VirtualViewer 5.1 Release Notes

- Snowbound Software Corporation
- questions@snowbound.com
- (617) 607-2010

Table of Contents

VirtualViewer 5.1 Release Notes

Page Limit Configuration for .XLSX Documents

Rendering large XLSX documents is both time- and memory-intensive. VirtualViewer 5.0 saw significant improvements in rendering speed, but by their nature, these large documents still use a great deal of memory. If VirtualViewer is not allocated enough memory, or if many users are opening large XLSX documents, the viewer application may experience memory exceptions.

Now, a new configuration parameter allows administrators to limit how many pages of an XLSX document will be rendered. The parameter is an `init-param` set in `web.xml`, called `xlsxMaximumPageLimit`. An example may be found in `web.example.xml`.

Setting this configuration parameter will cut off XLSX documents at the given page maximum. For example, an administrator may set `xlsxMaximumPageLimit` to 2. An XLSX document with one or two pages will render normally. An XLSX document with five pages will be cut off. On the client viewer, the document will only display two pages. A dialog will show to the user with a warning message that the document has been cut off. Additionally, the page count will not be displayed, but will be represented with an ellipsis: `...`.

The document itself is not affected by this limit. Opening a document will only limit what is displayed to the user; it will not modify the original document. Since viewer functionality like export, save, and save as *would* produce a page-limited result, these buttons and API functions are disabled if a document has been limited by this parameter.

New Configuration

The new configuration parameter is an `init-param` in `web.xml`:

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

A value of `0` is the default, which represents no page limitation. With a value of `0` set in `xlsxMaximumPageLimit`, all XLSX documents will render completely. Acceptable values for the parameter are positive integers.

Performance Optimization for Document Load

VirtualViewer now displays long documents, particularly SnowDoc formats, significantly faster.

The first and simplest optimization was completely on the client. Documents with one thousand pages or more would take longer than necessary to load, and would halt all UI operations until all page thumbnails could be rendered. All other pieces of loading are asynchronous and do not lock up the UI, so this experience stood out. Now, thumbnail rendering has been optimized to quickly stub out all the necessary thumbnails, and only create the more extensive HTML structure when the thumbnail comes into view.

The more complex optimization adjusted how the client loads documents. Previously, VirtualViewer would load a document by first loading a document model with a great deal of metadata about the document, and then loading metadata about a page image, and finally loading a page image. Waiting for the document model to load is necessary for most viewer operations, but it can take a long time to create this data structure, particularly for long documents or for complex formats like docx.

With new RasterMaster development, SnowDoc formats like docx are now processed slightly differently. All of the long document processing operations are handled in a separate thread. This thread persists in the background until the document is fully processed, an error occurs, or the thread is interrupted. The servlet thread can either wait for the SnowDoc processing thread to complete, or it can retrieve individual pages while the document is still processing.

This new development allows VirtualViewer to start retrieving pages much earlier. Now, VirtualViewer requests the document model data structure at the same time as the first page. The document model data structure still needs to wait for all document processing to be complete, and users will not be able to perform several operations until that is loaded; but pages will begin loading immediately. This effect will be most dramatic with SnowDoc formats that are entering the cache.

New Callbacks

- `pageLoaded` will be called once page load completes. Page load is the new initial load of a page, where the viewer will receive both metadata and image data for a page. The page image may fall out of the buffer and need to be reloaded; in that case, the image alone will be loaded. `pageLoaded` should be called only once for each page. The following parameters will be provided to the callback in the argument object:
    - `documentId` {String} The ID of the document whose page was loaded
    - `displayName` {String} If a new display name was sent with the page, this argument will store it. It will not yet be set in the viewer
    - `paneIndex` {number} The 0-based index of the pane displaying the loaded document
    - `loadedPageNumber` {number} The 0-indexed number of the loaded page
    - `searchableStatusBeforeLoad` {boolean} The document's searchable status--whether the document has machine-parsable text--may change on page load. This sends the old value.
- `pageLoadError` will be called if a page load fails. Page load is the new initial load of a page, where the viewer will receive both metadata and image data for a page. If page load fails, it will be retried once. The following parameters will be provided to the callback in the argument object:
    - `documentId` {String} The ID of the document whose page was loaded
    - `finalTry` {boolean} This will be true if the page failed on the second and final attempt to load
    - `paneIndex` {number} The 0-based index of the pane displaying the loaded document
    - `pageNumber` {number} The 0-indexed number of the failed page
- `pageCountUpdated` will be called if the page count updates. We may load pages before the document is fully loaded; if we do, and therefore if we don't have the correct page count yet, we update the page count as pages are loaded. The loaded page data will tell us if there's a next page, and the page count will be updated accordingly. The following parameters will be provided to the callback in the argument object:
    - `documentId` {String} The ID of the document that is currently loading
    - `paneIndex`

{number} The 0-based index of the pane displaying the loading document
  - `oldPageCount` {number} The page count before the update
  - `newPageCount` {number} The current page count, after the update

Stuck Thread Interruption

Most server platforms have tools to monitor and interrupt long-running application threads. Thread interruption is the safest way to stop a thread from processing. Attempting to kill a thread is extremely unsafe, and can create a number of insidious bugs.

With thread interruption, the server program will request that an application thread be interrupted. The listening application must act on the interruption by returning and exiting operation cleanly. Previously, VirtualViewer was not able to respond well to these thread interruption tools--the application did not check for interruption, and exited on error and success.

Now, VirtualViewer will check for interruption, and will return early if the thread has been interrupted. On the client side, the viewer receives a status code noting that operation has timed out. If the viewer receives three of these status codes for a particular document, it will no longer attempt to load the document.

Configuration

Each server platform will have a different mechanism for monitoring threads, and interrupting possibly stuck threads. For example, Tomcat implements a class called the StuckThreadDetectionValve; with an entry into server.xml, the server will use the StuckThreadDetectionValve to monitor long-running threads.

Changes to JavaScript Initialization Hooks

On initialization, VirtualViewer calls the two functions `beforeVirtualViewerInit` and `afterVirtualViewerInit`, if they exist. These functions are both intended to be defined to call custom code.

Simplification of `beforeVirtualViewerInit`

Now, the function `beforeVirtualViewerInit` is **not required** to return any value, and is **not** expected to call `afterVirtualViewerInit`. However, `beforeVirtualViewerInit` **may** return a value of `true` in order to cancel the VirtualViewer initialization process. If a VirtualViewer initialization API is called in `beforeVirtualViewerInit`, VirtualViewer will now detect that it has already been initialized, and will not attempt to reinitialize. VirtualViewer will call `afterVirtualViewerInit`; that is no longer the responsibility of the custom code. If `beforeVirtualViewerInit` has returned a value of `true`, initialization returns early.

Startup of VirtualViewer JavaScript code now follows this flow:

1. A VirtualViewer instance is created and assigned to the global variable `virtualViewer`.

2. `beforeVirtualViewerInit` is called, if it exists and is a function.

3. If `beforeVirtualViewerInit` has returned a value of `true`, initialization is canceled.
4. If the VirtualViewer instance has not yet been initialized, `virtualViewer.initViaURL()` is called. If the VirtualViewer instance has already been initialized, this step is skipped.
5. `afterVirtualViewerInit` is called.

Now, both `beforeVirtualViewerInit` and `afterVirtualViewerInit` take no parameters and are expected to return no values; `beforeVirtualViewerInit` may return a value of `true` to cancel initialization, but otherwise, VirtualViewer will detect whether initialization has occurred.

New Locations for Custom Code

Previously, `beforeVirtualViewerInit` and `afterVirtualViewerInit` could be defined within `index.html`. This could clutter `index.html`, and make it more difficult to upgrade.

Now, there are two new files in the `user-config` directory, `custom-code.js` and `custom-style.css`. Both are loaded by `index.html`, and are intended to hold only custom code and style. The stubs of `beforeVirtualViewerInit` and `afterVirtualViewerInit` have been moved from `index.html` to `custom-code.js`.

If definitions of `beforeVirtualViewerInit` and `afterVirtualViewerInit` are still in `index.html`, they must be moved to `custom-code.js` or must be defined before the script tag that loads `custom-code.js`.

Speech Synthesis for Documents

Several browsers implement a Javascript speech synthesis API to manually provide text-to-speech controls. VirtualViewer now may use those API to read documents that contain text.

When a user opens a document that contains text, a button to toggle text-to-speech controls will appear on the toolbar. Clicking the button activates the control bar, which allows the user to play, stop and pause the text-to-speech; the user may also use this bar to skip pages with the next and previous buttons.

Configuration

To ensure that VirtualViewer can read the document text, set the `enableTextExtraction` parameter in `config.js` to `true`.

Known Limitations

VirtualViewer's text-to-speech functionality will not read the correct text for documents with text that have rotated pages. If the user clicks rotate while the document is open, the text-to-speech function will read the correct text; once the user saves the document, the rotation is essentially "burned in" and the text itself is rotated, meaning that the text no longer can be read left-to-right.

The text-to-speech button will also remain deactivated if OCR is performed on a document, since OCR functions often return unpronounceable text.

Screen Reader Compatibility

VirtualViewer is now compatible with screen readers. In addition to the text-to-speech button described above, all the controls and the displayed document itself have been reorganized and updated to allow most screen readers to interact smoothly with the viewer. A screen reader will now read out navigation items as a user tabs through them or mouses over them, and will read out a displayed text document when it comes into focus. The text document is treated as the main content of the viewer.

The document may come into focus by tabbing through the viewer, or by tabbing to a new "skip to main content" link added at the top of the tab order, in the top left corner of the viewer.

Configuration

To ensure that VirtualViewer can read the document text, set the `enableTextExtraction` parameter in `config.js` to `true`.

Known Limitations

Essentially, VirtualViewer is extracting plain text from a document, and displaying it in a hidden HTML structure that a screen reader can navigate. Screen readers will therefore encounter the same limitations of the text-to-speech functionality described above, and to existing text selection, copy, and paste functionality existing in the viewer. A document may not be read if its format does not contain embedded text, if it was rotated such that the positional data of the text is no longer organized horizontally, or if OCR (Optical Character Recognition) returns poor results.

New Zooming Behavior for Document Compare

A new configuration item, `vvConfig.zoomLock`, controls whether two documents open in document compare will zoom together. This configuration is on by default; in order to turn it off, the configuration item must be set to `false`.

If `vvConfig.zoomLock` is not set or is set to `true`, zooming one document will zoom the other simultaneously. When opening a document in document compare, it will be initially set to the main document's zoom rather than the default zoom set in User Preferences. If `vvConfig.zoomLock` is set to false, VirtualViewer's previous behavior will apply, and documents may be zoomed independently in document compare.

Two new API functions programmatically control whether zoom is locked in document compare:

- `toggleZoomLock` will set the zoom lock to the opposite of its current setting. This API takes no arguments.
- `setZoomLock` allows zoom lock to be manually turned off and on.
    - `lock` {boolean} Pass in `true` to lock zoom for document compare, and `false` to unlock zoom.

Audio Support

In addition to video, VirtualViewer can now play audio files that are supported by most browsers. There is no editing or annotation support at this time; audio can only be viewed and downloaded. Audio format support will depend on the capabilities of the web browser.

Supported formats

VirtualViewer uses the browser's HTML5 audio player to play audio, and can play all types of audio supported by a browser's player. Most browsers support MP3, M4A, FLAC, Ogg and WAV. This browser compatibility chart has more details: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats

User preferences

There are two new preference options for audio. These options can be viewed and modified, along with video preferences, in the Media Preferences tab in User Preferences:

- Audio Loop: whether to start the audio again from the beginning when it has finished playing.
- Continue Audio on Tab Switch: whether to keep playing an audio file after you have opened or switched to another document. If this is enabled, you can listen to an audio file while viewing other documents in VirtualViewer.

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

New and Changed Client-side API

- `setPageDisplayNames` will set text display names for thumbnails, if `vvConfig.displayThumbFooters` is set to `true`. The pages will display the given names instead of "Page 1", "Page 2", and so on. This API was previously available through a non-public API, and this safer version, `virtualViewer.setPageDisplayNames(["A", "B"]);`, should now be used instead.
    - `displayNames` {string[]} An array of the display names to be used. This array should be the length of the document (the length returned by `virtualViewer.getPageCount()`). Any undefined or null values in the array will cause the page to display the page number.

Fixes and Changes

Updated Translations and Localization

Due to the accessibility improvements in VirtualViewer this release, many strings were updated and localization keys were shifted. VirtualViewer has significantly updated locale files as a result.

Zoom Settings in Internet Explorer

By default, VirtualViewer uses the Javascript image handling library `pica` to sharpen images when they are zoomed to less than 100%. The configuration item `vvConfig.useBrowserScaling` controls whether to use `pica`; on older browsers, `pica` can be slow. On Internet Explorer, `pica` may take minutes to zoom out an image, which uses an unacceptable amount of processing power and memory, when the user will likely close the document before seeing the zoomed image. This also has consequences for certain API like `invertImage`, which depend on the presence of a zoomed image.

Now, VirtualViewer disables `pica` by default when running on Internet Explorer, essentially treating the process as if `vvConfig.useBrowserScaling` were set to `true`. Set `vvConfig.serverScaling` to `true` to use server scaling instead.

Miscellaneous

- The language attribute in the HTML tag is now dynamically set to reflect the settings in `vvDefines` and the browser
- Bookmarks reference the correct pages after reordering or deleting pages in a document
- The annotation indicator no longer appears on thumbnails of cropped pages--annotations may not be placed on cropped pages
- When a document in an inactive tab is saved, it will update the tab name but will not erroneously display in the active tab
- The API `invertImage` now functions as expected in Internet Explorer

VirtualViewer 5.0 Release Notes

New User Interface Design

All toolbar icons and VirtualViewer's main user interface have been updated to a colorful, flat, and modern look. There is no change to the function or placement of any icons, and no changes to configuration. This feature is enabled by default.

Icons are now displayed as an icon font, rather than individual images loaded by URL. This speeds up the loading time of the viewer, especially on highly latent connections, and provides a smoother user experience.

Custom buttons may still be loaded as individual images. To do this, place the image in a web-accessible directory and point to the image path in the `iconImage` attribute of a button configuration item in `user-config/toolbar-config.js`. To blend in with VirtualViewer toolbar icons more smoothly, provide a blue version of the image to display when the user hovers over the icon; set additional CSS to display this blue icon in place of the original image when the CSS pseudoselectors `:hover` and `:focus` are active.

SnowDoc for VirtualViewer Java

VirtualViewer has significantly improved the performance of `docx`, `xlsx`, `pptx`, and `doc` files, using the new SnowDoc feature of RasterMaster Java 20.0.

OfficeX files are performance-intensive to lay out, since they generally don't contain data specifying where pages begin and end, or locations of characters on pages. Previously, VirtualViewer and RasterMaster were redoing that layout work multiple times; with SnowDoc, layout information is cached between calls so the performance-intensive processing only needs to be done once.

The most dramatic change can be seen by loading a long `docx` file in VirtualViewer and navigating to the last page. Before 5.0, loading the final page of `docx` document would take significantly longer than loading a beginning page. Now, the loading time is short, and no longer than any other page in the document.

This new feature does not require any updates to configuration or to content handler code. Initialization parameters for OfficeX and doc formats in `web.xml` will be respected, with the exception of the default page dimensions for xlsx files. In the case of xlsx files, the `xlsxWidth` and `xlsxHeight` initialization parameters will be disregarded in 5.0 in favor of the native dimensions of the document.

It is important to note that caching is crucial to the performance improvements of these files. The initial load of a file with SnowDoc takes the most time; VirtualViewer needs to cache these documents to avoid taking that performance hit over and over. If current configuration or content handler code disables use of ehcache in VirtualViewer, consider enabling caching for `docx`, `xlsx`, `pptx`, and `doc` files.

Load Font Files into VirtualViewer Java

It is now possible to include font files in the VirtualViewer Java war file. Those fonts will be used to render documents, even if the font is not installed on the server environment running VirtualViewer. The largest effects of this new feature are on Office formats. Previously, in order to accommodate uncommon or rarely-used fonts in an Office document, the font would need to be installed on the server running VirtualViewer, or the document might experience problems with text layout. With this feature, specific fonts no longer have to be installed on the entire server.

The effects of loading a font will differ depending on the format of the loaded document, and whether a document is loaded as an SVG or a PNG. For example, a .docx document loaded as an SVG will display with the correct layout, but may not render in the font unless the font is installed on the client machine. A .docx document loaded as a PNG will display with both the correct layout and the correct font rendered into the image.

To enable this feature, simply add a `.ttf` file to a folder called `fonts` within the VirtualViewer war. Within the war, there are folders called `js` and `css`. Add a folder called `fonts` as a sibling to these, so the folder structure looks like this:

```
VirtualViewer war file
--WEB-INF/
--resources/
--js/
--fonts/    <===
--css/
```

Add `.ttf` files to the `fonts` folder:

```
VirtualViewer war file
--WEB-INF/
```

```
--resources/
--js/
--fonts/
----MyFont-Italic.ttf <===
--css/
```

Finally, restart VirtualViewer to initialize the new fonts.

Improved Documentation

VirtualViewer API and code documentation will no longer be included in the PDF manual at www.snowbound.com, but will be generated fully by Javadoc and JSDoc utilities. This allows for complete and up-to-date documentation of API and public code.

The new documentation may be found at the VirtualViewer Manuals Page. Some highlights are:

- VirtualViewer client-side Javascript API documentation, generated by the JSDoc utility
- VirtualViewer Java Content Handler API documentation, generated by the Javadoc utility
- A transition guide for upgrading to VirtualViewer version 5.0. This document, written as a technical guide, steps through the process of modifying the content handler and updating interaction with VirtualViewer API

Content Handler Interface Updated (Content Handler Modification Required)

The interface between the content handler and VirtualViewer have been updated to make it easier to use and set up. Many of these changes will require existing content handlers to be updated to work with the new standard.

A detailed guide on transitioning to 5.0 can be found here.

FlexSnapSI to VirtualViewer

A few classes with names containing FlexSnapSI, our old product name, have been updated to VirtualViewer. This includes the new VirtualViewerContentHandlerInterface and VirtualViewerAPIException.

Usable classes organized into packages

All interfaces and classes that are usable by the content handler can now be found in the `com.snowbound.common` and `com.snowbound.contenthandler` namespaces, and all classes in those namespaces have a new detailed API documentation on our website. This will make it very easy to find what is needed for your content handler and see what additional tools are available.

Classes in namespaces other than `com.snowbound.common` and `com.snowbound.contenthandler` are for internal use by VirtualViewer and may be obfuscated.

Content handler interfaces modularized

VirtualViewer's interfaces have been refactored to make it easier to separate out the features you want to implement for your content handler. There is now only one required content handler interface, VirtualViewerContentHandlerInterface, which contains every method a content handler must implement. All other interfaces in `com.snowbound.contenthandler.interfaces` can be added to enable their features or left out without the need to 'stub out' methods.

Many of our interfaces for new features were already modular in this fashion, but now more older features like `getAvailableDocumentIds` and event notification have been modularized in the same way.

Existing content handlers should be updated to make sure they implement all the interfaces they need, or their old methods will not be called by VirtualViewer.

Explicit commands for content handler actions

The content handler method `saveDocumentComponents` is now expected to only save, not delete, document components such as bookmarks, document notes, and watermarks. There are three new content handler methods--`deleteBookmarkContent`, `deleteNotesContent`, and `deleteWatermarkContent`--to handle deletion.

Previously, `saveDocumentComponents` was expected to delete a component such as an XML file storing Snowbound bookmark data if it received a `null` value. Now, `saveDocumentComponents` is expected to make **no changes** on receiving a `null` value for a component. This is crucial, because VirtualViewer will now send a `null` value for a component if there have been no changes to save. VirtualViewer will call a `deleteBookmarkContent`, `deleteNotesContent`, or `deleteWatermarkContent` to delete all bookmarks, document notes, or watermarks.

With this feature, VirtualViewer is more maintainable in the future, may send smaller amounts of data, and solves a saving problem.

This feature requires content handler updates for VirtualViewer to function, but the content handler may still compile if the updates are not completed.

ERROR_MESSAGE fully replaced by VirtualViewerAPIException

Throwing a VirtualViewerAPIException is now the only way to pass an error to VirtualViewer. The erroring request will be interrupted exception's message will be logged and may be displayed to the end user.

New Thumbnail Tab for Viewing Annotation Information

A new tab in the thumbnail pane, displayed to the right of the main document pane, lists metadata for the annotations in a document. Each annotation is described by a small block of text containing the type of the annotation, the page that contains it, and its creation and most-recent modification details. This information is updated live as the user creates, deletes, and modifies annotations.

The annotation information tab is disabled by default, and may be turned on by setting the configuration item `showAnnotationList` in `config.js` to `true`.

New and Changed Callbacks

The VirtualViewer callback system is now implemented on top of JQuery custom events. The existing API to set a callback, `virtualViewer.setCallback(callbackName, handlerFunction)`, still has the same signature; now, calling `setCallback` multiple times will set multiple handler functions to a single callback event. The handler functions will be called synchronously in an arbitrary order. Note that `disableTextContextMenu` and `disablePageManipulationContextMenuOptions`, the only callbacks to support a return value from the handler, may still only have one handler function active at a time.

Handler functions can also be removed by a new API, `virtualViewer.removeCallback(callbackName, removeVirtualViewerHandlers)`. Removing a callback by setting its handler function to null or undefined (for instance, `virtualViewer.setCallback('rotation', null);`) will no longer deactivate the handler function; `removeCallback` should be used instead.

Using this mechanism for callbacks not only allows for more flexible integrated code, it allows for custom callbacks to be added without overriding VirtualViewer default callbacks. For instance, VirtualViewer code uses callbacks for default behavior after uploading or saving a document as. Removing a callback with the second parameter set to `true` will remove the default VirtualViewer behavior for such a callback, but otherwise, behavior can now be added while preserving the old.

Callback API

- `virtualViewer.setCallback(callbackName, handlerFunction)` adds a callback handler function. The handler function will be called synchronously from VirtualViewer code.
  - `callbackName` {string} The name of the callback. For instance, pass in `"saveAsDocument"` to set a handler function called when a document has completed saving as. Callback names and detailed descriptions can be found in the VirtualViewer clientside documentation.
  - `handlerFunction` {function} This function will be called and given a single argument object. The attributes of the argument object vary from callback to callback. VirtualViewer code will not respect the return value of a handler function except in two cases: the callbacks `disablePageManipulationContextMenuOptions` and `disableTextContextMenu`.
  - Returns `true` if the callback name was found and the handler function was set

- `virtualViewer.removeCallback(callbackName, removeVirtualViewerHandlers)` deactivates and removes all handler functions for the given callback name.
  - `callbackName` {string} The name of the callback. For instance, pass in `"saveAsDocument"` to remove handlers that would fire when a document has completed saving as. Callback names and detailed descriptions can be found in the VirtualViewer clientside documentation.
  - `removeVirtualViewerHandlers` {boolean} In some cases, VirtualViewer supplies default handlers to callbacks. Most of the time, these handlers should be left in place to avoid unexpected behavior. However, it may be necessary to turn off default VirtualViewer handlers to improve integration. In this case, set this parameter to `true`. In all other

    cases, this parameter may be left out, and only custom callback handler functions will be removed.
  - Returns `true` if the callback name was found

New Callbacks

- `onShowSelectTextContextMenu` will be called from the show event of the menu that appears when right-clicking the document. This menu may contain options to copy or redact text, to perform OCR, or to add highlight annotations. The following parameters will be provided to the callback in the argument object:
  - `documentId` {String} The ID of the document that the menu appears over

Modified Callbacks

- `onDocumentLoad` will be called when the document model has finished loading. Once the document model has been loaded, the viewer considers the document itself to be loaded and begins loading image- and page-specific information. The following parameters will be provided to the callback in the argument object:
  - `documentId` {String} The ID of the document that has completed loading
- `rotation` will be called when a page is rotated. Pages can be rotated to 0, 90, 180, and 270 degrees. When freshly loaded from the server, a page always starts at 0 degrees rotation, even if it was rotated in a previous VirtualViewer session. The following parameters will be provided to the callback in the argument object:
  - `documentId` {String} The ID of the document whose page was rotated
  - `page` {number} The 0-based index of the page that was rotated
  - `rotatedTo` {number} The angle, in degrees, that the page was rotated to, compared to its original zero position
  - `rotatedBy` {number} The angle, in degrees, that the page was rotated by, from its position before rotation
  - `degrees` {number} The angle of rotation. This angle will match either rotatedTo or rotatedBy
- `imageLoadCompleted` will be called when an image has completed loading. Loading in VirtualViewer can be a complicated process with several fallbacks for faulty images. This callback is called at the end of the process, when an image is completely loaded and is ready to draw. The following parameters will be provided to the callback in the argument object:
  - `val` {boolean} Whether the image was loaded
  - `documentId` {string} The ID of the document whose image is loading
  - `page` {number} The 0-based index of the page that was loaded
- `overlayImageLoadCompleted` will be called when an image has completed loading for use as a template overlay. The following parameters will be provided to the callback in the argument object:

    - val {boolean} Whether the image was loaded
    - documentId {string} The ID of the current base document
    - overlayDocumentID {string} The ID of the document that is made transparent and displayed over the base document

- `imageLoadRequested` will be called when a request for an image fires off to the server. VirtualViewer uses a buffering system to preload images. The following parameters will be provided to the callback in the argument object:

    - `val` {boolean} Whether the image was requested
    - `documentId` {string} The ID of the document whose image is loading
    - `page` {number} The 0-based index of the page that is being requested

New and Changed Client-side API

- `rotateSelectedPagesBy` rotates specified pagees of the current document by 0, 90, 180 or 270 degrees from its current state.
    - `pages` {number[]} An array of the 0-based page indices that should be rotated.
    - `degrees` {number} A value of 0, 90, 180 or 270. These values may be negative.
- `closeTab`
    - `tabId` {number|string} Either the 0-based index of the tab or the document ID that the tab refers to.
- `removeWatermarkFromPage`
    - `page` {number} The 0-based index of the page that the watermark is removed from. If a page has multiple watermarks, all the watermarks will be removed from that page.
- `hideAnnTagsPopUp`
    - `hide` {boolean} Whether to hide the annotation tag section. If this value is false or undefined, it will not hide the annotation tag section.

API with New Argument Objects

The following API previously took upwards of ten arguments. Parameter lists as long as that are unwieldy and fragile. These functions now take argument objects, with the parameter names as keys and the arguments as the values. Possible attributes of the arguments object are noted here as `options.parameterName`.

- `saveDocument` will save the current document, including annotations and any image manipulations. The most basic call of this function is simply virtualViewer.saveDocument(), which saves the current document as it is. Passing in additional parameters can save documents other than the current (pass in a docId); can save a document as (pass in options.newDocumentId); and in Save As can include or exclude annotations, redactions, pages, and many more aspects of a VirtualViewer document.

    - `docId` {string} The ID of the document to save. This document should be currently open (though it may not be currently displayed, it should at least be open in a VirtualViewer tab). If not set, the current document will be saved. This is the only parameter *not* passed in the `options` object.
    - `options` {object} An object storing optional parameters. All following parameters should be passed as attributes in this object.

- `options.newDocumentId` {string} Set this parameter to save the current document as a new document in the system. If set, this function operates as "save as". The original document will remain unchanged, and VirtualViewer will ask the content handler to save a new document with a new ID. Several of the following parameters are designed to only affect Save As.
- `options.newDisplayname` {string} Set the display name of the new document. This will only be respected if newDocumentId is set.
- `options.burnRedactions` {boolean} Set to true to permanently burn redactions into the new document. If redactions are burned in, the underlying image is converted to the redaction's black box, and underlying text is removed. This is not reversible.
- `options.includeRedactionTags` {boolean} Whether or not to include redaction tags. This is only respected if burnRedactions is true.
- `options.includeTextAnnotations` {boolean} Used to permanently burn text annotations into the new document.
- `options.includeNonTextAnnotations` {boolean} Used to permanently burn non-text annotations into the new document.
- `options.copyAnnotations` {boolean} Used to copy annotation layers (including redactions) into the new document.
- `options.includeWatermarks` {boolean} Whether to permanently burn watermarks into the new document. If true, copyWatermarks setting is ignored and watermarks are not copied.
- `options.pageRangeType` {string} Either "all", "complex" or "current". This parameter is only respected in Save As.
- `options.pageRangeValue` {string} A range of pages numbers to export. Dashes and commas can be used: for instance, "2-4,7,10". This will only be respected if pageRangeType is set to "complex". This parameter is only respected in Save As.
- `options.copyWatermarks` {string} Whether to save/copy watermarks alongside the document. Watermarks saved with this parameter will be editable in the future.
- `options.saveAsFormat` {string} This can be "Original", "PDF" or "TIFF". Note that "Original" may only be used when exporting a document without including annotations, redactions, watermarks, or document notes. Additionally, to export as the original format, pageRangeType must be "all". This parameter is only respected in Save As.
- `options.skipOverwriteDialog` {boolean} If the current document will be overwritten by a save action, VirtualViewer alerts the user with a dialog. Provide true to this parameter to suppress that dialog.
- `options.documentPaneIndex` {number} If document compare is open, provide an index to specify whether to save the first or second open document. Provide 0 for the left-hand document and 1 for the right-hand document. This may be helpful if the user has the same document open in both panes.

* `exportDocument` will export the current document as a PDF, a TIFF, or the original format, and locally download the resulting file.

  - `options` {object} The arguments object. All following parameters should be passed as attributes in this object.
  - `options.exportFormat` {string} Either "Original", "PDF" or "TIFF". Note that "Original" may only be used when exporting a document without including annotations, redactions,

watermarks, or document notes. Additionally, to export as the original format, pageRangeType must be "all".

  - `options.fileExtension` {string} This is based on the exportFormat parameter. If exportFormat is "Original", then the extension must be the current file extension of the document. If "PDF", the extension should be "pdf". If "TIFF", then "tif". Importantly, the extension can only be tif, pdf, or the original extension. VV will not convert a document to any format other than PDF, TIFF, or the document's original format.
  - `options.includeTextAnnotations` {boolean} Whether or not to include text annotations.
  - `options.includeNonTextAnnotations` {boolean} Whether or not to include non-text annotations.
  - `options.burnRedactions` {boolean} Whether or not to burn in redactions. If redactions are burned, the redaction becomes part of the image and any underlying text is removed. It is not reversible.
  - `options.includeRedactionTags` {boolean} Whether or not to include redaction tags.
  - `options.includeDocumentNotes` {boolean} Set this to true to append document notes as text pages at the end of the document.
  - `options.includeWatermarks` {boolean} Set this to include watermarks in the exported document.
  - `options.pageRangeType` {string} Either "all", "complex" or "current". "all" will export all pages; "current" will export just the current page; but "complex" requires the parameter pageRangeValue to be set as well.
  - `options.pageRangeValue` {string} A range of pages numbers to export. This may look like: "1-2,3,6,7-10". VV allows commas and dash-separated ranges.
  - Returns true if the export request has successfully been fired off, and a falsey value if not.

- `printDocument` will print a document, using the browser's print dialog.

  - `options` {object} An object storing optional parameters. All following parameters should be passed as attributes in this object.
  - `options.documentId` {string} The ID of a document to print. This document must be open in the viewer.
  - `options.printToPDF` {boolean} If true, a PDF version of the document will be exported to a file. The user will be presented with a save dialog.
  - `options.includeAnnotations` {boolean} Whether to include annotations on the printed document.
  - `options.includeRedactions` {boolean} Whether or not to fully redact the printed document.
  - `options.includeRedactionTags` {boolean} Whether or not to include redaction tags (only used when redactions are applied to the document).
  - `options.includeWatermarks` {boolean} Whether or not to print with any of the VirtualViewer-generated watermarks currently on the document.
  - `options.includeDocumentNotes` {boolean} Set this to true to append document notes as text pages at the end of the document.
  - `options.pageRangeType` {string} Options are "all", "complex" or "current". If "complex", the parameter pageRangeVal must be included; otherwise it will be ignored.
  - `options.pageRangeValue` {string} A range of pages numbers to print (only used for

"complex" pageRangeType). Commas and dashes may be used. For instance, "2-4,7,10-11" would print only pages 2, 3, 4, 7, 10 and 11.

- `emailDocument` will email the current document as a PDF, a TIFF, or as the original format. VirtualViewer email requires some server-side configuration.
    - `options` {object} The arguments object. All following parameters should be passed as attributes in this object.
    - `options.format` {string} Either "Original", "PDF" or "TIFF". The document must be emailed as a .pdf or .tif file if annotations, redactions, watermarks, or document notes are included, and if pageRangeType is any value other than "all".
    - `options.includeTextAnnotations` {boolean} Whether or not to include text annotations.
    - `options.includeNonTextAnnotations` {boolean} Whether or not to include non-text annotations.
    - `options.burnRedactions` {boolean} Whether or not to burn in redactions. If redactions are burned, the redaction becomes part of the image and any underlying text is removed. It is not reversible.
    - `options.includeRedactionTags` {boolean} Whether or not to include redaction tags.
    - `options.includeDocumentNotes` {boolean} Set this to true to append document notes as text pages at the end of the document.
    - `options.includeWatermarks` {boolean} [] Set this to true if watermarks should appear on the emailed document.
    - `options.pageRangeType` {string} Either "all", "complex" or "current". "all" will email all pages; "current" will email just the current page; but "complex" requires the parameter pageRangeValue to be set as well.
    - `options.pageRangeValue` {string} A range of pages numbers to export. This may look like: "1-2,3,6,7-10". VirtualViewer allows commas and dash-separated ranges.
    - `options.fromAddress` {string} The sender's email address. The default can be changed in config.js.
    - `options.toAddresses` {string} A comma-separated list of recipient email addresses. For instance, "myemail@email.com,youremail@email.com". The default can be changed in config.js.
    - `options.ccAddresses` {string} A comma-separated list of email addresses to CC. For instance, "myemail@email.com,youremail@email.com". The default can be changed in config.js.
    - `options.bccAddresses` {string} A comma-separated list of email addresses to BCC. For instance, "myemail@email.com,youremail@email.com". The default can be changed in config.js.
    - `options.subject` {string} The subject line (title) of the email.
    - `options.emailBody` {string} The message (body) of the email.
    - Returns true if the email request is sent off successfully, and a falsey value if not.

Loading Custom Image Stamps

With VirtualViewer's image stamp annotations, users can apply a set of small, administrator-defined images to a document. The available images were defined in `web.xml` or `web.config`, and provided in VirtualViewer's `resources` directory. This method did not allow dynamic adjustment to the list of stamps, and could cause problems on different servers.

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

New API

Several of the following API 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
}
```

- `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.

Configuration

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.

Due to the URI provided to the stamp object, stamp images may be stored in any accessible location. That may still be locally in the `resources/stamps` directory within VirtualViewer.

Retain Scroll Position Between Documents

The configuration item `fitLastBetweenDocuments` overrides the default zoom level, and will open a document at the same percentage of zoom as the last-open document. For instance, if a user zooms to 50% on a document open in the first tab, the document in the second tab will also appear at 50% zoom.

Now, there is a new configuration item to do the same with scrolling: `positionLastBetweenDocuments`. This will etain the scroll position of the previously-open document. So, for instance, the user may scroll document A to the third page, and scroll a little to the right to center a particular form field on the screen. When the user opens document B, it will open scrolled to the third page and a little to the right. This setting works best with `fitLastBetweenDocuments` enabled, and with documents of the same approximate size.

To enable, set the configuration item `positionLastBetweenDocuments` to `true` in `config.js`.

Fixes and Changes

Content Handler

In addition to the major content hander changes noted above, some minor fixes and changes have been made.

- Some deprecated and/or unused properties have been removed from ContentHandlerInput and ContentHandlerResult.
- Event notification properties are now deprecated on ContentHandlerResult. The equivalent properties on ContentHandlerInput should now be used.
- The annotation permission level constants on VirtualViewerContentHandlerInterface have been deprecated. The new PermissionLevel enum should be used instead.

Improved EhCache Error Handling in VirtualViewer Java

Previously, an incomplete or malformed ehcache.xml document would completely shut down VirtualViewer. Now, VirtualViewer handles and logs the error, and sets the cache that it was trying to configure as a no-op cache.

Miscellaneous

- Event notification now provides only necessary parameters to the content handler.
- The PermissionsEntities annotation layer property has been removed. This is not related to annotation permission levels; permission levels remain.
- Assigning the `clientInstanceId` with `virtualViewer.setClientInstanceId()` can now be done in `beforeVirtualViewerInit`; previously the value would be overwritten on initialization.
- Boostrap and JQuery external libraries are updated.
- Default fit and zoom preferences apply properly to newly-loaded documents.
- Posted messages are no longer caught by a VirtualViewer listener unless they apply to VirtualViewer.
- The last tab open in the viewer can now be closed, leaving an empty viewer.
- Only display the annotation tag section in the annotation pop up menu if tags are available.
- Annotation size "nubs" are now correctly sized for touch and non-touch environments.
- VirtualViewer Java no longer clutters the server log with irrelevant, non-VirtualViewer messages.
- `vvConfig.useBrowserScaling` turns off Pica scaling.
- The magnifier now displays a small 'x' button, so it can be closed directly rather than by clicking on the toolbar button a second time.
- VirtualViewer Java server code is now obfuscated to further clarify public and private code, improving content handler stability. All `public` classes should still be available for use. The VirtualViewer 5.0 transition document has more details on public and private classes in 5.0.
- The `JSESSIONID` cookie is restored and updated on the VirtualViewer client on each server request.

VirtualViewer 4.14 Release Notes

New Features

Document Filter

There is now an easy way to filter available documents by name in the document navigation pane. This feature is enabled by default, but can be disabled by setting the config.js parameter `showDocumentFilter` to false.

API for Retrieving User Display Name

`virtualViewer.getUsername()`: A new API has been added to programmatically retrieve the current user's displayed user name on the client.

- `virtualViewer.getUsername()` takes no arguments

- Returns a string containing the user name currently set in the browser

Streaming Video Support

Virtual Viewer now progressively loads videos when possible. This allows the user to play the video as it buffers, instead of waiting for the entire video to load before allowing playback. Browsers may not support progressive playback on certain files that have not been optimized for progressive loading, on certain files that the browser does not support, or for other factors of the browser's implementation of video. If the browser is not able to load a video progressively, then it will fall back to previous behavior and load the entire file.

This feature requires no configuration and is on by default. Video behavior in VirtualViewer is the same as before this feature; the only visible change is that the user may be able to play a video in VirtualViewer before the entire file has loaded. There are no API changes or additions.

Overwrite Original Format Notice

When a user makes a page manipulation (e.g. crop, rotate, insert a page, remove a page etc) on a document that is not TIFF or PDF and saves, the document's original format is always overwritten to either TIFF or PDF. With this feature the user is warned when saving if the original format will be changed. The notice dialog gives the user three options:

"Save and Overwrite": Continue the save and overwrite the original document's format.

"Save to New Document": Close the dialog and open the "Save As" dialog.

"Cancel": Quit the save operation and close the dialog.

The dialog only pops up if the user has made a page manipulation and the original file format is not TIFF or PDF. Simply saving annotations, or modifying a document that is already a TIFF or PDF, will not result in overwrite. This feature does not change saving behavior; it notifies the user of the current saving workflow. This warning can be prevented from appearing by setting `enableSaveOverwriteWarning` in config.js to `false`.

New and Changed Callbacks

- `imageLoadCompleted` will be called when an image has finished loading and is able to be displayed. Note that this callback was previously named `imageLoadFinished`; this callback `imageLoadCompleted` is a replacement of `imageLoadFinished`.
- `afterTabClosed` will be called after a tab closes successfully. It will not fire if there's an error while closing the tab, or if the user initiates closing the tab and cancels. The following parameters will be provided to the callback in the argument object:
    - `closedDocumentId` {String} The ID of the document that has just been closed.
- `onLoadUsername` will be called when User Preferences code has finished loading a username from localforage, or has found that there is no username to be loaded. The following parameters will be provided to the callback in the argument object:
    - `loadedUsername` {String} The user name that has just been retrieved from localforage
    - `previousUsername` {String} The user name that was previously set in the viewer

Fixes and Changes

Multiple network requests for each image on Internet Explorer and Edge

Previously, a quirk of Internet Explorer and Edge's image loading workflow could create a race condition. Both browsers may have significant time between when the image is done loading and when the image is usable; due to a bug in VirtualViewer's loading process, this could lead VirtualViewer to request the image again.

Now, VirtualViewer accounts for Internet Explorer and Edge with more nuanced checks for image readiness, preventing multiple requests.

Large images and SVGs on Internet Explorer

As an older browser, Internet Explorer can handle very large images poorly: if a web application uses a great deal of memory, Internet Explorer will behave in unexpected ways. A large image with a high DPI, or multiple large images, may not be displayed or may cause errors.

Now, VirtualViewer has several fallbacks in the event of an image loading or drawing failure. First, as before, if an image is loaded as an SVG, VirtualViewer will attempt to reload it as a raster image. After that, if the image is still too large or still cannot be drawn, VirtualViewer will load a downscaled, smaller image, as an attempt to use less memory in the browser. Beginning with such large and high-resolution images means that displaying a slightly smaller image will not provide a dramatic degradation in quality.

If the downscaled image still fails to function, VirtualViewer now has more fallbacks beyond displaying a blank page; an expanded version of the page's thumbnail will be displayed in place, to allow the user to manipulate the page and its annotations.

Miscellaneous Fixes and Changes

- Fixed a bug on VirtualViewer .NET only, where sparse documents would display the first document's image as every page. Now, sparse documents are properly displayed on VirtualViewer .NET.
- `enableCacheObfuscation` is no longer required to be set in the client-side config.js: only the server-side configuration is needed and the client setting will be ignored.
- Fixed a bug where if a user printed two documents in very short succession, VirtualViewer might print the last document instead of the most recent
- Improved annotation selection to make it more natural. A user must now click on the visible line of a line or arrow annotation in order to select it, instead of anywhere in its large bounding box
- Fixed an issue with the annotation tag dropdown
- Ensured the annotation navigation panel hides and shows correctly when switching between

documents

- Previously, the toolbar jump-to-page text box in Internet Explorer would behave unexpectedly, and sometimes interpret a backspace as a browser "back" command. Now, the text box behaves in a standard manner
- Fixed user interface problems in the layer manager dialog
- Removed a source of error in the layer manager dialog by modifying the layer deletion workflow. Previously, the layer would be deleted on the server immediately upon clicking "OK" in the layer manager dialog. Now, the layer will be deleted on the server only when the user saves the entire document
- Prevent Microsoft Edge from cutting off the bottom of extremely long documents
- Addressed video loading and downloading bugs
- Fixed a problem where document thumbnails could appear even if the thumbnail tab was disabled
- Videos resize properly in Edge
- If `vvConfig.enableSingleClickImageRubberStamp` is set to false, the stamp now draws in the correct location, instead of initially appearing off the page
- Fixed a bug where redaction buttons in the search panel might be enabled for documents without text
- Fixed a subtle bug could appear where drag-and-dropping a page thumbnail on a document tab, in exactly the right place, could cause a browser error
- Ensure that document notes load properly when switching between tabs
- Watermarks may now apply to a document created with Copy/Cut to New Document, if requested by the user
- Improved VirtualViewer's treatment of document IDs with special characters on .NET
- Fixed a bug where the dialog asking permission to OCR would appear inappropriately during document compare
- Update logic in the Export Document dialog so a user can no longer export a document in its original format while including document notes
- Hiding the top Image Controls toolbar no longer hides the thumbnail panel toggle
- Fixed UI bugs regarding disabled thumbnail tabs
- Annotations now cannot be copied and pasted onto a cropped document
- Annotation filtering and navigation works properly with Virtual Documents

- Postit Annotations now enforce minimum size on creation, instead of just on resize.

VirtualViewer 4.13 Release Notes

New Features

Save Default Choices for Document Dialogs

Users may now save custom default choices for the Save As, Export, Copy to New, Cut to New, Print, and Email dialogs.

For instance, a user's workflow may demand that all documents be exported as TIFFs. Previously, the user would have to find the Format section in the Export dialog and click the TIFF radio button for every export.

Now, the user can fill out the dialog with their preferred default choices and then click the button labeled Save Preferences in the bottom left of the dialog. When the user opens the Export dialog again, the form will be filled out with their saved defaults.

How to Use

To use this new feature, a user modifies the form choices in a dialog and saves those choices as the new default. For instance, they may choose to set defaults in the print dialog. The user opens the print dialog, and chooses the options to use going forward.

Clicking the Save Preferences button will save the user's choices. The dialog will still open normally, and the user may still change options normally. The options that are selected immediately on opening the dialog will now be the user's custom defaults.

Technical Details

Data will be stored in the browser's local storage, through the localforage library, so the preferences will persist across sessions of VirtualViewer on the same browser. Radio buttons and checkboxes will be stored; free text fields and page range fields will not have any defaults stored.

New Configuration Options

No configuration is necessary to enable this feature, but there are new configuration options to pre-set certain dialog defaults.

- `vvConfig.includeRedactions` The "Burn Redactions (Permanent)" checkbox will burn redactions into an image. If this configuration item is true, "Burn Redactions (Permanent)" will be checked by default.
- `vvConfig.includeRedactionTags` The "Include Redaction Tags" checkbox will write redaction tags onto the redactions on an image. If this configuration item is true, "Include Redaction Tags" will be checked by default.

- `vvConfig.includeDocumentNotes` The "Include Document Notes" checkbox will include the document notes in the exported, saved, printed, or copied document. If this configuration item is true, "Include Document Notes" will be checked by default.
- `vvConfig.includeWatermarks` The "Include Watermarks" checkbox will include added watermarks in the exported, saved, printed, or copied document. If this configuration item is true, "Include Watermarks" will be checked by default.

Updated Search

VirtualViewer now supports document searches and OCR on Virtual Documents, Sparse Documents, and compound documents. Search will also return correct results on documents whose pages have been manipulated and that have not yet been saved; previously, it would use the server version of a document, so could return results for a deleted page.

Pattern search is now supported on annotations. VirtualViewer may search annotation text, tags, and notes for social security numbers, telephone numbers, credit card numbers, and email addresses.

The user interface of the search tab has been updated with new button images, styles, and an adjusted layout.

The search API remains largely the same, with a new addition:

Unchanged API:

- `virtualViewer.cancelCurrentSearch()` stops the current search, and displays any already-returned results.
- `virtualViewer.clearSearchResults()` clears the current search, removing highlights from the document and thumbnails from the search panel.
- `virtualViewer.nextSearchResult()` advances the currently selected search result, switching pages if necessary.
- `virtualViewer.previousSearchResult()` moves the currently selected search result to the previous match, switching pages if necessary.
- `virtualViewer.isDocumentSearchable()` returns true if the document is searchable. It returns false if the document is not searchable
- `virtualViewer.searchText(searchTerm, firstPage, lastPage, skipOcrPrompt)` launches a search through the current document's text for the given search term. This search is performed on the server, and may perform OCR if the document has no text, OCR is enabled, and the user consents. A progress bar will appear when search is launched, as document search is performed asynchronously and in batches: a small batch of pages will be searched and a new batch sent to the server when the previous batch is returned.
    - `searchTerm` {String} The word or words to search for. Set case sensitivity in your

    configuration file.
  - `firstPage` {Number} Optionally define the start of a region of the document to search. This is 0-indexed, and the default is 0.
  - `lastPage` {Number} Optionally define the end of a region of the document to search. This is a 0-indexed, non-inclusive value. The default is the length of the document.
  - `skipOcrPrompt` {Boolean} If this parameter is set, document search will not prompt the user before using OCR, but will go ahead and use it if necessary and if OCR is enabled.
  - returns undefined

New API:

- `virtualViewer.searchAnnotationText(searchTerm)` launches a search through every annotation on the current document for the given search term. Searched annotation text includes annotation notes, text content, and tags. If no search term is provided and there is a search pattern currently selected in the search tab, a pattern search through the annotations will be launched.
  - `searchTerm` {String} The word or words to search for. Annotation search is case-insensitive.
  - Returns undefined

Configurable Highlight Colors for Search

Two new configuration parameters allow color customization for search. When a search is completed, all search results are highlighted in an orange color on the document; the current search result in focus is highlighted a light yellow.

The first option, `vvConfig.searchColors.matchColor`, sets the color for highlighting search results that are not in focus. The second, `vvConfig.searchColors.selectedMatchColor`, sets the color for highlighting the in-focus search result.

Both configurations may be set to a string that contains an rgba color, in the format `"rgba(255,78,0,0.2)"`. This is the default color for search-result highlights. The first three numbers are RGB values to establish the color, and the fourth number is an alpha value--treated like a percent--to define the transparency of the highlight.

New Callbacks

New callbacks have been provided to allow custom code to interact with VirtualViewer. In order to set a callback, call `virtualViewer.setCallback("callbackName", callbackFunction)`. This function returns `true` if the callback was set correctly, and `false` if it was not. The callback function should be defined, and should take a single argument object as a parameter. Then, for instance, if the function is declared as `function foo(args) { ... }`, the arguments are accessible in the callback function as `args.firstArgument`.

VirtualViewer is responsible for calling the provided callback function appropriately. For instance, VirtualViewer will attempt to call the function set to the `"switchToTab"` callback whenever a user switches their tab. Most callbacks do not pay attention to return values, but two new callbacks require a boolean return.

- `annotationChanged` is called whenever the user modifies an annotation; this will fire whenever VirtualViewer itself judges that an annotation has been changed and the asterisk appears in the tab name. The following parameters will be provided to the callback in the argument object:
    - `documentId` {String} The ID of the current document whose annotations have been modified
    - `annotationLayerId` {String} The ID of the layer that holds the modified annotation
    - `annotationId` {String} The ID of the modified annotation
- `disableTextContextMenu` is called when the text context menu is about to appear, and if the callback function returns `true`, the context menu will be disabled. This context menu can contain options to copy and cut text if any is selected on the document, to perform OCR, or to close document compare. The callback function will be provided one parameter in the arguments object, and must return a value:
    - `documentId` {String} The ID of the current document that the user is clicking on
    - Return `true` to disable the context menu, and return `false` to allow the context menu to show as normal
- `disablePageManipulationContextMenuOptions` is called when the page thumbnail context menu is shown. If the callback function returns `true`, page manipulation options will be removed from the context menu. Page manipulation options include cut, copy, and delete options; page insertion options; and page selection options. This function is equivalent to setting the configuration option `vvConfig.pageManipulations`, but allows document-by-document granular control. The callback function will be provided one parameter in the arguments object, and must return a value:
    - `documentId` {String} The ID of the current document whose pages the user is clicking on
    - Return `true` to remove the page manipulation options, and return `false` to allow VirtualViewer to show or hide the options as normal

Configuration to Disable User Preferences

Now, an administrator can completely disable User Preferences through a new option in vvConfig, `vvConfig.disableUserPreferences`. This configuration item can be set to `true` or `false`. If `true`, the User Preferences dialog will be unavailable to users. All configuration items that could be overridden in User Preferences will be drawn from vvConfig; users will not be able to override vvConfig settings. If not set or set to `false`, User Preferences will behave as normal.

Dynamic Debug Logging

To assist in debugging issues logging can now be toggled into a debug mode without having to change configuration files. Turning on dynamic logging can be done with the client-side call `virtualViewer.loggingOverride(true)` - while this flag is set all requests during that session will log all messages as high priority. This allows finely detailed logs to be created for a specific use case without changing global log configurations.

Simple Logging Facade (Java)

VirtualViewer Java now implements SLF4J (Simple Logging Facade for Java), a logging abstraction that allows clients to plug in the logging system of their choice. Documentation and examples can be found at https://www.slf4j.org/docs.html.

The default logger is still the java.util.logging framework. The init-param `logLevel` will only function for the default java.util.logging framework - if another logging framework is plugged in using SLF4J, that logging framework's configuration should be used instead.

Common Logging Facade (NET)

VirtualViewer NET now implements Common.Logging.NET, a logging abstraction that allows clients to plug in the logging system of their choice. Documentation and examples can be found at http://net-commons.github.io/common-logging/.

The default logging functionality is unchanged and is implemented in Common.Logging's configuration as SnowboundLoggerFactoryAdapter. The web.config parameters `logLevel` and `logToIIS` are also now implemented as arguments in Common.Logging's web.config section, although the original InitParam arguments will still work for the default logger. If another logging framework is plugged in using Common.Logging, that logging framework's configuration should be used instead - `logLevel` and `logToIIS` will only affect the default logger.

Fixes and Changes

Sticky note updates

The double-arrow button to minimize sticky notes will now scale with zoom. Previously, it was possible for the button to be drawn outside the bounds of the sticky note. Now, the button will no longer be larger than the area of the sticky note, and will disappear when the sticky note is zoomed out far enough.

Previously, on a zoomed-out document, it was possible for the size of a minimized sticky note to be larger than the full sticky note. Now, the minimized sticky note will scale properly as the document zooms.

enableOcr configuration fixed

The enableOcr configuration works again and now defaults to "true" (which will have no effect if your Snowbound license doesn't support OCR). Setting enableOcr to "false" will disable OCR even if your Snowbound license supports it. enableOcr was disabled in 4.12 and replaced with a simple license feature check for OCR.

PDF signature printing issue

Some PDFs have an issue with signatures disappearing when printing via VirtualViewer. We've modified our PDF.js printing solution to fix this issue. if you are encountering this problem, change config.js's `disableDirectPDFPrinting` to "true" to use our modified PDF.js instead of your browser's.

Misc. Fixes/Changes:

- Fixed issue with inserting annotations + disappearing layers
- Fixed client stack trace in sendDocument
- Added new config parameter, 'consolidateLayerName' to set the default name of a consolidation layer
- Prevent context menus from drawing off-screen when at the boundaries of the viewer
- Fixed document tab showing changes (with an Asterisk) when none were made
- Fixed bookmarks being lost during page manipulations

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).