ViewONE Pro Permanent Redaction Server Module

Version 1.1

Last Updated: 22nd March 2013

Email: info@daeja.com
Web site: http://www.daeja.com

Contents

Introduction *4*
Supported Platforms 5
Performance and Server Resources 6
Annotation burning *7*
Attachment Annotation burning *8*
Supported Annotation Types *8*
Installation *9*
Installing and using the Trial Permanent Redaction Servlet **9**
Apache Tomcat installation 9
Installation on other Web Application servers 10
Updating a Trial with a Licensed Update **10**
Additional Microsoft Windows Sparc Installation Steps **11**
Additional Sun Solaris Sparc Installation Steps **11**
Additional Linux Installation Steps **11**
Configuring the ViewONE Pro Permanent Redaction Server *12*
The Client (viewer parameters) **13**
annotationBurnURL 13
annotationDefaults 13
burnUseGZip 14
burnPDFToPDF 14
burnIncrementalAnnotations 14
burnMultiPage 14
showAnnotationBurnPrompt 15
receiveRedacted 15
annotationAutoBurnPromptOnClose 15
annotationAutoBurnOnClose 16
The Client (Viewer JavaScript Methods) **17**
burnAnnotations(prompt) 17
setBurnPDFToPDF(true/false) 17
The Client (Viewer JavaScript Events) **17**
The Servlet **18**
Configuring the ViewONE Pro Burner servlet *19*
logFile 19
detailedLogging 19
workingPath 19
tiffSaveColor 20
tiffSaveMonoG4 20
tiffSaveFormat 20
tiffJpegWriteQuality 20

The Burn Library **21**
Overview 21
ji.burn.jiBurner 22
setExtendedLogging 23
setIgnoreEmptyAnnotations 23
setIgnoreInvalidAnnotations 23
setBurnPDFToPDF 24
setBurnAnyFileFormat 24
setOutputFormat 24
setBurnColor 26
setHighMemoryTrigger 26
openFile 26
openPageArray 26
setAnnotationFile 27
closeDocument 27
burn 28
burnPage 28
burnSelected 28
setPage 29
getPage 29
setPageSelected 29
getPageSelected 29
clearPageSelection 30
getNumPages 30
setProperty 30
getProperty 31
sendCompletionOK 32
sendCompletionFailed 32
sendProgressText 32
burnAnnotations 33
burnAnnotations 34
burnAnnotations 36
burnAnnotationsFromIni 37
burnAnnotationsFromIni 38
mergePages 40
releaseResources 40
XML Checksum 41
ji.burn.jiBurnerListener 41
burnUpdate 41

Troubleshooting ***42***
Clearing temporary working files 42
Logging 42
Internal server error 42
java.lang.InternalError 42
Redacting Text 42
Single Redaction covers Shared Resources 43
java.lang.NoSuchMethodError: ji.burn.jiBurner.initTrace 43
Non-burnable redactions get burned 43

Introduction

ViewONE Pro Permanent Redaction Server Module (PRS) allows users to apply permanent mark-ups to any of the ViewONE supported documents. Redaction markups will remove and block out any part of a page. Selecting the area to redact is done easily (just like adding annotations) so the user can concentrate on the removal of potentially sensitive information contained in the pages of a document.

The PRS module will produce a TIF document containing the redacted images or, if redacting a PDF document[1], it will produce the same PDF document but with the redacted areas completely removed. The resulting PDF document will be searchable in precisely the same manner as the original. Unlike other redaction systems, Daeja's PRS module retains the original PDF format and so does not convert them to image based PDF's.

Because the data underneath the redaction is permanently removed from the file, the new documents are safe to distribute.

The PRS module is ideal therefore where high security is required and in applications where compliance dictates that, specifically where redaction features are offered, such redactions are un-moveable and non-removable.

Recent directives for data storage and archiving of records have resulted in a requirement for redactions applied by users and administrators to become records themselves, retaining the redaction as part of the document's image data, in effect 'burning' the data into the image. ViewONE Pro's Permanent Redactions Server module satisfies this requirement.

The PRS module is a Java server component which requires the ViewONE Pro Annotations Module and is flexible enough to allow integrations into any document back end system.

The PRS module is a Java servlet. Daeja supplies free sample servlet code or you can write your own servlet using the Java Burn Library interface.

In addition, for batch or automated processing, it is also possible to use the Burn Library interface on its own without a client or servlet, if so desired, to produce redacted documents on demand using your own template scheme.

[1] Note: Password protected PDFs are not currently supported.

Supported Platforms

Redaction of TIFF input files to TIFF output files is platform independent.

Redaction of PDF files to output PDF files is currently available for Microsoft Windows based servers, Sun Solaris/Sparc based servers and Linux 2.6 kernel (glibc 6) x86 based servers.

The following table shows the complete set of supported output formats for each of the supported input formats with respect to the server platform backend.

Table 1: Supported output formats for each supported input format, by server platform

Input format	*Windows 32 bit JRE*	*Windows 64 bit JRE*	*Solaris*	*Linux*	*AIX*	*HP-UX*
TIFF	TIFF	TIFF	TIFF	TIFF	TIFF	TIFF
JPEG	TIFF	TIFF	TIFF	TIFF	TIFF	TIFF
GIF	TIFF	TIFF	TIFF	TIFF	TIFF	TIFF
BMP	TIFF	TIFF	TIFF	TIFF	TIFF	TIFF
PNG	TIFF	TIFF	TIFF	TIFF	TIFF	TIFF
PDF	TIFF, Searchable PDF	Searchable PDF	Searchable PDF	Searchable PDF	Searchable PDF	-
Microsoft Office formats	TIFF*	TIFF*	TIFF*	TIFF*	TIFF*	TIFF*
Openoffice.org formats	-	-	-	-	-	-
Email (.eml, .msg)	-	-	-	-	-	-

A hyphen (-) means that the input format cannot be redacted on that particular server backend.

* This is only available with office documents loaded through the Office module

Performance and Server Resources

Predictions for runtime resource usage during document redaction vary significantly based on the types of documents that are being processed. For example, full color TIFF images require more memory, CPU and disk resource than mono TIFF images. The application of several hundred redactions to a PDF file requires more memory and CPU usage than the application of a few redactions.

The Permanent Redaction Server Module requires a minimum of 20 MB of memory for a single user. Per additional concurrent user, the memory requirement increases. The increase depends upon the type of documents that are being redacted. Typically, black and white TIFF documents incur a 500 KB or more per-user increase, a full color TIFF document incurs a 2 MB or more per-user increase and a PDF document could incur, in a worst-case scenario, up to two times the size of the PDF document per-user increase. Real-world usage figures are likely to vary between, and occasionally greater than these values.

Disk space requirements are related to the size and color depth of the documents that are being redacted and cannot be accurately predicted. Additional configuration options are documented later on in this manual

Annotation burning

ViewONE has the facility for providing the user with the ability to annotate documents with various annotation types. Some examples of these types include text, stamps, highlights and now redactions.

ViewONE offers storage of this annotation data separately from the document. This has many benefits but is not always the required solution (for more information on annotations, please refer to the ViewONE Annotations Manual). Image burning allows the user to permanently make changes to a document by creating a new TIFF Group 4 image. Such changes might include redacting sensitive information or applying a received stamp to a document. ViewONE meets this requirement in this burn module.

The functionality requires a server side component which does the actual burning and will require integration with a document management system that wishes to use it. Daeja supplies a sample Java Servlet which demonstrates how the process works. The sample uses the Daeja-supplied Java class library, ji.jar (from a ViewONE install), which provides the facilities which enable the server process to perform its task.

Use of this functionality requires a special license file that should be used with the server-based install of ViewONE so that the burn functionality becomes available. This license file is available from Daeja Image Systems.

Daeja took the approach of performing the actual burning on the server, as opposed to client-side, in order to minimize network traffic. If burning were performed client-side then the newly burned image would have to be sent back to the server each time a burn operation is performed thus causing excessive network traffic and unnecessary delays.

Instead, the ViewONE client sends only the minimum amount of data to the server so as to permit the server to apply the necessary annotations/redactions and for it to retrieve for itself the required images prior to burning.

Once the burn process is complete, the servlet can then store the newly created TIFF images in your repository. To that end we have chosen to make the server side burn feature available as a Java library and to supply sample servlet source code that you are free to edit to integrate into your specific repository/back-end system.

Note that to further minimize network traffic, once the burn process is complete, the viewer simply disables editing of the burned annotations rather than reloading the newly created TIFF image. By taking this approach there is no need to re-download the new image for viewing.

The viewer provides feedback to the user whilst the server burn process is executing.

Attachment Annotation burning

From version 1.1.172 the view is also capable of showing and annotating email attachments. The burning of annotated attachments does not alter the original email, but creates a separate TIFF, or PDF. Burning redactions into the original email document is not supported.

Supported Annotation Types

It is important to note that not all annotation types are supported by the PRS module and whether they are supported or not also depends on the required output format of the file. The complete list of supported types is as follows:

Output File → TIFF format

Arrow
Freehand
Line
Oval
Open Polygon
Redaction
Redaction Polygon
Text
Text Stamp
Image Stamp **(Note: it is important that the path entered for the location of the image stamp is accessible by the burner as well as the client, else stamp will not be available to burner and will not appear in burnt document)**

Additional annotation support added at release 3.1.176:
Highlight **(Note: only suitable for color output formats)**
Highlight Polygon **(Note: only suitable for color output formats)**

Output File → PDF format

Redaction
Rectangle

Further annotation types may be added as and when they become available.

Installation

There are two possible downloads for the PRS, a free trial (v1pro_permredact.zip) and a full version update (v31update_server.zip).

The trial download of the PRS can be downloaded for free from http://www.daeja.com/trial/. It is a zip file called v1pro_permredact.zip. This zip contains everything you need to trial the Permanent Redaction Functionality.

The full version update must be purchased. It provides the actual PRS that can be used in a production environment that must be properly licensed.

Installing and using the Trial Permanent Redaction Servlet

The trial is supplied as a standard web application directory that can be deployed into the web application server of your choice. It includes an example implementation of a servlet called `daeja.burner`. The code for this servlet is included and can be edited freely and used within your production environment.

Apache Tomcat installation

1. Install Apache Tomcat 5.5 (http://tomcat.apache.org/ - if you are using a Windows OS, the EXE download is recommended)

2. Navigate to the Tomcat webapps directory (eg. C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps)

3. Copy the "viewone" folder supplied as part of the Permanent Redaction distribution to the webapps directory described in the previous step.

4. Restart Tomcat (use Start->Programs->Apache Tomcat 5.5->Configure Tomcat to stop and re-start the Tomcat service)

5. Launch your web browser

6. Navigate to http://localhost:8080/viewone/index.htm

7. Navigate to ViewONE Pro – Permanent Redaction Server Module Demo

8. Select one fo the demos, draw a redaction press the "burn" button

.

Installation on other Web Application servers

There are no specific instructions for installing the PRS burner servlet on other web application servers such as WebLogic, JBoss, WebSphere, etc. The instructions provided above are simply a sample installation using the free Apache Tomcat web application server to get you up and running quickly with the free trial - they are not a definitive guide to installing the burner across all the different web application servers that are available.

The reason for this is that the PRS module is a servlet that needs to be installed on a web application server in order for it to work - but this can be ANY web application server of your choosing - and most customers will already have a preferred web app server they use and know about. Daeja are not (and cannot be) experts in every single web application server out there and so cannot provide specific instructions on how to deploy a servlet onto every single known web app server either.

So the minimum requirement for setting up the PRS server side component is that you have a web application server in place that you are familiar with and can deploy servlets on, gather logs files, etc so that you should be able to deploy the servlet and get to the point where you can enter the URL to the burner and see that message:

“Burner Servlet just ran successfully (but without any parameters and did nothing but test HTTP)”

Updating a Trial with a Licensed Update

1. Firstly, ensure that your web application server has been shutdown correctly.
2. Locate the WEB-INF/lib directory in your target web application server.
3. Copy the jiserver.jar file found in the downloaded WEB-INF/lib directory to the WEB-INF/lib directory in your target web application server.
4. Ensure that your license file (lic-server.v1) is present in the web application server's WEB-INF directory.
5. Restart your web application server and check that the servlet is running correctly. In a default installation, this can be done by calling http://localhost:8080/burner/burner within your web browser. Check that the logging correctly reports the upgraded version number.
6. Please note these instructions do not upgrade the client ViewONE Viewer. To upgrade the client, you should update the v1files directory in the normal way.

Additional Microsoft Windows Sparc Installation Steps

In addition to the jiserver.jar file, on Microsoft Windows an additional jar file is needed. This jar file is provided by default in the trial download. When the full version is being updated, this additional jar file (jiserver-resources-win32.jar) can be downloaded from the same place.

To install the additional Sun Solaris resource jar, locate the jiserver-resources-win32.jar in the zip file downloaded (v31update_server_resources_win32.zip). Place the jar file in the classpath. When using the Permanent Redaction Server in a servlet the classpath location should be the WEB-INF/lib directory of the servlet.

Additional Sun Solaris Sparc Installation Steps

In addition to the jiserver.jar file, on Sun Solaris with the Sun Sparc architecture an additional jar file is needed. This jar file is provided by default in the trial download. When the full version is being updated, this additional jar file (jiserver-resources-sunos.jar) can be downloaded from the same place.

To install the additional Sun Solaris resource jar, locate the jiserver-resources-sunos.jar in the zip file downloaded (v31update_server_resources_sunos.zip). Place the jar file in the classpath. When using the Permanent Redaction Server in a servlet the classpath location should be the WEB-INF/lib directory of the servlet.

Additional Linux Installation Steps

In addition to the jiserver.jar file, on Linux 2.6 with the x86 32-Bit architecture and x86 64-Bit architecture and Linux 2.4 with the x86 32-Bit architecture an additional jar file is needed. This jar file is provided by default in the trial download. When the full version is being updated, this additional jar file (jiserver-resources-linux.jar) can be downloaded from the same place.

To install the additional Linux resource jar, locate the jiserver-resources-linux.jar in the zip file downloaded (v31update_server_resources_linux.zip). Place the jar file in the classpath. When using the Permanent Redaction Server in a servlet the classpath location should be the WEB-INF/lib directory of the servlet.

Configuring the ViewONE Pro Permanent Redaction Server

The ViewONE Pro Permanent Redaction Server is supplied as a library with a set of methods on a Java class that perform the actual process of permanently redacting the image.

The example servlet that is supplied in the trial download shows how to integrate the redaction functionality with the ViewONE Viewer Applet. This servlet is provided without warranty and can be edited freely and used within your production system.

The code demonstrates how redaction requests are received from the viewer, which then prompts retrieval (by the servlet) of the document being displayed by the viewer (using the same URL used by the viewer), and how redactions are then applied.

To integrate this into your own production environment you will need to insert your own code to retrieve the document from your back-end store and if desired, storing of the redacted document into your back-end store.

Please be aware that in a multi-threaded environment, for example a servlet, each thread should use a different instance of the `ji.burn.jiBurner` class.

The rest of this document describes all the configuration and burn options available.

Parameter: **annotationBurnURL**

(Pro-Only) <PARAM NAME="annotationBurnURL" value="[URL of server burning process]">

This parameter specifies the URL of the burn servlet.

For example:

<PARAM NAME="annotationBurnURL" value="http://myserver/burner">

Note: When annotationBurnURL is defined the viewer will automatically display an additional 'burn' button within the annotation tool bar. When the user clicks on that button the burn process will begin.

Note: In order for burning to process you must always specify the "annotationDefault" paramater as well (defined below).

Note: For performance reasons, this will not burn the annotations into an image on the client-side. Instead, it will burn the annotations into an image on the server-side and mark the annotations on the client-side as un-editable.

Parameter: **annotationDefaults**

(Pro-Only) <PARAM NAME="annotationDefaults" value="[type {burnable=[burnable state]}]">

This parameter is used to specify the annotation types that are to be treated by the viewer as burnable, rather than, as is usual, a layer. The parameter works by taking a tag for a type identifier, followed by a property (in this case, whether the type is burnable or not) and its value.

For example:

<PARAM NAME="annotationDefaults" value="redact {burnable=true} text{burnable=true} line{burnable=true} arrow{burnable=true}">

The tags that can be used for type identifiers are listed in the 'Annotation types' section of the ViewONE Annotations Manual. The 'Definition:' given for each annotation type listed there should be the value used in this parameter – note that the list is space seperated

For burning, the only property that can be specified is the 'burnable' property. It can be specified as either 'true' or 'false'. True indicates that it is to be treated as burnable. The default for all types is 'false'.

Note, as of ViewONE Pro build 1.1.24, only the redaction type is supported as burnable when the output document is a PDF. When the output document is a TIFF, then the following types are supported, rectangle, oval, line, arrow, open polygon, text, text stamp, freehand, redact and redact polygon. Other types may be added in due course.

Note that this parameter can be used in conjunction with the setRedactionIsSemiTransparent() Javascript method to specify the set of annotation types which will be made semi transparent when the method is set to true.

Parameter: **burnUseGZip**

(Pro-Only) <PARAM NAME="burnUseGZip" value="[true | false]">

This parameter specifies whether the annotation data sent from ViewONE Pro to the burn Servlet is compressed. When set to "true", the "Content-Encoding" type of the data will be "gzip". When this parameter is used, the servlet **must** decompress the annotation data before passing it to one of the *burnAnnotations* methods on the *jiBurner* object.

Parameter: **burnPDFToPDF**

(Pro-Only) <PARAM NAME="burnPDFToPDF" value="[true | false]">

This parameter specifies whether the viewer should indicate to the server side burner process that if the source document is a PDF document, the burnt output document should also be a PDF document.

Note that when the output document is a PDF, the supported annotation types are limited, as described on page 8 (Supported Annotation Types). PDF to PDF burning is also currently supported on the platforms listed on page 5 (supported platforms).

The default for this parameter is "false".

Parameter: **burnIncrementalAnnotations**

(Pro-Only) <PARAM NAME="burnIncrementalAnnotations" value="[true | false]">

This parameter specifies whether the annotation data sent from ViewONE Pro to the burn Servlet contains changes since the last burn operation or all the changes since opening the document. By default, this parameter is set to "true".

Parameter: **burnMultiPage**

(Pro-Only) <PARAM NAME="burnMultiPage" value="[true | false]">

This parameter specifies whether the burn Servlet should create a single multi-page document or individually burnt single pages. Annotation data sent from ViewONE Pro to the burn Servlet contains changes since the last burn operation or all the changes since opening the document. By default, this parameter is set to “true”.

When this parameter is set to “true”, there is a “merge” attribute set to the value of “true” on the “pages” tag in the annotation data sent to the servlet. This can then be used to interpret which burn methods to use.

Parameter: **showAnnotationBurnPrompt**

(Pro-Only) <PARAM NAME=“showAnnotationBurnPrompt” value=“[true | false]”>

(Version Pro 1.1.106+)

This parameter specifies whether when the user presses the burn button, they get prompted as to whether they want to continue.

The default value for this parameter is ‘true’.

Parameter: **receiveRedacted**

(Pro-Only) <PARAM NAME=“receiveRedacted” value=“[true | false]”>

(Version Pro 1.1.106+)

This parameter specifies where the redacted output file gets directed. When set to "true", after the user presses the burn button and the burn has completed, they will get prompted as to whether they want to open or save the file locally. When set to "false", the user wil receive no output file and it is down to the configuration of the burner to store the output file in the backend repository.

The default value for this parameter is ‘true’.

Parameter: **annotationAutoBurnPromptOnClose**

(Pro-Only) <PARAM NAME=“annotationAutoBurnPromptOnClose” value=“[true | false]”>

(Version Pro 4.0.26+)

This parameter specifies whether a user is prompted or not if they attempt to leave the page without having burned annotations.

The default value for this parameter is ‘true’.

Parameter: **annotationAutoBurnOnClose**

(Pro-Only) <PARAM NAME="annotationAutoBurnOnClose" value="[true | false]">

(Version Pro 4.0.26+)

This parameter specifies whether or not any "non-burned" annotations will be automatically burned if the user leaves the page without having burned annotations.

The default value for this parameter is 'false'.

The Client (Viewer JavaScript Methods)

Method:

burnAnnotations(prompt)

(Pro-Only)

(Version Pro 1.1.96+)

This JavaScript method activates the burn process with whatever server-side permanent redaction object that has been defined. It calls the same functionality that pressing the "Burn" button calls.

The **prompt** argument defines whether the user should be prompted to accept or reject the burn before it is activated. The values are "true" and "false".

Method:

setBurnPDFToPDF(true/false)

(Pro-Only)

(Version Pro 1.1.172+)

This JavaScript method replicates the behavior of the html parameter **burnPDFToPDF**. When a burn request is sent from the client to the server side component, the value set by this Javascript method indicates if the source document is a PDF document, the burnt output document should also be a PDF document.

Please see the html parameter **burnPDFToPDF** for more details.

The Client (Viewer JavaScript Events)

The following list describes the events that will be received related to Permanent Redaction. Please see the ViewONE Javascript API Manual on how to use these events.

Id	Event Text	Description
48	Annotations burnt	This event is generated when annotation burning on the server has finished and the client is notified.
51	Annotation burn cancelled by user (Version 1.1.106 onwards)	This event is generated when the burn has been cancelled by the user when prompted.
52	Annotation burn failed (Version 1.1.106 onwards)	This event is generated when the client is notified that the annotation burning has failed.

The Servlet

The server-side part of the process takes the form of a servlet that uses the ji.burn library. This library is made available by use of the jiserver.jar and through a full ViewONE viewer deployment in combination with the server and client side lic.v1 license files.

The classes needed to implement the burn library can be found in the ji.burn package (found within jiserver.jar). To use that package the jiserver.jar needs to be included as a library within the chosen IDE.

In addition, when a servlet using that package is deployed, that same jiserver.jar needs to be copied into the correct library location (normally “WEB-INF/lib”). The specific mechanism for installing a servlet and dependent libraries on your server may be dependent on whatever Application Server you have chosen to use (and this information should be available in its corresponding documentation).

Ensure that your license file (lic-server.v1) is present in the web application server's WEB-INF directory. If you have a server-side license file for the ViewONE Pro Permanent Redaction Server that is named lic.v1, please rename it to "lic-server.v1".

An example of such a servlet is supplied with the downloaded zip file under the \batch_example folder that you can make use of. The servlet can be configured using parameters in the burner.properties configuration file.

A sample Burner configuration file is supplied inside the Burner demo package, named `burner.properties`. This is located in the `burner\WEB-INF` directory. You willneed to either copy this sample file to, or create a new file in, your target web application’s `WEB-INF` directory. This file can be created and edited in a plain text editor.

The format of the file is that of the standard Java properties file format (specifically `key=value` pairs). Comments are also allowed, but each line of a comment should be prefixed with a `#` sign. Blank lines are also acceptable.

If required, the default location of the `burner.properties` file can be changed by modifying the `web.xml` file which is also in the `WEB-INF` directory. You will find inside the `web.xml` file an `<init-param>` called `properties`. The value of this initialization parameter can either take the form of an absolute path such as `c:\burnerdemo\burner.properties` or (as is the default) a path that is relative from the burner web application root directory such as `WEB-INF/burner.properties`

Configuring the ViewONE Pro Burner servlet

What follows is a description of each parameter and how it affects the Burner's operation.

Parameter: **logFile**

```
logFile=c:\\burner\\burner.log
```

```
logFile=logs\\burner.log
```

In the case of a filename, logging information that is output during Burner operation will be stored in the specified file. If using the special identifier, logging will be directed to standard out which in most cases means it will appear in the web application server's log files.

If left blank, then logging will be disabled. Please note that any backslashes (\) should be entered as a double backslash. It can be in the form of an absolute path or relative. If it is specified as relative, then it will be relative to the root of the web application (the deployed burner directory).

The default value of this parameter is un-set (so no logging will occur).

Parameter: **detailedLogging**

```
detailedLogging=false
```

When set to true, this parameter will cause additional logging to be output to the log file. It is not recommended that this parameter should be set to true in production environments. It is a useful tool for troubleshooting errors.

Valid values are: true or false. The default is false

Parameter: **workingPath**

```
workingPath=c:\\burnerdemo\\temp
```

```
workingPath=WEB-INF\\working\\temp
```

This parameter should point to a writable directory in which temporary working files will be placed. Please note that any backslashes (\) should be entered as a double backslash. It can be in the form of an absolute path or relative. If it is specified as relative, then it will be relative to the root of the web application (the deployed streamer directory).

The default value is a subdirectory of the private temporary directory in the servlet context.

Parameter: **tiffSaveColor**

Where this parameter is set to true then all TIFF files will be output in the LZW compressed format which supports color. They will ***always*** be output in the LZW format even if there is no color present in the image/annotations ***unless*** the tiffSaveMonoG4 parameter is set to true (see below)

Valid values are: true or false. The default is false

Parameter: **tiffSaveMonoG4**

This parameter is only relevant where the tiffSaveColor parameter is also set to true (see above). When set to true, this parameter is used to prevent a monochrome TIFF file from being unnecessarily saved in the color LZW compressed format. So where the burner servlet detects color, it will still output in the color-compliant LZW format but if there is no color present in the image or any of the burnt annotations and this parameter is set to true then the TIFF file will be output in the TIFF Group 4 compression (mono) format.

Valid values are: true or false. The default is false

Parameter: **tiffSaveFormat**

Available from version 4.0.32+onwards, this parameter is only relevant where the tiffSaveColor parameter is also set to true. It allows you to specify whether redacted TIFF files will be generated using LZW or JPEG compression.

Valid values are: lzw or jpeg. The default is lzw.

Parameter: **tiffJpegWriteQuality**

Available from version 4.0.32+onwards, this parameter is only relevant where the tiffSaveFormat parameter is set to “jpeg” (see above). So when outputting a TIFF embedded JPEG format, this parameter is used to control the level of quality of the image in a range of 1-100 where 1 is worst, 100 is best.

Note that with a higher settings for quality value, the redaction servlet will take longer to generate the image – so there is a trade-off between quality and speed of rendering to consider here and this is entirely dependant on the system being sued and level of quality required base do the particular image(s) being generated.

The default is 80.

The Burn Library

This can be used for burning annotation data into images. The servlet (as described previously in this manual) uses this library to perform its functions.

Overview

Correct usage of the library is as follows:

- ji.burn.jiBurner instantiated
- <any number of API method calls>
- releaseResources() must be called

It is important to note that releaseResources() must be called when you have finished using the jiBurner object. This allows any remaining resources to proceed and any remaining threads to finish execution. Once releaseResources() has been called on a particular instance of a jiBurner, no further method calls should be made to that instance as they will result in undefined behaviour.

When the servlet receives a burn request from the ViewONE Pro, it must retrieve the following properties from the request:

- *Content-Encoding* – If ViewONE Pro has been set up with the html parameter "burnUseGZip" set to "true" then the annotation data passed to the servlet in the request will be compressed and the content-encoding will be set to "*gzip*". The servlet must then decompress the annotation data. This can be done with the Java InputStream class, java.util.zip.GZIPInputStream.
- *X-Client-BurnPDFToPDF* – If ViewONE Pro has been set up with the html parameter "burnPDFToPDF" set to "true" then it is asking the servlet to make the method call *jiBurner.setBurnPDFToPDF(true)*.

Class: **ji.burn.jiBurner**

An instance of jiBurner must be created to use the burn library. The object's constructors look like this…

jiBurner(String *strResourcePath*,
String *strOutputPath*,
jiBurnListener *listener*,
PrintWriter *logger*) throws *FileNotFoundException*

- *strResourcePath* defines the default path for the directory in which the license file is located. This argument can be set to **null** if the license file is located in a jar on the class-path.

- *strOutputPath* is the path the library will use for both its temporary files and the files it generates as a result of the burning process. If this argument is set to **null**, a subdirectory ("viewone-server") will be created and used under the directory defined by the system property "java.io.tmpdir".

- *listener* an optional burnListener for getting feedback on the burn process. This is only required if the servlet needs the library to report back progress on the burn process.

- *logger* an optional PrintWriter for troubleshooting the burn process. This is only required if problems are being experienced and the servlet needs the library to log what it is doing in order to troubleshoot them.

- Reasons for the constructor throwing a *FileNotFoundException* are as follows…

  - If an invalid strResourcePath parameter is passed in, i.e., one that cannot be found or is not readable.

  - If an invalid strOutputPath parameter is passed in, i.e., one that cannot be created or is not writable.

jiBurner(String *strResourcePath*,
String *strOutputPath*,
String *stLicensePath*,
jiBurnListener *listener*,
PrintWriter *logger*) throws *FileNotFoundException*

- *strResourcePath* defines the default path for the directory in which the license file is located. This argument can be set to **null** if the license file is located in a jar on the class-path or the *strLicensePath* argument is defined.

- *strOutputPath* is the path the library will use for both its temporary files and the files it generates as a result of the burning process. If this argument is set to **null**, a subdirectory ("viewone-server") will be created and used under the directory defined by the system property "java.io.tmpdir".

- *strLicensePath* defines the path for the directory in which the license file is located. This argument should be set to **null** if the license file is located in a jar on the class-path.

- *listener* an optional burnListener for getting feedback on the burn process. This is only required if the servlet needs the library to report back progress on the burn process.

- *logger* an optional PrintWriter for troubleshooting the burn process. This is only required if problems are being experienced and the servlet needs the library to log what it is doing in order to troubleshoot them.

- Reasons for the constructor throwing a *FileNotFoundException* are as follows…

  - If an invalid strResourcePath parameter is passed in, i.e., one that cannot be found or is not readable.

  - If an invalid strOutputPath parameter is passed in, i.e., one that cannot be created or is not writable.

Once the burner has been instantiated several methods can be called to set it up. They are as follows…

Method: **setExtendedLogging**

A flag setting method telling the library whether or not to output extended logging. The method looks like this…

void setExtendedLogging(boolean *boolLog*)

Method: **setIgnoreEmptyAnnotations**

A flag setting method telling the library whether or not to still burn (create) a page of a document even when there are no annotations on it. The method looks like this…

void setIgnoreEmptyAnnotations(boolean *boolIgnoreEmpty*)

Method: **setIgnoreInvalidAnnotations**

A flag setting method telling the library whether or not to still burn (create) a page of a document even when there are annotations on it that cannot be burnt. For example, when burning PDF to PDF the redact annotation type is the only valid type. The method looks like this…

void setIgnoreInvalidAnnotations(boolean *boolIgnoreInvalid*)

Method:

setBurnPDFToPDF

A flag setting method telling the library that when the source document is a PDF document, the burnt output document should also be a PDF document. The method looks like this…

void setBurnPDFToPDF(boolean *outputPDF*)

Method:

setBurnAnyFileFormat

A flag setting method telling the library whether or not to allow any supported image format to be the source format. The only possible output formats are PDF or TIFF when the source is a PDF document and TIFF for all other formats. The method looks like this…

void setBurnAnyFileFormat(boolean *anyFormat*)

Method:

setOutputFormat

A method for telling the library which output format to use when saving redacted images to disc.

void setOutputFormat(String *format*, Properties *params*) throws ji.filter.output.jiUnknownImageWriterException

- *format* is a string containing the format name. Formats currently supported are TIFF (G4/LZW/JPEG), JPEG and PNG. All output formats with the exception of TIFF G4 require a Java 2 runtime. JPEG and PNG output requires at least a Java 1.4 runtime.

  Accepted names may be of a filename extension or mime-type, for example:

  tiff
  jpeg
  png
  image/tiff
  image/png

- *params* is a set of properties that are passed to the image writer in order to set up particular configuration items. Simply add required configuration parameters to the *params* Properties object. Eg:

```
Properties props = new Properties();

props.setProperty("tiffSaveFormat", "jpeg");

props.setProperty("tiffJPEGWriteQuality", "80");
```

- A reason for the method throwing a *iiUnknownImageWriterException* would be if a request was made for an unknown output format..

Image format	Supported configuration
TIFF	tiffVersion By default, all TIFFs are written to the TIFF 7 standard. Some viewers are unable to cope with this standard so the option to override this default is provided. Valid values are: 6 tiffSaveFormat All TIFFs can be saved in a number of different formats: lzw – this will write TIFFs using LZW compression g4 – this will use TIFF Group 4 compression (mono) jpeg – this will use JPEG compression tiffMonoSaveFormat Mono TIFFs can be saved in two different formats: lzw – this will write TIFFs using LZW compression g4 – this will use TIFF Group 4 compression (mono only) tiffJPEGWriteQuality This configuration is valid when “jpeg” is set as the “tiffSaveFormat”. It should be provided as an integer between 1 and 100.
JPEG	NONE
PNG	NONE

Method:

setBurnColor

(Version Pro 4.0.2+)

A flag setting method telling the library whether or not to output color format for the redacted document. Where this parameter is set to true then all TIFF files will be output in the LZW compressed format which supports color. The method looks like this:

void setBurnColor(boolean *boolColor*)

Method:

setHighMemoryTrigger

(Version Pro 1.1.98+)

A flag used to set the amount of memory in bytes the burn process will use when creating a non-PDF image. If the burn process needs more memory than the specified amount then a swap file will be used. The default value for this property is 102400. Any attempt to set a value smaller that value will reset the value to the default.

void setHighMemoryTrigger(long memInBytes)

- *memInBytes* - memory in bytes to be used by the burn process

Method:

openFile

(Version Pro 1.1.106+)

Specifies the filename and initial page of the document to be loaded.

This filename can specify either a filename or a full URL.

void openFile(String filename, int initialPage) throws *Exception*

- *filename* - The filename of the document to open.
- *initialPage* – The initial page number. This is the equivalent of calling *setPage(initialPage)*.
- *Exception* – An exception can be thrown if there is a problem opening the document.

Method:

openPageArray

(Version Pro 1.1.106+)

This method will open multiple files as a single document.

Index 0 in the array will be page 1, index 1 will be page 2 and so on. If one of the filenames refers to a multi-page document, the first page will be the one used as the target page. To specify a page other than 1, use the # notation. For example, the following filename examples will reference page 3:

C:\multipage.tif#3

http://host/multipage.tif#3

This filenames can specify either a filename or a full URL.

void openPageArray(String[] filenames, int initialPage) throws *Exception*

- *filenames[]* – An array of filenames to open as a single document.
- *initialPage* – The initial page number. This is the equivalent of calling *setPage(initialPage)*.
- *Exception* – An exception can be thrown if there is a problem opening the document.

Method: **setAnnotationFile**
(Version Pro 1.1.106+)

This method sets the annotation file and must be called before any of the open methods described above.

If the current document is closed after this method has been called, the annotation file will be reset.

This filename can specify either a filename or a full URL.

void setAnnotationFile(String filename)

- *filename* - The filename of the annotation file.

Method: **closeDocument**
(Version Pro 1.1.106+)

Closes an open document.

NOTE: Closing the open document also resets annotations.

void closeDocument()

Method:

burn

(Version Pro 1.1.106+)

This method will burn all pages of the currently open document to the specified filename.

NOTE: Burning may fail if the input document has multiple pages and the selected output format does not support multi-page images.

void burn(String filename) throws *jiBurnFailedException*

- *filename* - The filename to save to.
- *jiBurnFailedException* – An exception can be thrown if there is a problem whilst burning the document.

Method:

burnPage

(Version Pro 1.1.106+)

This method will burn the current page of the currently open document to the specified filename.

Please see the *setPage(int pageNumber)* and *getPage()* methods for details on how to set a particular page.

void burnPage(String filename) throws *jiBurnFailedException*

- *filename* - The filename to save to.
- *jiBurnFailedException* – An exception can be thrown if there is a problem whilst burning the document.

Method:

burnSelected

(Version Pro 1.1.108+)

This method will burn all the pages in the current selection of the currently open document to the specified filename. The resulting document will consist of only the pages specified by the selection.

Please see the *setSelectedPage(int pageNumber, boolean state)* and *getSelectedPage (int pageNumber)* methods for details on how to manipulate page selections.

void burnSelected(String filename) throws *jiBurnFailedException*

- *filename* - The filename to save to.
- *jiBurnFailedException* – An exception can be thrown if there is a problem whilst burning the document.

Method:

setPage
(Version Pro 1.1.106+)

This method will set the current page of the currently open document.

void setPage(int page)

- *page* - The page number to set.

Method:

getPage
(Version Pro 1.1.108+)

Gets the current page of the currently open document.

int getPage()

- The int returned indicates the current page.

Method:

setPageSelected
(Version Pro 1.1.108+)

Sets the page selection status of the page in the currently open document indicated by the "pageNumber" parameter (available for multi-page documents only).

void setPageSelected(int pageNumber, boolean state)

- *pageNumber* - The page number to select or de-select.
- *state* – When set to true, selects the page. False will de-select the page.

Method:

getPageSelected
(Version Pro 1.1.108+)

Gets the selection status of the page in the currently open document indicated by the "pageNumber" parameter (available for multi-page documents only).

boolean getPageSelected(int pageNumber)

- *pageNumber* - The page number to retrieve the details of.
- The boolean returned indicates the page selection status (True if selected, false if not).

Method:

clearPageSelection

(Version Pro 1.1.108+)

This method clears the current page selection.

void clearPageSelection()

Method:

getNumPages

(Version Pro 1.1.106+)

This method will return the number of pages in the currently open document.

int getNumPages()

- *page* - The page number to set.
- The integer returned represents the number of pages in the currently open document.

Method:

setProperty

(Version Pro 1.1.118+)

A method used to set configuration options on the burner. It takes a property name and a property value. For properties that take a true/false values, the following values are acceptable to mean true, "1", "true" or "yes". All other possible values mean false, including an empty string and a null value.

The possible property names that this method can use are a subset of the properties that can be set as html parameters on the ViewONE applet. Please see the html.pdf for information on the properties.

Currently the supported string properties are:
"**textfilterMimetypes**"
"**textfilterLocalFileExtensions**"

Currently the supported boolean properties are:
"**convertToTIFFOnSave**" – can be used instead of method "**setBurnAnyFileFormat**"
"**burnPDFToPDF**" – can be used instead of method "**setBurnPDFToPDF**"

Returns true if the supplied name is of a supported property and the value could be set. Returns false if the name is of a non-supported property.

boolean setProperty(String propertyName, String value)

Method:

getProperty

(Version Pro 1.1.118+)

A method used to retrieve configuration options values set on the burner. It takes a property name and returns a property value.

For boolean property values, the strings “true” and “false” are returned. For properties that are currently not supported by the method a null value is returned. For currently supported properties see the “**setProperty**” method.

String getProperty(String propertyName)

The jiBurner class provides several static methods to facilitate communications with the client viewer during the burn process. They are as follows…

Method: **sendCompletionOK**

A static method to allow a servlet to send a 'complete' message with an OK status to the client. The method looks like this…

void sendCompletionOK(OutputStream *response.getOutputStream()*)
throws *IOException*

- *response* is the object for the server's POST response.
- A reason for the method throwing an *IOException* would be if there was an error when writing to the response object's OutputStream.

Method: **sendCompletionFailed**

A static method to allow a servlet to send a 'complete' message with a FAILED status to the client. The method looks like this…

void sendCompletionFailed(OutputStream *response.getOutputStream()*,
String *strFailureMessage*) throws *IOException*

- *response* is the object for the server's POST response.
- *strFailureMessage* is the text describing the reason for the failure.
- A reason for the method throwing an *IOException* would be if there was an error when writing to the response object's OutputStream.

Method: **sendProgressText**

A static method to allow a servlet to send a 'status' message with some status text to the client. This method can be used in combination with the *jiBurnerListener* to retrieve status from the burn process and then communicate it back to ViewONE Pro. The method looks like this…

void sendProgress(OutputStream *response.getOutputStream(),*
int *intPercentComplete*,
String *strProgressMessage*) throws *IOException*

- *response* is the object for the server's POST response.
- *intPercentComplete* is used to indicate the percentage of the burn work that is complete.
- *strProgressMessage* is the text of the status message.

- A reason for the method throwing an *IOException* would be if there was an error when writing to the response object's OutputStream.

There are two sets of burn methods in the jiBurner class.

The first set of methods require the annotations that are to be burnt to be supplied as an xml document. This document is supplied to the methods as string arguments. These methods can be used in conjunction with the ViewONE Pro as it provides the annotations when burning in the XML format. The example servlet supplied uses the XML based methods.

The second set of methods require the annotations to be supplied in a file that is formatted in the style of an INI file. A path where the file is located is provided to the methods. The format of the INI file is defined in the "ViewONE Annotations Manual".

The xml format is following

```
<burnspec version="1.0">
   <pages merge="[true|false]">
      <page number="<pagenumber>" url="<url of page"
         subPage="<page number within the file>"
         CRC32Checksum="<Checksum of Image File>"/>
      ...
   </pages>

   <annotations>
      <annotation type="<annotationType>" page="<documentPageNumber>"
         subPage="<Page number within file>"
         CRC32Checksum="<Checksum of Image file>"
         <annotation type specific properties> />
      ...
   </annotations>

</burnspec>
```

The burn methods that require an XML formatted input are as follows…

Method:

burnAnnotations

This method allows you to burn annotations supplied in the XML document format. The output is a single file. These methods are best to be used when the *filename* or *filenames* argument contain the full document.

The method looks like this…

String burnAnnotations([String *filename | String[] filenames]*,
String *xml*)
throws *Exception*, *jiBurnFailedException*

- *filename* or *filenames* tells the library which image file (or in the case of a String[] parameter, files) to use for the burning process.

The filename(s) can be local to the server or can be an HTTP/HTTPS URL. Such a URL could point directly to a file or it could point to a service that streams image data back.

The original file will not be modified in anyway.

- *xml* is a string containing the XML data that contains the annotations that are to be burnt.

- The *String* that the method returns represents the path to the new image file with the annotations burned in. The image file name is guaranteed to be unique.

  Note that it is up to the servlet to remove the new image file and process it, as the burner will not remove any image files created.

- Reasons for the method throwing a *jiBurnFailedException* are as follows…

  – If a null filename parameter is passed in.

  – If a filename parameter is passed in which points to a source file that does not exist or cannot be read.

  – If the library suffers a parsing problem as it runs through the XML prior to placing and burning the annotations.

  – If a new image file is created that has invalid pixel data, i.e., where the pixels are either all white or all black.

Method:

burnAnnotations

This method allows you to burn annotations supplied in the XML document format. The output is a single file. This method is best used in the following circumstances:

- The document being burnt is a multi-file document (sometimes known as a multi-segmented document).

- The individual burnt pages of the document must be kept as separate files.

The *imagePageNumber* and *documentPageNumber* arguments can be used in combination with the *jiBurnerListener* listener class to provide more accurate update percentages.

The method looks like this…

String burnAnnotations(String *filename*,
int *imagePageNumber*,
int *documentPageNumber*,
String *xml*)
throws *Exception*, *jiBurnFailedException*

- *filename* or *filenames* tells the library which image file (or in the case of a String[] parameter, files) to use for the burning process.

  The filename(s) can be local to the server or can be an HTTP/HTTPS URL. Such a URL could point directly to a file or it could point to a service that streams image data back.

  The original file will not be modified in anyway.

- *imagePageNumber* relates to the page in the file which makes up the current page of the document.

  For example, if the viewer was displaying a document made up of a multipage tiff and this annotations burn was for page 3 then the imagePageNumber parameter passed should also be page 3.

  Alternatively, if the viewer was displaying a document made up of several multipage tiffs (or a mixture of multipage and single page tiffs), say two three page tiffs, and this annotations burn was for page 5 then the imagePageNumber parameter passed should be page 2 (of the second tiff).

- *documentPageNumber* relates to the page's position in the document.

  For example, if the page the library was to burn related to page 6 in the document the viewer was displaying then the documentPageNumber parameter passed should be page 6.

- *xml* is a string containing the XML data that contains the annotations that are to be burnt.

- The *String* that the method returns represents the path to the new image file with the annotations burned in. The image file name is guaranteed to be unique.

  Note that it is up to the servlet to remove the new image file and process it, as the burner will not remove any image files created.

- Reasons for the method throwing a *jiBurnFailedException* are as follows…

  - If a null filename parameter is passed in.

  - If a filename parameter is passed in which points to a source file that does not exist or cannot be read.

  - If the library suffers a parsing problem as it runs through the XML prior to placing and burning the annotations.

  - If a new image file is created that has invalid pixel data, i.e., where the pixels are either all white or all black.

Method:

burnAnnotations

This method allows you to burn annotations supplied in the XML document format. The output is a single file.

The method looks like this…

String burnAnnotations(String *filename*,
int *imagePageNumber*,
int *documentPageNumber*,
int *numPages*,
String *xml*,
PrintWriter logger) throws *jiBurnFailedException*

- *filename* tells the library what image file to use for the burning process. This will correspond to the page that annotations are to be burned on to.

  The filename can be local to the server or can be an HTTP/HTTPS URL. Such a URL could point directly to a file or it could point to a service that streams image data back.

  The original file will not be modified in anyway.

- *imagePageNumber* relates to the page in the file which makes up the current page of the document.

  For example, if the viewer was displaying a document made up of a multipage tiff and this annotations burn was for page 3 then the imagePageNumber parameter passed should also be page 3.

  Alternatively, if the viewer was displaying a document made up of several multipage tiffs (or a mixture of multipage and single page tiffs), say two three page tiffs, and this annotations burn was for page 5 then the imagePageNumber parameter passed should be page 2 (of the second tiff).

- *documentPageNumber* relates to the page's position in the document.

  For example, if the page the library was to burn related to page 6 in the document the viewer was displaying then the documentPageNumber parameter passed should be page 6.

- *numPages* relates to the number of pages that make up the document in the viewer.

- *xml* is the String holding the annotations XML sent by the viewer. All the XML – the XML for the entire document, not just the page being burned - must be passed every time the burnAnnotations call is made.

- *logger* is the PrintWriter also used to instantiate the jiBurner object. This is an optional parameter that overrides the logger defined on the jiBurner constructor.

- The *String* that the method returns represents the path to the new image file with the annotations burned in. The image file name is guaranteed to be unique.

  Note that it is up to the servlet to remove the new image file and process it, as the burner will not remove any image files created.

- Reasons for the method throwing a *jiBurnFailedException* are as follows…

  – If a null filename parameter is passed in.

  – If a filename parameter is passed in which points to a source file that does not exist or cannot be read.

  – If the library suffers a parsing problem as it runs through the XML prior to placing and burning the annotations.

  – If a new image file is created that has invalid pixel data, i.e., where the pixels are either all white or all black.

The methods that require a path to an annotation INI file are as follows:

Method:

burnAnnotationsFromIni

This method allows you to burn annotations supplied in the standard ViewONE annotations INI format onto an image, resulting in a TIFF image file.

String burnAnnotationsFromIni([String *filename* | String[] *filenames*] ,
String *annotationsFile*,
int *pageNumber*) throws *Exception*, *jiBurnFailedException*

- *filename* or *filenames* tells the library which image file (or in the case of a String[] parameter, files) to use for the burning process.

  The filename(s) can be local to the server or can be an HTTP/HTTPS URL. Such a URL could point directly to a file or it could point to a service that streams image data back.

  The original file will not be modified in anyway.

- *annotationsFile* should be a path or URL that points to an annotation file in the ViewONE INI file format. This type of file is created by default by the viewer when saving annotations. For a full explanation of the format of this file, please refer to "Annotations File Format" in the ViewONE Annotations Installation, HTML and JavaScript Manual.

- *pageNumber* the meaning of this parameter changes depending on whether you are supplying a single file or multiple files for the first parameter of this method. It is an index value that starts from 1.

  When using a single file for the first parameter, the supplied file is assumed to be either a single page image or a multi-page image. In the case of a single page image, *pageNumber* should be 1. In the case of a multi-page image, *pageNumber*

tells us which page inside the multi-page image we are interested in burning. The resulting output image will be a combination of the page specified by the *pageNumber* index within the multi-page image and the annotation data specified for the same page number within the annotations INI file.

When using multiple files for the first parameter, *pageNumber* essentially gives us an index (starting from 1) within the supplied multiple file list indicating to the library which page we would like to burn from the set. If you intend to burn page 2 then there should be at least two filenames supplied and some annotation data for page 2 in the annotations INI file.

- The *String* that the method returns represents the path to the new TIFF image file with the annotations burned in. The image file name is guaranteed to be unique.

  Please note that there will only be a single page in the resulting image file.

  Note that it is up to the servlet to remove the new image file and process it, as the burner will not remove any image files created.

- Reasons for the method throwing a *jiBurnFailedException* are as follows:

  – If a null filename parameter is passed in.

  – If a filename parameter is passed in which points to a source file that does not exist or cannot be read.

  – If a new image file is created that has invalid pixel data, i.e., where the pixels are either all white or all black.

Method:

burnAnnotationsFromIni

This method allows you to burn annotations supplied in the standard ViewONE annotations INI format onto an image, resulting in a multi-page TIFF image file.

String burnAnnotationsFromIni([String *filename* | String[] *filenames*] ,
String *annotationsFile*) throws *Exception*, *jiBurnFailedException*

- *filename* or *filenames* tells the library which image file (or in the case of a String[] parameter, files) to use for the burning process.

  The filename(s) can be local to the server or can be an HTTP/HTTPS URL. Such a URL could point directly to a file or it could point to a service that streams image data back.

  The original file will not be modified in anyway.

- *annotationsFile* should be a path or URL that points to an annotation file in the ViewONE INI file format. This type of file is created by default by the viewer when saving annotations. For a full explanation of the format of this file, please refer to

"Annotations File Format" in the ViewONE Annotations Installation, HTML and JavaScript Manual.

- The ***String*** that the method returns represents the path to the new multi-page TIFF image file with the annotations burned in. The image file name is guaranteed to be unique.

  Note that it is up to the servlet to remove the new image file and process it, as the burner will not remove any image files created.

- Reasons for the method throwing a ***jiBurnFailedException*** are as follows:

  - If a null filename parameter is passed in.

  - If a filename parameter is passed in which points to a source file that does not exist or cannot be read.

  - If a new image file is created that has invalid pixel data, i.e., where the pixels are either all white or all black.

Method:

mergePages

This method allows you to merge several single page TIFF images into one single multi-page TIFF document. This can be used in combination with *burnAnnotations* methods that output single page files to produce a single-file multi-page TIFF document.

String mergePages(String[] *filenames*, PrintWriter logger)
throws *jiBurnPageMergeFailedException*

- *filenames* tells the library which image files to use for the merging process.

  The filenames can be local to the server or can be an HTTP/HTTPS URL. Such a URL could point directly to a file or it could point to a service that streams image data back.

  The original files will not be modified in anyway.

- *logger* is a PrintWriter. This is an optional parameter that overrides the logger defined on the jiBurner constructor.

- The *String* that the method returns represents the path to the new multi-page TIFF image.

  Note that it is up to the servlet to remove the new image file and process it, as the burner will not remove any image files created.

- Reasons for the method throwing a *jiBurnPageMergeFailedException* are as follows:

  - Some of the parameters passed to the method are invalid;

  - There was an error during the merging process.

Method:

releaseResources

This method releases any resources including memory and threads that were used during the operation of the library. It is important that this method is called when you have finished using the jiBurner object. Please note that no other calls should be made to the library after this method has been called; doing so will result in undefined behaviour.

XML Checksum

When the viewer sends the annotations XML it also adds a checksum for each annotation representing the image it is placed on. When the burnAnnotations method is called this XML is parsed by the burn library and compared to the checksum of the image it is about to burn annotations on to.

If there is a mismatch then it means the image it is about to burn on to and the image the user wants to add annotations to are not the same.

Note that, although the viewer retrieves the original image for itself, the burn library also retrieves the original image for itself. The viewer sends the burn library the URL (as part of the annotations XML) in order that it can retrieve the same image. If, however, the sample servlet is modified such that this process goes wrong in some way, then it may become possible for images retrieved by the viewer to be different to images retrieved by the library.

In this case, the library will throw a jiBurnFailedException with an appropriate message and the checksum processing should help you debug your code.

Interface: **ji.burn.jiBurnerListener**

Implement this interface to use to obtain feedback from the burn process.

Method: **burnUpdate**

Implement this method to retrieve progress information from the burn. When the burn process has some progress it will call the *burnUpdate* method on the *jiBurnerListener* object passed as an argument to the *jiBurner* constructor.

The method looks like…

void burnUpdate(jiBurneEvent *event*)

Troubleshooting

Once installed, the servlet can be called with no URL parameters to ensure that it is running. For example, if your web application server is accessed on port 8080 and the Burner has been installed under the default web application name on a host called `cohost`, then the following URL entered into a web browser should display "**Burner Servlet just ran successfully (but without any parameters and did nothing but test HTTP)**":

```
http://localhost:8080/burner/burner
```

If you do not get this response then there could be a problem with the deployment of the redaction web application or there is a configuration problem.

Clearing temporary working files

Unless the behavior has been overridden with the "workingPath" parameter of the burner.properties file, the Permanent Reaction Server uses the web application provided temporary area for servlets which should be cleared down automatically by the web application server. This is normally done when re-starting the server or re-loading servlets but you should refer to your web application server's documentation for exact details.

If a "workingPath" is in use, then the folder that this parameter points to should be emptied.

Logging

The first step to diagnosing issues is to ensure that the `logFile` parameter in `burner.properties` has been set up correctly. This will output information as the servlet processes burn requests.

If you find that after enabling logging that no log file is produced then you may need to check the logs of your web application server to see if the servlet has started at all or if it is unable to write to the log file location specified.

Internal server error

Internal server errors (code 500) indicate that there has been a problem on the server-side and will require investigation of firstly the Streamer's log file and secondly your web application server's log file in order to investigate why a problem has occurred.

java.lang.InternalError

If you see this error "java.lang.InternalError: Can't connect to X11 window server using ':0.0' as the value of the DISPLAY variable error" appearing in the log files then you need to ensure that you ***either*** point to an X11 server ***or*** make use of the -Djava.awt.headless=true method

Redacting Text

Users may find that final burnt redactions over text can cover a larger area than the area actually drawn on-screen prior to the burn. This is a security feature of the underlying redaction libraries in that it detects if the redaction is also covering ***any part*** of the line of text characters below it and where it finds that it does, it applies the redaction to ***all*** the underlying text n the line below. This behavior errs on the side of caution when it comes to

whether a line of text should be redacted or not since a partially covered line can still be read so is a potential security loophole.

Single Redaction covers Shared Resources

Another behavior that can be seen is in the case of a multi-page document with multiple references to a single instance of one resource (such as an image object) embedded in the document. This behavior errs on the side of caution because where a component of that shared resource is to be redacted, if all other instances are redacted in the same manner then this is a potential security loophole.

java.lang.NoSuchMethodError: ji.burn.jiBurner.initTrace

Since version 4.0.14 of viewer, the InitTrace() method has been changed and whilst it was undocumented, this method was available in the sample code for the redaction servlet that is shipped with the viewer. So users who have updated to a version 4.0.14 or later and see this error message appearing in the burner log files will need to recompile their Java source code because of this change in order to resolve the issue.

Non-burnable redactions get burned

This is likely to be related to annotation security because there is a valid use case for such a scenario. Suppose a high level user has redacted some sensitive information and marked the redaction as "uneditable" (using annotations security) but not necessarily "burnable".
But then, a low level user takes the same document and applies a redaction elsewhere on the document, the PRS module will always burn the non-editable redaction as well, even though it is not listed as burnable. If it did not do this, the low level user would be able to see the sensitive data after they had carried out the burn process – because only the "burnable" redaction would have been burned and the other one (covering sensitive data) would have been removed.
So the burner wil always burn non-editable annotation data whether its burnable or not and this is expected behaviour because of the consideration described above.