsxDPI</param-name>
      <param-value>200</param-value>
</init-param>
<init-param>
      <param-name>docxBitDepth</param-name>
      <param-value>24</param-value>
</init-param>
<init-param>
      <param-name>docxDPI</param-name>
      <param-value>200</param-value>
</init-param>
<init-param>
      <param-name>xlsxBitDepth</param-name>
      <param-value>24</param-value>
</init-param>
<init-param>
     <param-name>xlsxDPI</param-name>
     <param-value>200</param-value>
</init-param>
<init-param>
     <param-name>docxBitDepth</param-name>
     <param-value>24</param-value>
</init-param>
<init-param>
     <param-name>docxDPI</param-name>
     <param-value>200</param-value>
</init-param>
```

Note: If you experience errors processing Office 2007-2010 documents, please ensure that Aspose.Words.<jdk>.jar, Aspose.Cells.jar and dom4j-1.6.1.jar are on the CLASSPATH. This issue should not typically appear however in VirtualViewer versions 5.0 and later.

Troubleshooting

Submitting a Support Issue

You may encounter an issue that is not covered by the documentation. Snowbound technical support is standing by to help you succeed.

In order to get a fast, helpful response, please make sure Snowbound has everything needed to reproduce the issue, including:

- The configuration files: index.html, config.js, output.properties and
- .\WEB-INF\web.xml.
- The document that the user is trying to view. Most issues are document specific.
- The Java console log and the server log.
- A list of steps that the customer took from starting the Viewer until they see the error.
- It is often helpful to have screen shot of what the user is doing when they encounter the error as well.
- The version of VirtualViewer and Java that are being used.

"Please wait while your image is loaded" Message Displays Indefinitely

In some cases, images do not load in the VirtualViewer HTML5 for Java client, and the "Please wait while your image is loaded" message displays indefinitely in the browser. This generally happens when:

1. The web server is not properly configured to handle the necessary http requests made by the client
2. The VirtualViewer server configuration itself is incorrect.

To resolve this issue, you should log the http traffic between the client and the server in order to determine which http requests are failing and why. This can be done using a browser plugin such as httpWatch (http://www.httpwatch.com) or Firebug (http://getfirebug.com). You can also use a standalone application such as Fiddler (http://www.fiddler2.com) or Wireshark (http://www.wireshark.org) which can be run independently on the client machine. For Internet Explorer 9 users, the traffic can be

captured using the IE Developer Toolbar (http://www.microsoft.com/download/en/details.aspx?id=18359).

Once the http traffic has been captured, you should be able to see which requests are failing. Typically, a failed request will cause a 400 or 500 error code to be generated in the logs. Some common error codes that can occur for VirtualViewer HTML5 for Java are as follows:

404 Not Found

This error code indicates that the requested resource on the server could not be found. This error can occur if the servlet mapping is incorrectly configured on the server. First, make sure the `servletPath` parameter value in config.js contains the correct URL mapping to the AJAX servlet. If you changed the default directory name for VirtualViewer HTML5 for Java on the server, you will need to update this value to be consistent with that change. For more information on defining the `servletPath` parameter, please see Defining the Servlet Paths.

For VirtualViewer HTML5 for Java, the web.xml configuration should also be reviewed in addition to config.js. Make sure that the values for `<servlet-class>` and `<url-pattern>` are correct for the relative `<servlet-name>`. Please note that by default, the servlet name is set to `AjaxServlet`.

405 Method Not Allowed

This error code indicates that the http request contains an action (e.g. POST, GET, HEAD, etc.) that is not allowed by the requested IIS server module.

With respect to VirtualViewer HTML5 for Java .NET, this typically means that the IIS handlers for AJAXServer and aspnet_isapi.dll have not been properly configured in IIS. First, make sure web.config contains the following handler mapping for AJAXServer:

```
<httpHandlers>
<add verb="*" path="virtualviewer" type-
e="Snowbound.virtualviewer.AjaxServerHandler, Snow- bound.virtualviewer" />
</httpHandlers>
```

Then, make sure that a wildcard mapping for aspnet_isapi.dll has been created for your website configuration. This DLL is a required resource for VirtualViewer and is usually

located in Windows under C:\Windows\Microsoft.NET\Framework\v2.0.50727\. To add aspnet_isapi.dll to your IIS configuration, please review the instructions below:

For IIS5:

1. Go to **<VV web application> Properties > Directory (tab) > Configuration > "Add"**.
2. For the "Executable" setting, provide the path to aspnet_isapi.dll.

3. Set the "Extension" setting to **".*"** and left click inside the "Executable" path field to enable the **OK** button below (this is a bug in IIS5; see http://support.microsoft.com/kb/317948 for more).

ForIIS6:

1. Go to **<VV web application> Properties > Virtual Directory (tab) > Configuration > "Insert Wildcard application map"**, and provide the path to aspnet_isapi.dll.

ForIIS7:

1. Go to **<VV web application> Handler Mappings > Actions > "Add Wildcard Script Mapping"** and provide the path to aspnet_isapi.dll.

500 Internal Server Error

This error may occur if the content handler mapping is not correctly set in the web configuration.

For VirtualViewer HTML5 for Java, check the `contentHandlerClass` parameter value.

For VirtualViewer HTML5 for Java .NET, check the `contentHandler` key value. Make sure this value contains the correct path to the content handler.

Annotation Text Does Not Appear on Separate Lines

An issue may occur where annotation text does not appear on separate lines. This occurs because Linux has different line-end characters than Windows. Linux uses just a line feed while Windows uses a carriage return + line feed (CRLF).

To solve this issue, add the following line in your code so that line-end characters will be the same on all systems:

```
System.setProperty("line.separator","\r\n")
```

Unable to Enter More Text After Using the "-" Key in an Annotation

An issue may occur where you cannot enter any more text after entering the "-" key in an annotation. This was caused by the keyboard shortcut for zoom out being defined without the CTRL modifier.

This issue will be resolved in the next release by changing the shortcuts for zooming. For zoom in, select CTRL+. For zoom out, select CTRL-.

Getting an Evaluation Period Expired Error Message When Creating a War File

An issue may occur where you receive an "Evaluation Period Expired" error message when creating a war file.

To solve this issue, look for the `servletURL` parameter in your html file. If you are using that parameter and it is pointing to an evaluation version of the servlet (possible on another machine), you will get the error messages.

Fonts Do Not Display Correctly

An issue may occur where the font displays incorrectly in the following way:

- The text in the output document is not in the right font.
- The text in the output document does not display the same way on Windows and on Linux.

To solve this issue, follow the steps below:

2. Inspect the document to determine what fonts it requires.
3. Make sure those fonts are installed on the system. The fonts are usually installed in the font.properties file.
4. Make sure the fonts are registered with Java and are of a type supported by your version of Java.

There are several resources on the Internet that discuss how to do this. There are also some helpful tools such as font viewers that make this easier. Some resources we like are:

- Java Font resources: http://mindprod.com/jgloss/font.html

- Windows Font knowledgebase article: http://support.microsoft.com/kb/918791
- Java Font.Properties description from Sun: http://java.sun.com/j2se/1.4.2/docs/guide/intl/fontprop.html
- Linux Font installation: http://linuxandfriends.com/2009/07/20/how-to-install-fonts-in-linux-ubuntudebian
- Linux Font configuration man page: http://linux.die.net/man/5/fonts-conf

Excel 2007 XLSX files return `-7 Format_not_found error`

Note: This issue should only be found in old versions of product.

To render Word 2007, Excel 2007 and PowerPoint 2007 documents, VirtualViewer HTML5 for Java may rely on third party packages. In order to properly integrate these packages, the CLASSPATH may have to be modified.

You can specify additions to the CLASSPATH using the web.xml parameter `classPathAddition` according to the following example:

```
<init-param>
<param-name>classPathAddition</param-name>
<param-value>c:\sample\sample-cells-7.4.3.jar;c:\sample\dom4j-1.6.1.-
jar;c:\sample\sample.slides-7.3.0.jar;c:\sample\log4j-1.2.16.jar;
</param-value>
</init-param>
```

XLS or XLSX Page Content Truncated

Your XLS or XLSX page content may be truncated because XLS and HTML-formats do not include the page size in the document like Word and PDF. It can be set explicitly similar to how you can set the page size when printing. To set the page size to avoid truncated content, use the `xlsHeight`, `xlsWidth`, `xlsxHeight`, and `xlsxWidth` parameters in the web.xml file as shown in the examples below. .

For XLS files set the parameters as shown in the example below to the height and width that you would like for your document:

```
<init-param>
<param-name>xlsHeight</param-name>
<param-value>11</param-value>
</init-param>
<init-param>
<param-name>xlsWidth</param-name>
<param-value>14</param-value>
</init-param>
```

For XLSX files, set the parameters as shown in the example below to the height and width that you would like for your document:

```
<init-param>
<param-name>xlsxHeight</param-name>
<param-value>11</param-value>
</init-param>
<init-param>
<param-name>xlsxWidth</param-name>
<param-value>14</param-value>
</init-param>
```

Overlay Resources Not Pulled into APF or MODCA Document

If overlay resources such as signatures are not being pulled into an AFP or MODCA document, then make sure that the resource filename does not have a filename extension. If the resource filename has a filename extension, remove it.

Documents Loading Slowly in Multiple Documents Mode

Performance may be affected, and documents may take several minutes to load, if the `multipleDocMode` parameter is set to `availableDocuments` and the directory specified in the `filePath` configuration parameter (The default value is =".\sample-documents".) has several hundred files. To avoid this issue, set the `multipleDocMode` parameter to `specifiedDocuments`.

The default setting for the `multipleDocMode` parameter is now `specifiedDocuments`.

Images Disappear in Internet Explorer 9 when Zooming or Rotating

> **Note**: For this reason and many others, Snowbound DOES NOT RECOMMEND any Internet Explorer older than Microsoft Edge.

An Internet Explorer 9 canvas bug can cause images to disappear when you click to zoom or rotate the image.

The issue occurs because VirtualViewer AJAX is drawing faster than IE 9 can handle.

To correct this issue, we added a variable in vvDefines.js called `ie9DrawDelay`. It inserts a delay in milliseconds into the IE 9 drawing code which can help work around this bug.

Please add this entry to the vvDefines.js file:

```
    ie9DrawDelay: 900,
```

The vvDefines.js file is located in ./virtualviewer/js/ vvDefines.js.

The default value is 100 (100 milliseconds). The user can set the `ie9DrawDelay` variable as high as necessary. However; if it is set too high, it could cause a delay for the user each time they zoom.

The `ie9DrawDelay` variable will not work if IE 9 is set to compatibility mode. It will only work in IE 9 standard mode. Please try adding

```
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
```

to index.html to force IE 9 standard mode.

Internet Explorer Limits URLs to 2048 Characters

> **Note**: For this reason and many others, Snowbound DOES NOT RECOMMEND any Internet Explorer older than Microsoft Edge.

Customers that included a lot of parameters on their VirtualViewer command line ran out of room because the viewer infrastructure was using an HTTP GET operation. The GET URL length was limited by the Internet Explorer browser. Other browsers like Chrome and Firefox allow a much longer URL.

Please see the following work around to resolve the case where the user enters a URL longer than 2048 characters into the URL bar:

For long `DocumentIDs/cIIDs`, call `init()` instead of calling `initViaURL()`.

You can call `init()`, then call `setDocumentId()` and `setClientInstanceId()`, followed by `openInTab()`.

Please see the following example to create a Javascript function:

Instead of calling `initViaURL` in index.html, call `init` such that:

```
<body onload="virtualViewer.initViaURL()">
```

becomes:

```
<body onload="virtualViewer.init()">
```

Then, initialize the viewer as needed:

```
< VirtualViewer.setClientInstanceId("whatever their clientInstanceId is, if
any");
VirtualViewer.openInTab ("whatever their document id is");
```

This solution works for all supported browsers.

Internet Explorer Defaults to Compatibility View

> **Note**: For this reason and many others, Snowbound DOES NOT RECOMMEND any Internet Explorer older than Microsoft Edge.

In Internet Explorer, users may experience an issue where their browser defaults to IE7 compatibility view when they open VirtualViewer. Since VirtualViewer no longer supports IE7, VirtualViewer will not work in this mode.

Users can resolve this issue by going into the Compatibility View Settings and unchecking the box next to **Display intranet sites** in Compatibility View.

Allowing Relative Paths to Work with Tomcat 8

Please note that in Tomcat 8.0, documents will load using absolute path, and not a relative path.

The relative path is resolved to an absolute in `FileContentHandler.setFilePath()` via a call to `context.getRealPath(relativePathName)`.

The `relativePathName` should start with a /.

Please see the following code sample to allow relative paths to work on Tomcat 8 (also, see FileContentHandler in the Javadocs directory of your product).

```
public static void setFilePath(String pathParam, ServletContext context)
{
if ((pathParam.startsWith("./") || pathParam.startsWith(".\\"))
&& context != null)
{
pathParam = pathParam.replace("./", "/"); pathParam =
pathParam.replace(".\\\\", "/");
gFilePath = context.getRealPath(pathParam)
+ File.separator;
}
else
{
gFilePath = pathParam;
}
Logger.getInstance().log(Logger.INFO,
"File path for documents is configured to "+ gFilePath);
}
```

Displaying a Document as Landscape

If the text input document is displayed as portrait and you would like to display it as landscape, set the `ascii.attribute` parameter.

Getting a `ClassNotFoundException` Error

A return status of -38 EXCEPTION_ERROR indicates an exception was thrown. This usually includes a stack trace with information about what caused the exception.

If the stack trace includes `java.lang.NoClassDefFoundError`: this indicates the issue is a missing .jar file. The name of the class following `java.lang.NoClassDefFoundError`: can be used to determine which .jar file cannot be found.

Check your java CLASSPATH carefully to ensure the directory containing the .jar is in the path. Please feel free to contact Snowbound Support at http://support.snowbound.com for help determining which .jar is missing.