VirtualViewer® 4.12 (and older) New Features and Corrected Issues

July 25, 2018

VirtualViewer 4.12 (and older) New Features and Corrected Issues

Important Phone Numbers and Links

For the most current information, please contact Snowbound Sales at

Telephone: 1-617-607-2010 or

http://register.snowbound.com/MQL-contactUs-Website-2017.html or

questions@snowbound.com or
https://mylivechat.com/chatnoscript.aspx?HCCID=17729140 (sales inquiries only)

Filing a Support Ticket

https://snowboundsupport.force.com/SupportPortal/CommunityLogin?

Release Notes and Product Manuals

http://www.snowbound.com/support/manuals

VirtualViewer 4.12 (and older) New Features and Corrected Issues

Important Information

- See **releasenote.md** in your product shipment for the very latest Release Notes
- 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
- Given the accelerated update release schedule by Oracle, we strongly recommend that customers upgrade to the latest JRE (currently Java 8) as quickly as possible. Newer JRE's generally support older Java applications without any modification. However features introduced in newer JRE's will not work for older JRE's.

VirtualViewer v4.12 Release Notes

New Features

Video

VirtualViewer can now load and play videos, in formats supported by HTML5-compatible browsers. There is no editing or annotation support at this time - videos can only be viewed and downloaded. Video format support will depend on the capabilities of the web browser.

Supported formats

VirtualViewer uses the browser's HTML5 video player to display video, and can play all types of video supported by a browser's player. Most browsers support MP4, WebM and Ogg Vorbis. This browser compatibility chart has more details: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats

User preferences

There are several new configuration options for displaying video. These options can be viewed and modified in User Preferences:

- Video Autoplay: If enabled, the video will play as soon as it is opened. If disabled, the user must click play in order to start the video.
- Mute: If enabled, the video will start muted and the user must click to unmute. If disabled, the video will start at full volume.
- Video Controls: If enabled, the video will have the controls appear on the bottom of the player to control playback, fullscreen, volume, and the ability to download the video. Available options may change depending on the browser you are using. If disabled, video controls will be hidden from view and only accessible through the right-click menu.
- Video Stretching: If enabled, the video will stretch beyond its original size to fill the viewer. It will keep its aspect ratio, which means the video will not distort as it stretches. It acts like the Fit to Window zoom option, so it will fit to height or width and may not actually fill the viewport. If

disabled, the video will not expand beyond its original size and will center in VirtualViewer's main display area.

All of these options have equivalent config.js configuration options as defaults.

Supported features with video

We currently support the following actions with video:

- Opening and viewing video.
- Downloading the video. This cannot be disabled.
- Change video viewing size (original size, fit to window and fullscreen).

The following are some things that are limitations of the HTML5 Video Player:

- Some browsers may not be able to seek, they just play from start to finish. Firefox had no issues seeking, Chrome did.
- Fast Forward and Rewind aren't baked into HTML5 Video Player.

Add configuration to auto-resize only sticky notes

When the configuration parameter `vvConfig.autoResizeTextAnnotations` is set to true, the viewer automatically resizes text annotations to fit the annotation text. If `vvConfig.autoConfirmTextAnnotations` is also set to true, the text annotation will change its size as the user types.

Now, there is a new configuration parameter `vvConfig.autoResizeStickyNoteAnnotations`. This fine-tunes the configuration control. `vvConfig.autoResizeTextAnnotations` will now only affect text annotations, and `vvConfig.autoResizeStickyNoteAnnotations` will only affect sticky note annotations.

Set document display name API

A new Javascript API `virtualViewer.setDisplayName(newDisplayName, documentID)` will set the document specified by the given document ID to the new display name. The display name will update on the document tab and document thumbnail.

Highlight annotation button currently in use

When the user clicks a button to draw an annotation, that button will stay highlighted until the user completes the annotation.

Redaction navigation

Enhanced redaction navigation has been added to VirtualViewer. When the configuration for `vvConfig.showAnnNavToggle` is set to true, the previous navigation buttons are displayed as well as a set of radio buttons. The radio buttons are for the two navigation modes and display the annotation and redaction counts for the document. When the annotation navigation mdoe is selected, navigation occurs as it did in the past. When redaction navigation is selected, it goes from redaction to redaction throughout the whole document. The redactions are selected and are focused on. The filter pages button does not work in redaction navigation mode and is disabled.

Page manipulation with bookmarks

VirtualViewer now retains bookmarks created on pages of a document as they are subject to page manipulations. The bookmarks would remain with the page in both reordering pages, cut/paste to other documents and cut/copy to new document.

Cache-seeding support (Java)

There is a new client side API that allows the server to pre-cache images for future use. By calling virtualViewer.seedCache(documentId, pages, clientInstanceId), the user can get pages ready on the server, allowing for quicker retrieval.

Fixes and changes

Stricter URL encoding requirements in Java

Recent versions of Java enforce stricter URL validation. Now, URLs entered directly into the browser's address bar must properly encode all URI components. Unencoded URLs that may have succeeded in the past may now fail.

For instance, Virtual Documents in VirtualViewer may be opened directly from the address bar, by entering a Virtual Document ID in the documentID field: `VirtualDocument:documentName.pdf[1-3]`. This request uses square brackets to specify a page range. The square brackets will cause the request to fail on newer versions of Java. To work, the square brackets must be URI-encoded as `%5B` for the opening bracket `[`, and `%5D` for the closing bracket `]`. The Virtual Document ID, properly encoded, will work: `VirtualDocument:documentName.pdf%5B1-3%5D`. Square brackets and other invalid characters must be URI-encoded.

VirtualViewer API for launching documents (`virtualViewer.openInTab(documentID)`, for example) will automatically encode the document ID as needed. Manual encoding only needs to occur when typing a VirtualViewer URL directly into the browser's address bar.

officeLicensePath parameter has been replaced by ODFLicensePath

The `officeLicensePath` parameter is only needed for working with OpenOffice OpenDocument Format files (ODF). You will only need to use the `ODFLicensePath` parameter to provide the path to this license if you've licensed the OpenOffice format.

VirtualViewer initialization API are easier to work with

The API function beforeVirtualViewerInit is called before the VirtualViewer object is initialized, and the API function afterVirtualViewerInit is called after; these functions are intended to be defined by customers to easily run code as VirtualViewer starts up. Previously customers would have to define these API functions after index.html was loaded; now, they can be defined any time.

Preserve document scroll when zooming

Previously, zooming in and out would cause the document to jump to the top of the current page. Now, older functionality is restored, and the document will stay in the same scroll position while zooming in and out.

Preserve document scroll and zoom when switching between tabs

When switching between open document tabs in the viewer, documents will now stay at the current zoom and scroll position that the user set, rather than snapping to the default zoom and going to the top of the current page. If the configuration parameter fitLastBetweenDocuments is set to true, the document will still apply the zoom of the current document to the next document opened.

disableUploadDocument parameter moved to server

The config.js parameter to disable the Upload Document functionality has been replaced with a server setting, disableUploadDocument. The config.js parameter disableUploadDoc is no longer used.

When the server setting disableUploadDocument is set to true, the service endpoint for upload will be completely disabled, so that clever users can no longer bypass the UI to "force" a document into the system. In prior releases, we specified that the content handler should handle the filtering of uploaded documents. Filtering should still be performed by the content handler, but there is now a way to completely disable upload document.

Removed sample content handlers from VirtualViewer distributable

Some sample content handlers were removed from compiled VirtualViewer code. They will no longer be accessible. Any of the following sample content handlers in use should be replaced by a customized version of the default content handler.

- InputStreamContentHandler
- FileAndURLRetriever
- MergedImageContentHandler
- MultiContentElementContentHandler

Public print API

The parameters for the printDocument() method have changed. The new parameters are as follows:

- {String} The documentId to print, must be of a document open in the viewer
- {Boolean} If true, a PDF will be exported to a file. The user will be present with a save dialog
- {Boolean} If true, the annotations will be printed.
- {Boolean} Whether or not to burn in redactions.
- {Boolean} Whether or not to include redaction tags (only used when includeRedactions is true).
- {Boolean} Whether or not to include watermarks.
- {Boolean} Whether or not to include document notes.
- {String} Either "all", "complex" or "current".
- {String} A range of pages numbers to export (only used for "complex" pageRangeType).

Alternative License Loading (Java)

Some customers, who's servlet containers were not expanding the WAR file, but instead operating on it directly (like WebSphere) were seeing an issue using the `env-entry` to specify the VV license file. We added an alternative mechanism to specify the license file using an `init-param`:

```
<init-param>
  <param-name>snowboundLicensePath</param-name>
```

```
    <param-value>./WEB-INF/lib/SnowboundLicense.jar</param-value>
</init-param>
```

If you use this `init-param`, you will need to completely remove the `env-entry` from your web.xml.

Note: Features which depend on JNI, namely DWG and OCR support, are incompatible with the `init-param` license loading. If you need these features you will need to use the `env-entry` mechanism

Misc. Fixes/Changes:

- Improved the responsiveness of the toolbars
- Added a close button to the split screen/document compare panel
- Fixed an issue with the autoLayerPrefix when there were existing autolayers
- Previously, if you created an annotation layer in the Layer Manager dialog and clicked the dialog box's OK button instead of the layer UI's OK button, the layer would not be created
- Fixed issue with annotation layers and page manipulations
- Fixed several minor/cosmetic issues with search UI
- Load the document model asynchronously, to improve responsiveness
- Make sure the document model has been loaded before requesting an image from the server
- Fixed tab naming after closing document compare panel
- Fixed issue with watermarks and reordering pages
- Confirm user wants to save when closing the browser window/tab
- Fixed an issue with image buffering
- Fixed issue with page-change callbacks where they were firing even if the page didn't actually change (like calling `firstPage()` when you were already on page one).

New Features 4.11

Document Compare

Summary

This feature will take the text of two documents open in the viewer and compare them together. The results of this comparison are displayed in a new tab in the right-hand thumbnail pane, and users can navigate through each edit to the document.

Entering the workflow

To start comparing documents, first open a document. This document will be the "original" or "old" document in the comparison. Navigate to the document tab in the right-hand thumbnail pane, and right-click the document thumbnail. Select the option "Document Comparison" in the menu that appears.

This option will open your second document in a new pane of the viewer, splitting the view in half. It will also start the process of document comparison. In the right sidebar, a list of pages will appear. Clicking on one of those pages will reveal a list of differences between the documents.

Types of changes

Consider a document that has the first line "To be or not to be," open in the left-hand pane as the original document. Choosing the second document, the user opens a document with the first line "To think or not to think." Document comparison interprets that "be" has been removed and "think" has been added. If this were reversed, and the "to think or not to think" document were open on the left, document compare would declare that "think" has been removed and "be" has been added.

Navigating document compare

Changes in the sidebar are displayed page by page. Click on the "Page 1" text to see all the changes on page 1; click again to collapse those changes away. Clicking on the text of a change in the sidebar will scroll the document to where the text is actually located on the page.

If a page has no changes, it will display "There are no changes to display." If page text has not loaded yet, a loading gif will display. There will also be a "Load More" button at the bottom of the sidebar. If

pages have not been compared, it means their text has not been loaded; either clicking the Load More button or scrolling through the documents will load more page text and thus more document comparison. Page loading and document compare is incremental to preserve performance on long documents.

At the top of the document compare tab in the sidebar, there are three small buttons for controlling document compare. The first, with an icon like an eye, will toggle whether the red and green highlighting on the document is visible. The second, with an icon like a padlock, will lock document scrolling. Scrolling up or down in one document will scroll the other visible document just as much. Finally, the refresh button will re-run and re-display document compare.

Exiting the workflow

Right-clicking on the right-hand document will bring up a menu with the option "Close Document Comparison." This will close the second pane, returning to the full-screen view. The document comparison tab in the right-hand thumbnail pane will also be hidden.

Using OCR with document comparison

If OCR is set up, it is possible to compare documents using OCR. This will occur automatically. If document comparison is initiated and at least one of the documents contains no text, the user may request that OCR is performed. The resulting text will be compared. If OCR is not configured, it will not be attempted, and documents without text simply cannot be compared.

API

virtualViewer.compareDocuments(firstDocumentID, secondDocumentID, { splitScreenDirection })

Parameters:

- firstDocumentID {String} This document will be opened in the left pane, as the "original" document.
- secondDocumentID {String} This document will be opened in the right pane, as the "new" or "revised" document.
- options {Object} This options object does not need to be passed in, and currently contains one optional parameter; it may be modified later to contain more. To pass data in through the options object, create an object with a key-value pair: virtualViewer.compareDocuments("myOldDocument.pdf", "myNewDocument.pdf", { splitScreenDirection: "vertical" });

- splitScreenDirection {String, "horizontal" or "vertical"} The default value is "horizontal". The value "horizontal" will display document panes side-by-side. Pass in the value "vertical" to display the document panes one on top of the other.

Returns: undefined

The API compareDocuments will open both of the provided document IDs in the viewer. The first document ID will be the "original," left-hand document, while the second will be the right-hand document. The final parameter is an options object, which can pass in optional parameters. There is one optional parameter, splitScreenDirection.

virtualViewer.getDocumentCompareReport()

Parameters: onReportCompleteCallback

Returns: A string containing the document comparison data, if no callback is provided as a parameter.

The API getDocumentCompareReport will provide an HTML report of all the changes between the two open documents in a unified diff. The report will be provided as an argument to the onReportCompleteCallback given as a parameter, or returned directly from the function getDocumentCompareReport if no callback is provided. Providing a callback is recommended, since then document compare will be able to finish loading and processing asynchronously.

The return string is a element; each changed item is a within that . This includes unchanged text.

- Spans containing unchanged text will have the class name documentCompareReportNoChange
- Spans containing text removed from the original will have the class name documentCompareReportTextRemoved
- Spans containing added text will have the class name documentCompareReportTextAdded

This report does not contain styling.

Callback Event Manager

Starting with 4.11, there is a single API method for setting callbacks and a new manager for the callbacks.

API

virtualViewer.setCallback('callbackString', callbackFunction{})

Parameters:

- *callbackString*: The following are the callback names. You pass in the string and it will properly set that callback.
- *callbackFunction* Pass in the function that you want to be called. It should accept one Arguments object that has the values listed next to the callback below.
- *onDocumentLoad* documentId
- *saveDocument* documentId, clientInstanceId, documentIdToReload
- *saveAnnotation* documentId, clientInstanceId, documentIdToReload
- *saveAsDocument* oldDocumentId, newDocumentId, clientInstanceId
- *uploadDocument* uploadedDocumentId, clientInstanceId
- *sendDocument*
- *pageChange* page
- *pageCopied* pages
- *pagePasted* pages
- *pageDeleted* pages
- *rotation* page, degrees
- *textSelect* text
- *imageLoadFinished*
- *imageLoadRequested*
- *annotationCreationCallback* type

Sticky Note Background Color Support

You can now set the background color of your sticky notes. You can set the default preference in the config.js, a personal user preference in the UserPreferences menu, and individual background colors for stickies in their annotation properties menu.

User Preference Option for Default Thumbnail Tab

Now in the user preference General Preferences tab there is a section that lets you select which tab of the thumbnail panel is the default upon opening the viewer. This should help users get to the navigation they need faster.

Copy Annotations Across Documents

We expanded the copy/paste annotation functionality to work across documents in the viewer. This is only implemented for the current session, you can't copy and paste an annotation between different windows of the viewer. This is strictly for switching between tabs in the viewer.

Get Page Dimensions

You can make an API call to return the actual dimensions of an image.

API

virtualViewer.getOriginalPageDimensions()

Parameters: None

Returns: An object that contains the height and width of the image. E.g. *{height: 595, width: 679}*

Require.js

We now use Require.js to compile and load our javascript code. Each of our javascript files is now an AMD module, and VirtualViewer javascript code will be delivered to the the browser in concatenated files to reduce latency from loading multiple script files. The code delivered to the browser will look very different from previous versions of VirtualViewer.

API

To make it easier to hook into VirtualViewer as it initializes, we look for two functions as VirtualViewer starts up. Define these functions in the global space of the iframe or window that VirtualViewer is running in.

beforeVirtualViewerInit

This function is called before VirtualViewer calls any initialization function. The virtualViewer object containing API will exist, but API functions may perform unexpectedly before initialization functions. beforeVirtualViewerInitis an appropriate place to call an alternate VirtualViewer initialization API, or initialize other objects.

Parameters: None

Returns: If you return true from this function, VirtualViewer will continue with normal initialization procedures. Return false if you are making calls to initialize VirtualViewer in beforeVirtualViewerInit, as these should override VirtualViewer's initialization.

afterVirtualViewerInit

This function is called after VirtualViewer has completed initialization. This is an appropriate place to assign callbacks, and perform initialization tasks that depend on VirtualViewer API.

Parameters: None

Returns: None

Thumbnail name customization

Thumbnail names can now be specified within the Pages tab

Code Sample:

```
virtualViewer.getCurrentState().setPageDisplayNames(["My First Page", "My Second Page"]);
```

Changing the Document display name

For changing the display name for the open Document tab, the individual pages within the Pages tab, & the hover tooltip over individual pages..

```
virtualViewer.getCurrentState().setDisplayName("newDisplayName")
```

Code Sample:

```
virtualViewer.setCallback("onDocumentLoad", function(event) {
  var state = virtualViewer.getStateForDocumentId(event.documentId);
  state.setDisplayName("My Document: " + event.documentId);
});
```

API call to toggle annotations on/off

virtualViewer.toggleAnnotationVisibility(show)

VirtualViewer 4.12 (and older) New Features and Corrected Issues

'show' is an optional boolean: if not provided, the function will toggle back and forth; if set to true/false it will make the annotations visible/invisible.

VirtualViewer 4.11 Bug Fixes

Annotation Revision History saving (fixed)

Emailing as original sends a tif (fixed)

Two Clicks to Move Annotation (fixed)

Mapping Alfresco permissions to Snowbound - document available from customer support

Annotation paste / save no longer fails but may be misplaced - still in development

ContentHandlerInput input now includes a httpServletRequest object in its table

pathnames missing within a war file corrected -

XHTML: File hangs - XHTML Files over 1000 pages long may cause problems for the viewer

New Features of VirtualViewer 4.10

Mobile Device Control Improvements

The user can pan on an image, if it's zoomed in, in all directions (imageScrollbars should be set to false in config.js for this to work).

Using two fingers. the user can pinch to zoom in or zoom out on the document. The zoom motion is not animated so the increase/decrease in size will happen when the user is done pinching.

- Scroll

- Pan

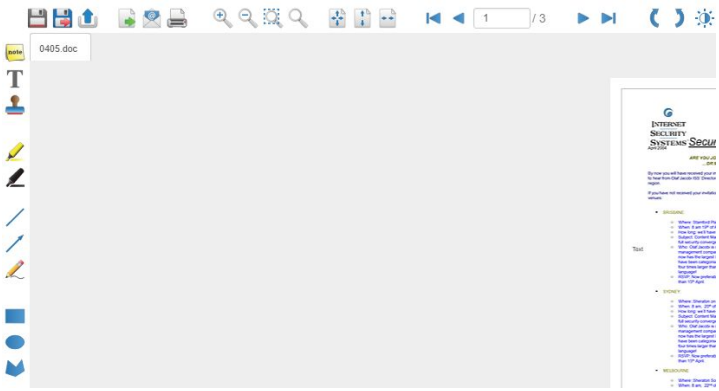
- Zoom In

- Zoom Out

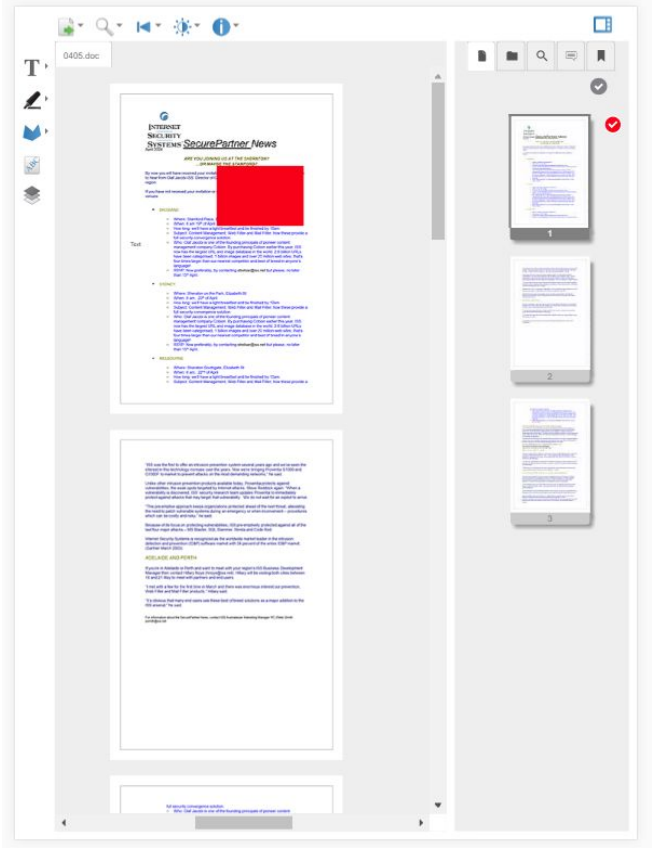
The viewer now works more elegantly in small iframes, on low-resolution monitors, and when browser windows resized smaller.

Toolbar details

On large screens, the toolbars look exactly the same as before:



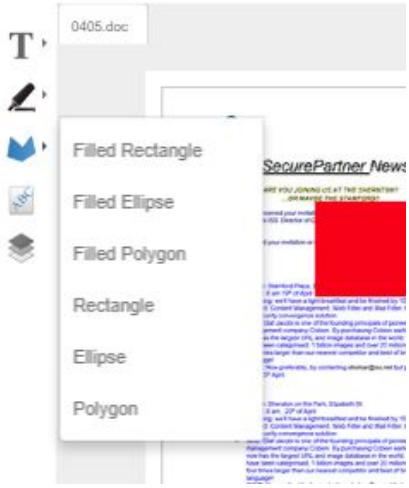
But as you shrink down, buttons collapse into menus. This happens as you shrink your monitor size, lower your resolution, or even shrink your browser. It's especially important for embedding the viewer in small iframes--like Alfresco--or for using the viewer on an iPad:



(Note that these icons are temporary.

Here, you can see what the dropdown menus look like:

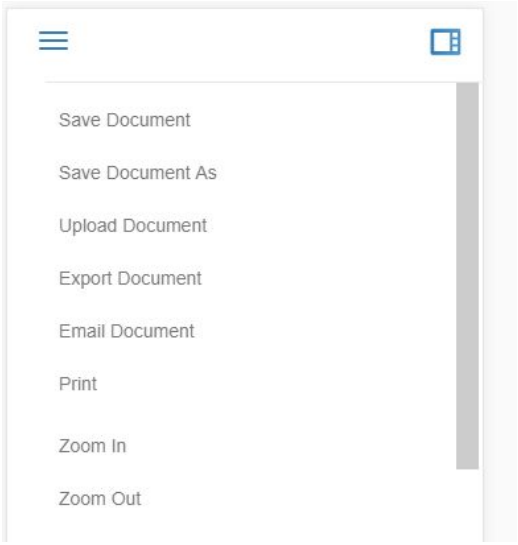
VirtualViewer 4.12 (and older) New Features and Corrected Issues

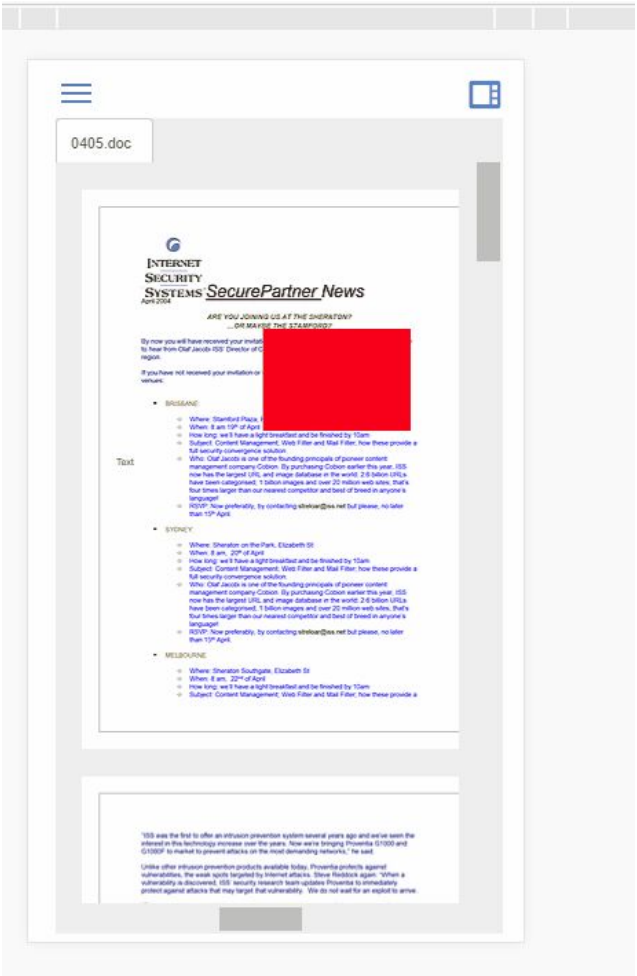


As you shrink down even further, into phone real estate or into the smallest your browser can get, we have even less space on the toolbar. So, we stuff all the options into a large menu, reachable by the three lines in the top left (called a "hamburger button"):

VirtualViewer 4.12 (and older) New Features and Corrected Issues

This menu takes up the entire screen when open:





Toolbar buttons are also now highly configurable.

A deep dive into the 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.

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.

VirtualViewer 4.12 (and older) New Features and Corrected Issues

-*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.)

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.

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

Unsupported Features on Mobile Devices

Drag and Drop

Search and redact are disabled if redaction is disabled

Document comparison, when offered, will not be supported for mobile devices

Though several mobile devices have been approved, we cannot guarantee that all mobile devices are supported.

Alfresco Quickshare Support

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

VirtualViewer 4.12 (and older) New Features and Corrected Issues

Revision history for Annotation Create Date/Time

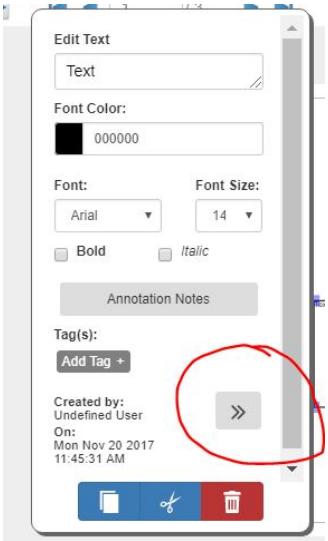
Use Case:

1. User 1 creates an sticky note on page 1 of a document, the userId/date/time are recorded
2. User 2 edits that same sticky note (color, size, placement, any change really....ect)
3. The userId/date/time are updated in a scrolling list reflecting each edit to that object.

- Works for all annotations, image stamps and redactions
- Resizing/moving records as a change
- Changing text records as a change
- Page manipulations or moving pages with annotations to a new/other document will not record a change
- The use of pages with annotations in a VD will record changes to the annotations
- A modification will be added if the annotation is modified and then saved.
- If the user changes the color, moves around the annotation, and expands the annotation, and then saves once, only one modification item will be saved.
- If an annotation is pasted, it will have a clean slate--it won't keep the modifications of the original.
- If a document is saved-as, the annotations on the new document will still have a modification trail but will not add a new modification for the saving event.

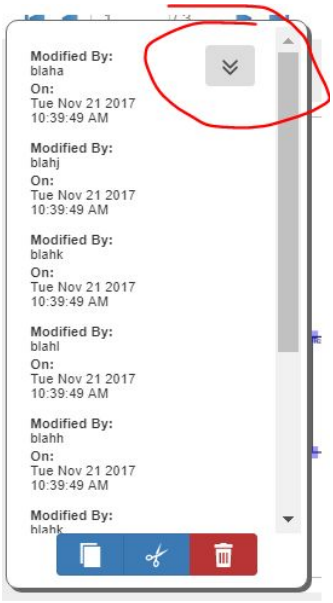
Display and Use

The revision history is displayed in the annotation popup by clicking an expando button:



VirtualViewer 4.12 (and older) New Features and Corrected Issues

And shrunk again by hitting the same button:



Filenet F_CREATOR Tag Support Added

NET 4.5.2+ Requirement Added for TLS 1.2 support

OCR Integration (beta)

This is a beta version of the OCR option in VirtualViewer. The final version is expected in the VirtualViewer v4.11 release where some alignment issues and ability to work with additional input file formats will be added. We also expect to offer a choice of OCR recognition engines in 2018.

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. Searching is performed using the Search tab in the thumbnail panel.

The original image will overlay the OCRed textual data to maintain the greatest similarity to the original document. The search text string will be highlighted. “Previous” and “Next” match buttons will work as normal. “Redact” and “Redact All Matches” work as normal. Applying redaction tags to results works as normal.

A wait icon will be displayed while the OCR process is running.

VirtualViewer 4.12 (and older) New Features and Corrected Issues

OCR will not be initiated if the input document is not raster. Saving to a PDF file is an option. Additional language support can be added by the customer.

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".

Fixed Customer Issues in VV 4.10

Small screen issues fixed

- The document no longer overlaps the thumbnail panel when the viewing canvas gets very small
- Thumbnails work on iOS devices, annotations may be drawn and modified on all sizes of screen
- The thumbnail pane starts out hidden if it's going to take up too much screen real estate
- Modal dialogs are usable on small screens
- Annotation context menu is usable on small screens
- Scrolling works on small screens

Other Fixed Customer Issues in VV 4.10

- IOS Mobile: Export to Tiff/Original opens only the 1st page in new tab
    - This is an issue with IOS where it only displays the first page of a multi-page TIFF. The other pages are preserved and viewable on other applications. (This is also true when viewing such documents in Gmail.
- VV Net 4.9: Constant reloading of thumbnails
- FileAndURLRetriever class missing from VV Java v4.9
- VV: PDF Arial to Times New Roman conversion error fixed
- Tabs, canvas, and thumbs can lose sync
- Annotation Highlight Rectangle does not use highlightFillColor
- Annotations are not burnt in via export
- Text Ann Color Selection
- Confirm Changes dialog box not appearing
- Snowbound annotation tags are overwritten
- Page size increased after redaction applied
- Filenet annotation values are overwritten
- PDF document comments are not being displayed
- VV Net HTML plug issue
- PaperPort 12 PDF: Large file stalls or fails to load
- VV .NET cannot get files over HTTPS TLS 1.2 on framework 4.5
- Color is not displayed and size is shrunk in pdf document
- Cannot view annotation with permission PERM_VIEW = 40

END OF RELEASE NOTES

VirtualViewer 4.12 (and older) New Features and Corrected Issues