RasterMaster® for Java™ v20.2 Programmer's Guide

RasterMaster® is the industry's leading document/image conversion and imaging library for Java. It is continually enhanced with new functionality and formats and was developed by Snowbound's experts who have nearly a hundred years of combined imaging expertise. It provides High-Speed File Conversion as well as Extensive Format Support.

This guide is designed to provide developers with a high-level overview of RasterMaster's functionality and capabilities, including conceptual information as well as code samples. For the full API reference, please see the Javadocs included with your build or visit docs.snowbound.com/rastermaster/latest/java/rastermaster-api/.

IMPORTANT NOTICE:

The online version of this manual contains information on the latest updates to RasterMaster. To find the most recent version of this manual, please visit the online version at www.rastermaster.com or download the most recent version from our website at www.snowbound.com/support/manuals.html.

Copyright Information

Manual Title: Snowbound RasterMaster® for the Java™ Platform Programmer's Reference Guide Part Number: DOC-0141-07-B

Revision: 20.2

RasterMaster® SDK Java Release Number: 20.2

Published Date: December 2019

Published by: Snowbound Software Corporation.

309 Waverley Oaks Road

Suite 401

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

Sales: 1-617-607-2010

Fax: 617-607-2002

Contents

About Snowbound Software

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

Important Phone Numbers and Links

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

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

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

Release Notes and Product Manuals:

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

Snowbound Target Markets

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

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

VirtualViewer® HTML5

Easy-to-Use in Any Environment

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

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

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

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

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

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

One Quick & Easy 10 Minute Installation

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

Technical Information

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

Server options:

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

RasterMaster® SDK

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

High-Speed File Conversion

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

Extensive Format Support

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

Technical Information

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

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

Responsive Support

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

RasterMaster Overview

The RasterMaster Java toolkit is written entirely in the Java programming language, permitting rapid development and cross-platform support. As such, there are a number of system and memory requirements and recommendations to ensure successful and rapid development.

The two major classes in the RasterMaster SDK are `Snowbnd` and `SnowANN`. While this guide will allude to many of the methods available for these classes, the full API reference can be found in the RasterMaster Java API documentation.

Determining System Requirements

System requirements to run RasterMaster Java include:

- Development Kit for the Java platform (JDK). We recommend using the latest stable release of the JDK. If you require a significantly older JDK in your environment, please contact support to verify compatibility.
- Minimum memory requirements are related to image size and necessary buffers. Buffers may require multiple megabytes if images are large (see below for more information).
- Virtual machine. Contact your hardware vendor to see if a virtual machine is available for your platform.

Snowbound Image Object

The Snowbnd image object is the main API class in the Snowbound SDK, representing a single image and all of its related operations, such as decompress, pan, scroll, zoom, and others. Just place snow.jar into your CLASSPATH, add a `Snowbnd` image object to your application, and then you will have easy access to RasterMaster Java's ability to read, manipulate, display, and save images.

Constructing a New Snowbnd Object

To construct a new `Snowbnd` object, use the default constructor below:

```
Snowbnd Simage = new Snowbnd();
```

Installation

RasterMaster® Java is easy to install as an evaluation, developer, or run-time version. Both the developer and runtime version of the RasterMaster Java ship as fully serialized builds which are very easy to install. This product is fully enabled as either a developer or runtime product.

When you purchase RasterMaster Java, you will receive a set of fully licensed binary files. The files will include Snow.jar for each purchased option.

RasterMaster Java is delivered as a .zip archive that can be manually extracted to an installation directory chosen by the user. For example: C:\Program Files\Snowbound\RMJava directory

> **Note**: For RasterMaster Java, Snowbound requests that you place all Snowbound components in a non-common directory within Windows. They should NOT be placed in the \windows, \windows\system32\ directory or other such common area.

There are three kinds of RasterMaster distributions:

1. **Evaluation**: The evaluation license is a full version of the product with the following limitations:
    - You will see a popup banner when you view or convert your first document. Subsequent documents in the same session will not elicit the banner.
    - You will see large thin Xs across each page after 50 pages or thumbnails.
    - After your expiration date, you will see a banner stating the evaluation has expired. You will not see any output. Other than that you will have full use of the product including support for all document formats.
2. **Developer**: When a developer's license is purchased, a Developer's Version banner appears. You must select the notification box for the program to continue.
   Contact sales if you need to eliminate this notification for your development purposes.
3. **Distribution**: When a distribution license is purchased, all banners disappear.

Installing RasterMaster® Java

To install the RasterMaster Java, use WinZip or another utility to extract the files to a location such as the following directory: C:\Program Files\Snowbound\RMJava.

Installation Notes

If you rename SnowboundLicense.xml, remove the SnowboundLicense.xml file from SnowboundLicense.jar and call `IMGLOW_get_license_path()` with the new name of the SnowboundLicense.xml file.

When creating a web application, edit the generated web.xml file to include the absolute path to the SnowboundLicense.jar file.

Quick start guide

This section describes how to quickly get started with the RasterMaster Java. Additional samples may be found in RasterMaster Java Samples.

If you do not find the information that you are looking for in this manual, please open a support ticket at support.snowbound.com to request a specific sample, for clarification of a method description or to help you find the information you need. We are dedicated to helping our customers succeed and we are constantly enhancing our products based on feedback from customers like you.

For an initial introduction, we've included the Single Conversion Sample below, which we find to be the fastest way to get started with RasterMaster® for Java.

The SingleConversion Sample

This sample converts all of the pages in one document. You can find the sample in the [RM Java install dir]\sample-code\com\snowbound\samples\convert directory.

The Convert sample uses three routines that are at the heart of RasterMaster Java:

1. `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.

2. `IMG_display_bitmap_aspect(java.awt.Graphics, java.awt.Container, int, int, int, int, int)` displays a valid Snowbound image.

3. `IMG_save_bitmap(byte, int)` saves the valid Snowbound image to any available format.

Below please find the full list of methods used in this sample:

- `Snowbnd()` – RasterMaster library constructor
- `IMGLOW_get_filetype`
- `IMGLOW_get_pages`
- `IMGLOW_set_ooxml_license`
- `decomp_vect`
- `IMGLOW_set_document_input`
- `IMGLOW_set_fontmap_path`
- `IMGLOW_set_overlay_path`

- `getXdpi, getYdpi`
- `getHeight, getWidth`
- `IMG_decompress_bitmap`
- `IMG_diffusion_mono`
- `IMG_color_gray`
- `IMG_promote_24`
- `setXdpi, setYdpi`
- `IMG_resize_bitmap`
- `snow.IMG_save_bitmap`

For the full method reference, see the RasterMaster Java API documentation.

Adding RasterMaster's `Snowbnd` class to your application

All classes that are a part of RasterMaster Java are contained in the Snow package. Use the import statement to call the classes, either `import Snow.*;` or `import Snow.Snowbnd;`.

Placing classes in the CLASSPATH directory

The Snow.jar containing the RasterMaster Java classes must be placed in a directory included in your CLASSPATH environment variable. If they are not in your CLASSPATH, or if your CLASSPATH environment variable is not specified, you can add them with the statement below:

```
SET CLASSPATH=C:\JAVA
```

Specifying the CLASSPATH At Compile

You can also specify the classpath at compile and run time by passing in the CLASSPATH as a parameter to the virtual machine by using the statement below:

```
c:\jdk1.7\bin\java -classpath c:\snowbound\; MySnowClass
```

Note: When using the CLASSPATH, specify the parent directory of the Snow.jar file. For example, if the Snowbound class files are located at c:\snowbound\Snow.jar, the classpath to the Snowbound classes is c:\snowbound.

Run RasterMaster samples in your IDE

To run one of the demos in your development environment, perform the following:

Note: These instructions are based on Eclipse.

1. Extract all of the package files to the workspace.
2. Be sure to download the license .zip and extract the license. Place the license in the [FilePath][MainRMFolder]\lib directory.
3. Open Eclipse and open the prepared workspace.
4. Go to **File > New Project > Java Project**.
5. Name the project "sample-code" and click **Finish**.
6. Next, right-click the project and select **Build Path > Configure Build Path...** from the drop down menu.
7. In the Libraries Tab click **Add External JARs...** and include all the files in the [FilePath][MainRMFolder]\lib directory.
8. Click **Apply** and **Close**.
9. Select the sample (e.g., SingleConversion.Java) in the convert package.
10. Adjust the `inputFile` variable to a full file path for your test file.
11. Finally, run the project.

Run Samples from the Command Prompt

To run a sample from the command prompt:

1. Make sure that that OracleJDK or OpenJDK (Java Development Kit) is installed on your system. Make sure that you have added SnowboundLicense.jar to RasterMaster/lib folder

   In the following example, the JDK is installed at this location (Windows):

```
C:\Program Files\Java\jdk-11.0.2\bin
```

2. Make sure that the code sample you want to run is in a directory name that matches the package name. In this example, RasterMaster Java was installed on c:\me. For example, the SingleConversion sample is in this directory:

```
C:\me\sample-code\com\snowbound\samples\convert
```

3. Follow the example below to change directory to be in the top-level directory, to the directory above com.

```
C:\> cd me

C:\me>
```

4. Use the example below to compile the sample from the command line by running javac from the JDK.

```
C:\me> "C:\Program Files\Java\jdk-11.0.2\bin\javac" -classpath
".;..\lib\*;.\com\snowbound\snowcommon.jar;C:\Program
Files\Java\jdk-11.0.2\bin” .\sample-
code\com\snowbound\samples\convert\SingleConversion.java
```

5. Finally, run the sample from the command line, again from the top-level directory:

```
C:\me> java -cp ".;..\lib\*;.\com\snowbound\snowcommon.jar"
com.snowbound.samples.convert.SingleConversion
```

Java samples

This section describes the Java samples included with RasterMaster Java. There are several samples included with RasterMaster Java. They are used to manipulate an image in many ways, such as altering its alter viewable size, changing its orientation, correcting its imperfections, modifying its size and resolution, converting it to another format, and adding text and graphic annotations.

You can find the all of the samples in the [RM Java install dir]\sample-code\com\snowbound\samples\ directory.

For the full method reference, please see the RasterMaster Java API documentation.

File conversion samples

These samples all primarily focus on the conversion from one type of document or file format to another. These samples can be found in the [RM Java install dir]\sample-code\com\snowbound\samples\convert\ directory.

The SingleConversion sample

This sample converts all of the pages in one document. For a list of methods used in this sample, please see The SingleConversion Sample above.

The BatchConvert sample

This sample provides a GUI that lets you choose a directory, then converts all of the files in the directory.

Listed below are the methods used in this sample.

- `IMG_display_bitmap_aspect(java.awt.Graphics, java.awt.Container, int, int, int, int, int)` displays the current image, corrected for aspect ratio, at the current X andY coordinates.
- `IMGLOW_get_pages(String)` prints the total number of pages in the input document.
- `IMGLOW_set_document_input(int, int, int)` specifies the input DPI and bit-depth for PDF, MSOffice, RTF, PCL, and AFP files.
- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.

- `IMG_promote_24(void)` promotes 1, 4, or 8-bit images to 24-bit.
- `IMG_color_gray(void)` reduces a color image to 8-bit gray scale.
- `IMG_save_bitmap(String, int)` saves the current page and print the resulting status code

The HangDetector sample

This sample converts one specified document with a timeout. If a document does not convert within three seconds, then this sample halts the conversion and prints a timeout message. Timeout speed can be adjusted to log in a way that works work best for you.

To use this sample, do the following:

1. Search within the example for the number for milliseconds that you want to use for the timeout. For example, search for 3000.
2. Search for the file location on your system. For example, \Users\imgs\1234567.pdf
3. Change the output location in the following line.

After that, you can run the sample like any other.

Listed below are the methods used in this sample:

- `Snowbnd(int, int, int)` is an alternate constructor for the `Snowbnd` object. This allows the creation of a blank image with allocated memory.
- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMG_save_bitmap(byte, int)` saves the current `Snowbnd` image object to the format specified by `comp_type`. The string value is the output file.

The saveSearchablePDF sample

This sample extracts the text from the input document and saves it as a searchable PDF. The user can also specify a text string to search for by assigning a value to the `stringToSearch` variable.

Listed below are the methods used in this sample:

- `IMG_save_document(String, byte, int)` saves to a searchable PDF file.

- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMGLOW_extract_text(String, int, int, int)` extracts text from PTOCA, PCL, PDF, MS Word, AFP, RTF and MSExcel files. It returns the buffer of extracted text in ASCII format.
- `IMGLOW_search_text(byte[], String, int, int, int[])` returns an array of structures of classes of the type `SNBD_SEARCH_RESULT`.

The Loadandsave sample

This sample is an extension of the Load sample which demonstrates saving an image. It allows you to choose JPEG with quality and interleave values. This sample can be found in the \samples\loadandsave directory.

Print samples

These samples focus on the various print capabilities of the RasterMaster SDK. These samples can be found in the [RM Java install dir]\sample-code\com\snowbound\samples\printing\ directory.

The Print sample

This sample demonstrates printing using the Snowbound product. The features includes altering the size of an image and image printing. You can find the samples in the \samples\printing\PrintingUI.java directory.

Listed below are the methods used in this sample:

- `IMG_print_bitmap(java.awt.Graphics, int, int, int, int, int)` prints the current image at the specified coordinates.
- `IMG_print_applet(java.awt.Container, int)` prints images loaded in to an applet.
- `IMG_print_applet_pages(java.awt.Container, int)` prints high-resolution multi-page documents from applets.

The SilentPrint sample

This sample demonstrates how to perform a silent print.

Listed below are the methods used in this sample:

- `IMGLOW_set_document_input(int, int, int)` specifies the input DPI and bit-depth for PDF, MSOffice, RTF, PCL, and AFP files.
- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to avalid Snowbound image.
- `IMG_print_bitmap(java.awt.Graphics, int, int, int, int, int)` prints the current image at the specified coordinates.

Image manipulation samples

These samples focus on the various image manipulation capabilities of the RasterMaster SDK.

The Thumbnails sample

This sample demonstrates creating thumbnails. When an image is opened, the thumbnail is created. You can open multiple images and view all thumbnails at once. When you click the thumbnail, the image appears in its original size. You can find the samples in the \samples\snippets\CreateThumbnail.java and \samples\thumbnails directories.

Listed below are the methods used in this sample:

- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMG_display_bitmap_aspect(java.awt.Graphics,java.awt.Container, int, int, int, int, int)` displays the current image, corrected for aspect ratio, at the current X andY coordinates.
- `IMG_resize_bitmap_bicubic(int, int)` scales down images using a bicubic interpolation algorithm.
- `IMG_resize_to_gray(int, int)` resizes a 1-bit black and white image to a (smaller) 8-bit grayscale image.
- `Snowbnd()`

The Watermarks sample

This sample demonstrates creating watermarks. You can find the sample in the \samples\watermark directory.

The Manipulation sample

This sample demonstrates image manipulation using the Snowbound product. You can find the samples in the \samples\manipulation directory.

The Color Detection sample

This sample converts a document to TIFF and selects the appropriate TIFF format (TIFF Group 4, or TIFF_LZW) based on the color profile of each source page. You can find the samples in the \samples\snippets\ColorDetection.java directory.

INPUT

The input is the document name defined in variable `inputFilename` and defaults to C:\images\mysource.pdf.

OUTPUT

The output is the document name defined in variable `outputFilename` and defaults to C:\images\myoutput.tif.

Listed below are the methods used in this sample:

- `IMGLOW_set_document_input(int, int, int)` specifies the input DPI and bit-depth for PDF, MSOffice, RTF, PCL, and AFP files.
- `IMGLOW_get_pages(String)` prints the total number of pages in the input document.
- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMGLOW_detect_color()` checks all pixels to determine if the image is color or gray scale.
- `IMG_thresh_mono(int)` permanently converts 4, 8, or 24-bit images to 1-bit per pixel using a threshold passed as the second argument.
- `IMG_save_bitmap(byte, int)` saves the current `Snowbnd` image object to the format specified by `comp_type`. The string value is the output file.

The Save Page to Memory sample

This sample loads an image from a document and saves it in the desired format to a byte array. You can find the samples in the \samples\snippets\SavePageToMemory.java directory.

INPUT

The input is the document name defined in variable `inputFilename` and defaults to C:\images\mysource.pdf.

OUTPUT

The output is a byte array containing the image data of the page.

Listed below are the methods used in this sample:

- `IMGLOW_get_pages(String)` prints the total number of pages in the input document.
- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMG_save_bitmap(int, int, int, int)` allows saving to a byte array without knowing or guessing the final size of the compressed image or document.
- `IMG_save_bitmap(byte, int, int, int)` saves the current `Snowbnd` image object to a returned byte array.
- `IMG_decompress_bitmap(java.io.DataInputStream,int)` decompresses the current page and print the resulting status code.

Viewing and Annotating Documents

These samples focus on the various display and annotation capabilities of the RasterMaster SDK.

The Viewing and Annotations sample

This sample is an image viewer that demonstrates the following image manipulations: altering the viewable size, changing the orientation, changing the size and resolution, making corrections, and flipping through pages of a multi-page file. You can find the samples in the \samples\annapp\DemoAnnApp.java directory.

This sample also demonstrates several annotation methods, such as adding edits, bitmaps, lines, sticky notes, and more through a variety of methods. These annotations can be in both text and graphic format.

Listed below are the methods used in this sample:

- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMG_deskew_bitmap(int)` is a fast rotate for 1-bit skewed images.
- `IMG_despeckle_bitmap(int) removes noise (random pixel data)` from 1-bit images.
- `IMG_display_bitmap(java.awt.Graphics, int, int, int, int)` displays the current image at the current X andY coordinates and is not corrected for aspect ratio.
- `IMG_display_bitmap_aspect(java.awt.Graphics,java.awt.Container, int, int, int, int, int)` displays the current image, corrected for aspect ratio, at the current X and Y coordinates.
- `IMG_erase_rect(int, int, int, int, int, int, int)` fills the inside or outside of the specified rectangle(xs, ys, xsize, and ysize parameters) with the color passed in. It can be used to remove borders, for instance.
- `IMG_flip_bitmapx(void)` flips the image horizontally to produce a mirror image.
- `IMG_flip_bitmapy(void)` flips the image vertically to produce a mirror image.
- `IMG_get_deskew_angle(int[], int, int)` returns the skew angle for 1-bit images.
- `IMG_invert_bitmap(void)` inverts the current image, changing pixels by a simple XOR operation applied to each byte of the image in memory.
- `IMG_merge_block(Snow.Snowbnd, int, int, int)` draws a source image on top of the current image at the x and y coordinates(x pos and y pos) passed in.
- `IMG_print_applet_pages(java.awt.Container, int)` prints high-resolution multi-page documents from applets.
- `IMG_resize_bitmap(int, int)` resizes the image.
- `IMG_rotate_bitmap(int)` rotates the current image clockwise by display angle.
- `IMG_save_bitmap(byte, int)` saves the current Snowbnd image object to the format specified by `comp_type`. The string value is the output file.
- `IMG_scroll_bitmap(java.awt.Container,java.awt.Event)` scrolls the bitmap. It essentially moves the current cropping rectangle in the direction indicated by the Scroll Bar thumb. For users who do not want the scroll bars visible, a similar effect can be achieved by moving the cropping rectangle around using `IMG_set_croprect`.

- `IMG_window_level(int, int, int)` changes the range of displayed pixels. It performs window leveling and 16-bit gray scale images.

- `SANN_read_ann(DataInputStream, int)` reads an annotation file from disk. Pathnames are absolute. It supports multi-page annotation files.

- `SANN_set_coprect(int, int, int, int)` sets the cropping rectangle is essentially the size of the display container (which is usually a subclass of `java.awt.Container`), mapped on to the image (corrected for image scaling).

- `SANN_display_annotations(java.awt.Graphics, java.awt.Container, int, int, int, int)` displays all current annotations (superimposes on the image) to the Graphics context pointed by g.

- `GetClientRect(java.awt.Container, Snow.SANN_RECT)` fills a `SANN_RECT` object with the current height and width of the display area.

- `SANN_highlight_object(java.awt.Container, Snow.SANN_RECT, int)` highlights the bounding box of the annotation object specified by `graphic_num`.

- `ui_delete(java.awt.Container)` deletes the currently selected annotation object. It is used by a built in annotation editing pop-up menu to fire the SANN_delete_object method on the currently selected annotation object.

- `ui_startmove(java.awt.Container)` notifies the annotation library that the current annotation object is being moved.

- `ui_startresize(java.awt.Container)` notifies the annotation library that the current annotation object is being resized.

- `ui_wm_lbuttondown(java.awt.Container, int, int, byte[])` begins tracking mouse movements when the mouse button is pressed. Call this method during the mouse down or mouse pressed events.

- `ui_wm_mousemove(java.awt.Container, int, int, byte)` notifies the library of the current mouse coordinates.

- `ui_wm_lbuttonup(java.awt.Container, int, int, byte[]) notifies` the annotation library that the mouse button has been released.

- `SANN_write_ann(String, int, byte[])` writes out annotations to the disk file.

- `SANN_activate_all_objects(void)` activates all text based annotation objects to allow text editing in the TextArea represented by these objects. Currently used only on the `SANN_EDIT` and `SANN_POSTIT` objects.

- `SANN_deactivate_all_objects(void)` is used on the `SANN_EDIT` and `SANN_POSTIT` objects to disallow editing of text in all text based objects. It makes the window inactive and draws the object data statically, superimposed on the image.
- `SnowAnn Sann= new SnowAnn(int, int);`
- `Snowbnd()`

The Swing sample

This sample demonstrates using Java Swing classes. It demonstrates several image methods, such as altering the viewable size, changing the orientation, modifying the size and resolution, making corrections, and converting to other formats. You can find the samples in the \samples\swing directory.

Listed below are the methods used in this sample:

- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMG_deskew_bitmap(int)` is a fast rotate for 1-bit skewed images.
- `IMG_despeckle_bitmap(int)` removes noise (random pixel data) from 1-bit images.
- `IMG_display_bitmap(java.awt.Graphics, int, int, int, int)` displays the current image at the current X and Y coordinates and is not corrected for aspect ratio.
- `IMG_display_bitmap_aspect(java.awt.Graphics,java.awt.Container, int, int, int, int, int)` displays the current image, corrected for aspect ratio, at the current X and Y coordinates.
- IMG_erase_rect(int, int, int, int, int, int, int) fills the inside or outside of the specified rectangle (`xs`, `ys`, `xsize`, and `ysize` parameters) with the color passed in. It can be used to remove borders, for instance.
- `IMG_flip_bitmapx(void)` flips the image horizontally to produce a mirror image.
- `IMG_flip_bitmapy(void)` flips the image vertically to produce a mirror image.
- `IMG_get_deskew_angle(int[], int, int)` returns the skew angle for 1-bit images.
- `IMG_invert_bitmap(void)` inverts the current image, changing pixels by a simple XOR operation applied to each byte of the image in memory.
- `IMG_merge_block(Snow.Snowbnd, int, int, int)` draws a source image on top of the current image at the x and y coordinates (`xpos` and `ypos`) passed in.

- `IMG_print_applet(java.awt.Container, int)` prints images loaded in to an applet.
- `IMG_resize_bitmap(int, int)` resizes the image.
- `IMG_rotate_bitmap(int)` rotates the current image clockwise by display angle.
- `IMG_save_bitmap(byte, int)` saves the current Snowbnd image object to a byte array.
- `IMG_window_level(int, int, int)` changes the range of displayed pixels. It performs window leveling in 8 and 16-bit gray scale images.
- `Snowbnd()`

The VectorPDF sample

This sample demonstrates how to use vector PDF. You can find the samples in the \samples\vectorpdf directory.

Listed below are the methods used in this sample:

- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMG_display_bitmap_aspect(java.awt.Graphics,java.awt.Container, int, int, int, int, int)` displays the current image, corrected for aspect ratio, at the current X and Y coordinates.
- `IMG_print_bitmap(java.awt.Graphics, int, int, int, int, int)` prints the current image at the specified coordinates.

The VectorPDFPageManipulation sample

This sample demonstrates how to delete, extract, and append pages to and from a vector PDF file without causing additional pages in that vector PDF to change into non-vector pages because of those manipulations. Pages being appended to vector PDFs can be vector or non vector pages. You can find the samples in the \samples\vectorpdf directory.

Listed below are the methods used in this sample:

- `IMGLOW_delete_page(String, int, int)` returns the ability to delete the specified page from a multi-page PDF document.
- `IMGLOW_extract_page(String, int, int)` extracts the specified page from a multi-page PDF or TIFF document.

- `IMGLOW_append_page(String, byte, int)` allows the appending of PDF content for individual pages without any changes.
- `IMG_decompress_bitmap_stream(String, int)` reads images from a standard URL.
- `IMG_save_bitmap(int, int, int, int)` allows saving to a byte array without knowing or guessing the final size of the compressed image or document.

Scanning

This sample focus on the scanning capabilities of the RasterMaster SDK. These sample can be found in the [RM Java install dir]\sample-code\com\snowbound\samples\scanning\ directory.

The Scanning sample

This sample demonstrates how to use scanning.

Listed below are the methods used in this sample:

- `IMG_scan_acquire()` is a TWAIN scanning routine. It scans an image on the current installed scanner or input device and returns the standard library image handle. `IMG_scan_open_source()` must be called first.
- `IMG_scan_acquire_feeder()` is a TWAIN scanning method that allows scanning from a scanner with an automatic document feeder.
- `IMG_scan_feeder_close()` is a TWAIN method used to close the scanner after calls to `IMG_scan_acquire_feeder()`.
- `IMG_scan_open_source()` is a TWAIN scanning method that allows setting the desired scan or input device.
- `IMG_scan_get_source_list()` calls the native SDK method from `ScanInterface` to get a list of scanners.
- `IMG_scan_pages()` is a TWAIN scanning routine. It scans all pages from the currently selected scanner or input device and saves to the path\filename specified by filename in the image format file type.
- `IMG_scan_get_cap()` returns the current setting of the specified TWAIN capability.
- `IMG_scan_set_cap()` sets the value of the specified TWAIN capability.

- `IMG_scan_set_caps()` sets scanning parameters such as height, width, and bits per pixel for the current scanner.
- `IMG_scan_setup()` sets the value of the specified TWAIN capability.

Converting File Formats

RasterMaster supports over 100 file formats. Many formats such as TIFF are very broad in the internal support of compression and bit depths. Not all formats can support all bit depths.

All RasterMaster Java products import and convert file formats to Snowbound Software's internal format at decompress time. The format is a simple uncompressed DIB (Device Independent Bitmap) format stored in memory.

This format is decompressed or imported and can be saved out to any supported format. See Supported File Formats (available in your build directory) for a complete list of supported file formats.

Automatically Detecting File Formats

In most cases, you do not need to know the file format you wish to convert since RasterMaster automatically detects the file format regardless of the file extension. File extensions are not mandatory.

RasterMaster supports automatic promotion of images to destination files. For example, JPEG images can only be written out at 8 and 24-bits per pixel. In order to save a 1-bit TIFF image as JPEG, the developer must promote the TIFF to 8 or 24-bits per pixel.

In our other libraries, this is accomplished by calling one of the promote functions. In RasterMaster, however, the library automatically determines the bits per pixel for the destination format and promotes accordingly.

Supported Pixel Depths

JPEG supports 8, 24, and 32-bits per pixel. Black and white images use 1-bit per pixel, while color images usually require 24-bits per pixel. For many of these cases, RasterMaster Java automatically converts the pixel depth to the appropriate value, based on the output format specified.

Converting Up 8 or 24-bit Pixel Depths

To convert any pixel depth up to 24-bit, use `IMG_promote_24`. For information about this method, please see the RasterMaster Java API documentation.

See the Supported File Formats document for the pixel depths of each supported format.

For more information on converting from a higher pixel depth format to a lower pixel depth format or a lover pixel depth format to a higher pixel depth, please see Color Manipulation below.

File Format Conversion Error Messages

When saving to a format, if the error returned is `PIXEL_DEPTH_UNSUPPORTED (-21)`, the output format does not support the current bits per pixel of the image you are trying to save.

It is possible to convert the pixel depth of these images using `IMG_promote_24`. For information about this method, please see the RasterMaster Java API documentation.

Extracting Text

Snowbound Software supports extracting text to an ASCII file for some popular formats such as PTOCA, PCL, PDF, MS Word, AFP, and MS Excel. Please see `IMGLOW_extract_text (String, int, int, int)` in the RasterMaster Java API documentation for more information on extracting text.

Input Documents and Images

This section describes how to input documents and images and describes how to use the format hash for the Java platform. It also describes how to work with PDF and other document file formats in Snowbound Software's RasterMaster® SDK products.

Determining the Document's Format and its Capabilities

The format hash methods provide an easy way to get information about supported formats in the RasterMaster Java.

To use these methods, you must first create the `FormatHash` class as follows. This class maintains the set of valid Snowbound file formats.

To get instances of this class, use the static method `getInstance()`.

```
FormatHash fh = FormatHash.getInstance();

Format faxFormat = fh.getFormat ("TIFF_G4_FAX");

int faxCode = faxFormat.getFormatCode();
```

Working with Document File Formats

Snowbound Software supports many document file formats in most of its viewing, converting, and RasterMaster® SDK products.

The supported file formats include:

- PDF - Adobe portable file
- PCL - Hewlett Packard printer format
- DOC - Microsoft Word processor format
- XLS - Microsoft Excel spreadsheet format
- PPT - Microsoft PowerPoint presentation format

- RTF - Rich text file format
- AFP and MODCA - IBM advanced printing format

These formats, like any other file formats supported by Snowbound Software's products, are read in using a call such as `IMG_decompress_bitmap(String, int)`. Snowbound Software's products auto-detect the file format so there is no need to specify which format you are reading. You don't even need a specific extension.

These formats contain graphics commands such as line and text drawing. This differs from bitmap formats that are a 2-dimensional array of bytes forming a picture. Normally, documents are drawn or rendered to a bitmap in RasterMaster Java, but RasterMaster Java can also produce searchable text documents.

The PDF file created can be searched for words or phrases with the use of a text searching application. Please note that the only currently supported input formats for creating searchable PDF output are AFP/MO:DCA, PTOCA, PCL, RTF, DOC(MS Word), and MS Excel files. For more information on creating PDF files that are text searchable, please see Extracting Text for Searchable PDF output.

The rendering size for the bitmap can be set by the following call to Snowbound Software's library:

```
IMGLOW_set_document_input(int dpi, int bits_pix, int format);
```

This method sets the destination size for saving PDF files. The `xsize` and `ysize` are the output sizes in points. A point is 1/72 of an inch. Only PDF, PCL, and AFP file formats can be saved. The PCL and APF file formats are saved as bitmap. The PDF file format can be saved as bit- map or searchable text. You can specify the output size with PDF using the `IMGLOW_set_pdf_output(int, int)` call.

Reading Multiple Pages

All Snowbound libraries decompress a single page at a time. This section describes the read multi-page formats and how to decompress and determine page count.

RasterMaster Java currently supports the following multi-page formats:

- TIFF
- MO:DCA/IOCA lPDF

- PCL
- MS Word (read only)
- MS Excel (read only)
- RTF (read only)

Decompressing a Multi-page Image

Each page must be decompressed and then saved separately. In other words, a decompress then save must be performed for each page.

Determining Multi-page Page Count

You can keep decompressing pages until a `PAGE NOT FOUND` or `-11 error` is returned from the decompress method, or you can query the number of pages with the `IMGLOW_get_pages` method.

See `IMGLOW_get_pages(java.io.DataInputStream)` and `IMGLOW_get_pages(String)` in the RasterMaster Java API documentation for more information.

For more information on the methods used or multi-page formats, consult the following methods in the RasterMaster Java API documentation:

- `IMGLOW_get_pages(java.io.DataInputStream)`
- `IMGLOW_get_pages(String)`
- `IMG_decompress_bitmap(java.io.DataInputStream, int)`
- `IMG_decompress_bitmap(String, int)`

Extracting Text for Searchable PDF output

The document conversion feature extracts and converts vector or document file formats such as AFP/MO:DCA, PCL, and MSWord to vector PDF format. The PDF file will be in a true vector format, meaning that it will not be in a bitmap format. The PDF file will retain the original text and graphics commands. Font information such as the font typeface, font height, and bold/Italic attributes will remain the same. This allows the output PDF file to be created as text searchable. The PDF file created can be searched for words or phrases with the use of a text searching application. Please note that the only currently supported input formats for creating searchable PDF output are AFP/MO:DCA, PTOCA, PCL, DOC (MS Word), and MS Excel files.

Conversion and text extraction occur in the following two step process:

1. A call is made to extract the text, graphics, and bitmap data. The `IMGLOW_extract_text()` method extracts text, graphics, and position information from the file name passed in.
2. The buffer returned is used as an argument in the call to write out the new PDF file. See `IMGLOW_extract_text(String, int, int, int)` for more information.

The `IMG_save_document()` method takes a buffer passed in with text, graphics, and position information to create the document file output. The output file contains searchable text. Normally, the `IMG_save_bitmap()` methods only create a bitmap file. This only supports the PDF file as an output file. See `IMG_save_document(String, byte, int)` and `IMG_save_document(byte, byte, int)` in in the RasterMaster Java API documentation for more information.

The `IMGLOW_extract_text(String,int,int,int)` method extracts the specified page from a multi-page document. `IMGLOW_extract_page` differs from `IMG_decompress_image` in that it preserves the format of the original page rather than converting it to the Snowbound common raster format. The currently supported input formats are raster PDF, searchable PDF, and TIFF.

For an example of this capability, please see the saveSearchablePDF sample.

Working with RasterMaster Java's Vector Display Technology

This section describes how to work with Snowbound Software's vector display technology. The VectorPDF sample demonstrates how to use vector PDF.

Information about RasterMaster Java's Vector Display Technology

Vector display technology gets its name from the fact that it was originally designed to display, convert, and manipulate bitmap or raster graphics. These are files that contain just an array of bytes that form a grid of pixels or dots.

Anything can be contained in the array, such as color images or plain text. There is no way to easily tell. Vector graphics such as PDF, MS Word, and MS Excel files contain commands that tell the viewer to display text or graphics at specific positions. Graphics could be lines, circles, or other shapes. In the past, all Snowbound products would rasterize or draw the text and graphics commands into an off-screen buffer which would then be imported into the main code as a bitmap image.

The main problem with this approach is one of resolution, or fidelity of the image being displayed. Bitmap images contain a finite array of pixels. When zoomed enough, the display just makes each pixel bigger. An image then becomes pixelated or blocky, sometimes referred to as the jaggies.

The new vector display technology will no longer rasterize the image, but instead it will store the list of graphic commands for an image. At display time, the image is drawn directly to the monitor screen, and each time the image is displayed it is redrawn. This can be a little slower than displaying a bitmap image, but the benefit is infinite resolution. The new display is no longer making pixels bigger, but instead it redraws text and graphics at a larger size.

Lines, circles, graphics, etc. remain smooth since they are continually redrawn to match the appropriate zoom level. Text remains at a perfect quality since a larger font is selected for each successive zoom level. An example benefit would be for large engineering drawings, which can contain thousands of graphic commands. They can continually be zoomed to provide more and more detail.

Implementing RasterMaster's Vector Display Technology

If you are an existing customer, implementation is simple. Just set the public variable as follows:

```
Simage.decomp_vect = true;
```

The decompress and display methods listed below will work the same:

- `IMG_decompress_bitmap(String, int)` reads in a document in any format and converts it to a valid Snowbound image.
- `IMG_display_bitmap_aspect(java.awt.Graphics,java.awt.Container, int, int, int, int, int)` or `IMG_display_bitmap(java.awt.Graphics, int, int, int, int)`

Currently, RasterMaster Java's vector display technology is only available for PDF images. Other formats such as MS Word, MS Excel, PowerPoint, and AFP are currently in development.

For a concrete example of this capability, please see the VectorPDF sample.

AFP Specific Considerations

AFP files can use specific fonts to achieve a particular look. If you find that the fidelity of the out- put for your AFP documents is lacking, particularly in regard to text size and spacing or barcodes, then you can customize RasterMaster Java to use particular fonts when processing your AFP files.

RasterMaster Java is easy to customize to improve the look of your AFP documents. Snowbound Software allows you to map the fonts in your AFP/MODCA document to fonts on your system using a mapping file named snbd_map.fnt. The snbd_map.fnt file is custom crafted to specify the fonts used in your AFP files and on your system.

If you provide a representative sample AFP document to Snowbound Software by entering a support issue at support.snowbound.com, we will provide you with a custom snbd_map.fnt file usually in a few business days that will improve the display and print quality of your AFP documents.

RasterMaster Java automatically loads the customized snbd_map.fnt file if you place it in one of the following directories:

- The directory into which images are being read. For example: C:\AFP\fontmap
- The directory where your application exists as long as you are not changing directories with a dialog box.

The following methods allow you to set font mapping:

1. The `IMGLOW_set_fontmap_path()` method sets the path of the font mapping file.
2. The `IMGLOW_set_fontmap()` method programmatically sets the font mapping.

Format of Font Mapping Data

Any AFP font name can now be mapped to the following:

- `face name`
- `point size`
- `bold attributes`
- `italic attributes`

The snbd_map.fnt file is a simple ASCII text file. Each entry is ended with a carriage return line feed. The following are two sample entries:

```
C0BC25I3,Courier,10,0,0

C0CGT12S,Times New Roman,14,0,1
```

Table 4.15: Description of a sample entry in the snbd_map.fnt file

Variable	Description
C0BC25I3	Font resource name in the AFP file.
Courier	New face name to map to.
10	New size in points or 1/72 of an inch.
0	Bold attribute: 0 - off, 1 - on.
0	Italic attribute: 0 - off, 1 - on.

PDF Specific Considerations

Reading and writing support for PDF formats is included in the following Imaging SDK products:

- Imaging SDK for WindowsDLL
- Imaging SDK for ActiveX
- Imaging SDK for the Java Platform

Currently, Snowbound Software does not provide PDF reading support in UNIX and Macintosh products. Snowbound Software does provide PDF writing support for its UNIX and Macintosh products.

PDF reading is available as a separate option. For the Imaging SDK for Java, a special PDF version is available. For the other products, the pdfplug.dll file is required for PDF reading support. This file must be included in the same directory as the ActiveX or DLL. You can also include the file in the following directory: \Windows\system.

READING OR DECOMPRESSING A PDF DOCUMENT

To read in or decompress a PDF document, use any of the decompression calls such as `IMG_decompress_bitmap(String, int)`. All Imaging SDK products automatically detect the file format and do not use the extension on the file name.

The default size for decompressing PDF documents is 200 dots per inch (DPI) and 24 bits per pixel. To alter this or to convert PDF files, you can call the `IMGLOW_set_pdf_input(int, int)` method.

Saving to a PDF Document

Imaging SDK products support writing PDF files. To save a file as PDF, use any of the saving calls such as `IMG_save_bitmap()` with the file format set to PDF or the file type constant number 59.

RasterMaster Java can save to two different types of PDF files; raster PDF and vector PDF. `IMG_save_bitmap(byte, int, int, int)` saves a rasterized PDF with high fidelity rendering. `IMG_save_document(byte, byte, int)` outputs a vector PDF with text that can be searched so vector PDF is also known as searchable PDF. Currently, the text output is limited to ASCII.

Working with Black and White Images

If you are working with black and white documents, you will save a lot of memory and increase performance by setting the DPI to 300 or 200 and the bits per pixel to 1 when reading PDF images. For 1-bit or black and white images, will write out or save a PDF with CCITT - G4 compression for black and white images. For more information on improving performance, please see Improving Performance in Improving Performance or Quality.

Working with Color Images

For color bitmaps, a PDF will be created with JPEG compression.

Changing Output Page Size

The default page size is set to 8.5 x 11. The bitmap image saved is centered at this page. To alter the output page size, use the call the IMGLOW_set_pdf_output(int, int) method. The xsize and ysize parameters are the page size in points or 1/72 of an inch.

Performance

If you are working with black and white documents, you will save a lot of memory and increase performance by setting the DPI to 300 or 200 and the bits per pixel to 1 when reading all document formats. For more information on improving performance, please see Improving Performance in Improving Performance or Quality.

Ensuring Java Reading for Text-based Formats

This section describes how to read PDF, AFP, Word, HTML, PCL, and ASCII images in Java on UNIX-based operating systems when using the Imaging SDK for the Java platform.

To read PDF, AFP, Word, HTML, PCL, and ASCII images in Java on other UNIX-based operating systems, Snowbound software uses the Abstract Windowing Toolkit (AWT). In Solaris, calls into the java.awt package will trigger a connection to the X11 Windowing environment.

Since servers often operate as headless, or without a display, there is no X11 environment. The result is a NoClassDefFound error, similar to the following:

```
java.lang.NoClassDefFoundError

at java.lang.Class.forName0(Native Method)

at java.lang.Class.forName(Class.java:115)

at java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment(
GraphicsEnvironment.java:53)

at

java.awt.image.BufferedImage.createGraphics (BufferedImage.java:1006)
```

For JDK 1.4 or greater In JDK 1.4 and above, instruct the JDK to operate as a headless server, by setting a java run time option as follows:

```
java -Djava.awt.headless=true
```

In many cases, setting this option involves modifying the start-up script to your application server (often `JAVA_OPTIONS` or `JAVA_OPTS`).

Search for Text and Redact Code

The following is a sample to search for text and redact code:

```
/* get all text from source document */
buff = Simage.IMGLOW_extract_text(FileName,length,error,myFrame.page); /*
find all instances of search string */
SNBD_SEARCH_RESULT mSearchResults[] = null;
mSearchResults = Simage.IMGLOW_search_text(buff,
new String("image"),
0, error);
// myFrame.Sann = new SnowAnn(Simage.getWidth(),Simage.getHeight()); //
SANN_RECT rc = new SANN_RECT();
int x = 0;
/* find and redact all instances of the searched text */
int xs,ys,xsize,ysize;
while (x < mSearchResults.length)
{xs = mSearchResults[x].rc[0].left;
ys = mSearchResults[x].rc[0].top;
ysize = mSearchResults[x].rc[0].bottom - mSearchResults[x].rc[0].topx- size
= mSearchResults[x].rc[0].right - mSearchResults[x].rc[0].left; /* draw a
black rectangle over the searched result */
Simage.IMG_erase_rect(xs,ys,xsize,ysize,0,1,0);
// Sann.SANN_add_object(2,rc,null,null,0);
x++;}/* save out new image redacted doc */
stat = Simage.IMG_save_bitmap("c:\\temp\\redact.pdf",Snow.Defines.PDF);
```

Output Documents and Images

This section describes how to output an image or document to a different file format and the available save methods for the `Snow.Snowbnd` class.

Please see the RasterMaster Java API documentation for detailed information about the methods mentioned below.

Image Compression

This section describes compressed images.

Preferred Formats

Most of the compression algorithms only compress a specific type of image data. Compression techniques for 24-bit color images usually do not work well on 1-bit or bi-level images. Similarly, compression for 1-bit images do not compress well for 24-bit color images.

24-Bit Color Images

For 24-bit color images, use the JPEG format in RasterMaster Java for the best conversion results.

8-Bit Gray Scale Images

For 8-bit gray scale images, use the JPEG format in RasterMaster Java for the best conversion results.

1-bit Bi-Level Images

For 1-bit bi-level images, use the TIFF G4 in RasterMaster Java for the best conversion results. The JBIG format works about twenty percent better than the TIFF G4 format, but it can be much slower.

Saving multi-page documents and images

This section describes how to save and convert multi-page images in RasterMaster Java.

Multi-page Images

All Snowbound libraries decompress a single page at a time. This section describes the multi-page formats and how to decompress, determine page count, and save multi-page images.

Supported Multi-page Formats

RasterMaster Java currently supports the following multi-page formats:

- TIFF
- MO:DCA/IOCA
- PDF
- PCL
- MSWord-Reading
- MSExcel-Reading
- RTF-Reading

Decompressing a Multi-page Image

Each page must be decompressed then saved separately. In other words, a decompress then save must be performed for each page.

Determining Multi-page Page Count

You can keep decompressing pages until a `PAGE NOT FOUND` or `-11 error` is returned from the decompress method, or you can query the number of pages with the `IMGLOW_get_pages` method.

Saving Multi-page File Formats

When saving to a multi-page file format, keep in mind that all RasterMaster Java products append or add new pages to an image file if the file already exists. If you do not want to keep saving to the same file, you must first delete the page or save each page to a unique file name

Methods Used for Multi-page Format Methods

Method	Description
IMGLOW_get_pages()	Detects the number of pages in a file.
IMG_decompress_bitmap()	Reads pages and specifies the specific page to import.
IMG_save_bitmap()	Saves multi-page images. For an existing file, new pages are appended. For new files, new pages with the specified name are created as necessary. **Note**: Use a unique filename when saving to multi-page formats, otherwise, images are automatically appended to any file with the same name.

Changing the DPI in the RasterMaster SDK Library

This section describes how to change the DPI in the RasterMaster® SDK library. Changing the DPI in the RasterMaster® SDKLibrary.

When the library decompresses an image, it is converted into an MS-Windows device independent bitmap (DIB), which contains a header for the width, height, and also the dots per inch.

To change or look at the DPI information from an image use the following code:

```
int xDPI, yDPI;
xDPI = Simage.getXdpi();
yDPI = Simage.getYdpi();
if (xDPI < 300 || yDPI < 300)
{
Simage.setXdpi(300);
Simage.setYdpi(300); }
```

Color Manipulation

This section describes the RasterMaster Java color manipulation methods. The quality of the conversion is determined by the input and output dots per inch resolution and the output compression type. If you go from a higher to a lower resolution or a lossy compression type, then the quality of the image will degrade. Use color reduction methods to create a lower image quality that is smaller in size and is speedier to process.

Please note that some formats only support a certain number of bits per pixel, so you may get an unsupported pixel depth error when trying some of these methods. Please see the Supported File Formats document for the pixel depths of each supported format.

Please see the RasterMaster Java API documentation for detailed information about these methods.

Methods:

- `IMG_antique_effect(void)`
- `IMG_color_gray(void)`
- `IMG_deskew_bitmap(int)`
- `IMG_despeckle_bitmap(int)`
- `IMGLOW_detect_color()`
- `IMG_diffusion_mono(void)`
- `IMG_histogram_equalize()`
- `IMG_invert_bitmap(void)`
- `IMG_promote_8(void)`
- `IMG_promote_24(void)`
- `IMG_remove_red_eye(int, int, int, int)`
- `IMG_thresh_mono(int)`
- `IMG_rgb_to_cmyk(void)`
- `IMG_sharpen_bitmap(int)`

Photo Editing

This section describes the RasterMaster Java photo editing methods. The quality of the conversion is determined by the input and output dots per inch resolution and the output compression type. If you go from a higher to a lower resolution or a lossy compression type, then the quality of the image will degrade. Use color reduction methods to create a lower image quality that is smaller in size and is speedier to process. If you go from a higher to a lower resolution or a lossy compression type, then the quality of the image will degrade.

Please note that some formats only support a certain number of bits per pixel, so you may get an unsupported pixel depth error when trying some of these methods. Please see the Supported File Formats document for the pixel depths of each supported format.

Please see the RasterMaster Java API documentation for detailed information about these methods.

Methods:

- `IMG_antique_effect(void)`
- `IMG_auto_crop_bitmap(int)`
- `IMG_despeckle_bitmap(int)`
- `IMG_erase_rect(int, int, int, int, int, int, int)`
- `IMG_flip_bitmapx(void)`
- `IMG_flip_bitmapy(void)`
- `IMG_histogram_equalize()`
- `IMG_resize_bitmap(int, int)`
- `IMG_resize_bitmap_bicubic(int, int)`
- `IMG_rotate_bitmap(int)`
- `IMG_sharpen_bitmap(int)`
- `IMGLOW_detect_color()`

Image Display

This section describes the available image display methods for the `Snow.Snowbnd` class.

Please see the RasterMaster Java API documentation for detailed information about these methods.

Methods:

- `display_angle(int)`
- `fit_to_height(int)`
- `fit_to_width(int)`
- `IMG_auto_crop_bitmap(int)`
- `IMG_auto_orient(Snow.Snowbnd, int)`
- `IMG_display_bitmap(java.awt.Graphics, int, int, int, int)`
- `IMG_display_bitmap_aspect(java.awt.Graphics,`
- `IMG_display_bitmap_smooth(java.awt.Graphics,`
- `IMG_erase_rect(int, int, int, int, int, int, int)`
- `IMG_fill_bitmap_pattern(int, int, int, int, int, int, int, int,`
- `IMG_flip_bitmapx(void)`
- `IMG_flip_bitmapy(void)`
- `IMG_resize_bitmap(int, int)`
- `IMG_resize_bitmap_bicubic(int, int)`
- `IMG_resize_to_gray(int, int)`
- `IMG_rotate_bitmap(int)`
- `IMG_scroll_bitmap(java.awt.Container, java.awt.Event)`
- `IMG_set_croprect(int, int, int, int)`
- `IMG_window_level(int, int, int)`
- `IMGLOW_put_raster(int, byte[])`
- `IMGLOW_set_ascii_attributes(int, int, int, int, int, int, int, int, int, int, int, int, int, boolean, boolean, String, int, double)`
- `IMGLOW_set_brightness(int)`
- `IMGLOW_set_contrast(int)`
- `IMGLOW_set_decompsize(int, int)`
- `IMGLOW_set_decomp_rect(int, int, int, int)`
- `IMGLOW_set_image_orientation(int)`
- `IMGLOW_set_pdf_flags(int)`

- IMGLOW_set_pdf_flags(1)
- IMGLOW_set_pdf_output(int, int)
- map_image_to_wnd(java.awt.Container, java.awt.Point)
- map_wnd_to_image(java.awt.Container, java.awt.Point)
- set_croprect(int, int, int, int)
- set_croprect_scroll(java.awt.Container, int, int, int, int, int)
- setFrame(java.awt.Container)

Image Quality Methods

This section describes the available image quality manipulation methods for the `Snow.Snowbnd` class.

Please see the RasterMaster Java API documentation for detailed information about these methods.

Methods:

- `IMGLOW_set_comp_quality(int)`
- `IMGLOW_set_document_input(int, int, int)`
- `IMGLOW_set_document_input(int, int, int, int, int)`
- `IMGLOW_set_gamma(int)`
- `IMGLOW_set_jpg_interleave(int, int)`
- `IMGLOW_set_pcl_input(int, int)`
- `IMGLOW_set_pdf_fontpath(String)`
- `IMGLOW_set_pdf_input(int, int)`
- `setYdpi(int)`

Page Manipulation Functionality

This section describes the available page manipulation functionality and methods.

RasterMaster Java allows for page manipulation of a supported file format and saves it with original page formatting. Page manipulation of documents is at a page-by-page level, such as deleting pages, moving pages, adding pages, and creating new documents from pages of other documents. Page content remains the same and preserves the formatting.

The ordering of the pages may be changed, and pages may be added or removed. Pages of a searchable PDF that are reordered may be saved as a searchable PDF. New documents can be created by moving or copying pages of other documents. You can create a new document with pages copied from other documents. Every document that can be paginated is capable of page manipulation.

For more about this capability, please see the The VectorPDFPageManipulation sample.

Page Manipulation Functionality

The Page Manipulation functionality can be used in the following scenarios:

- To remove pages, such as blank pages, duplicate pages, or pages irrelevant to the document.
- To add pages. Separator pages can be added between sections, or pages for notes which are subsequently written on using annotations.
- To reorder pages. If the original document's pages were scanned in the incorrect order, they can be rearranged. Also, the document's pages can be rearranged to place more relevant pages first.
- To copy pages. Similar to reorder pages, some pages may be copied and placed first in the document leaving the rest of the document in its original form.
- To rotate pages. In some cases, pages may be scanned with the wrong orientation. This allows the orientation to be fixed. A page in landscape orientation may be rotated within an otherwise portrait-oriented document. Please note that rotating pages changes searchable PDF pages into non-searchable rasterized PDF pages.
- To create new documents. New documents can be created from pages of other documents either copied or moved.

Please see the RasterMaster Java API documentation for detailed information about these methods.

Methods:

- `IMG_decompress_bitmap(String, int)`
- `IMGLOW_rotate_page(byte [], int, int )`
- `IMG_save_bitmap(int, int, int, int)`
- `IMGLOW_append_page(DataInputStream, byte[], int)`
- `IMGLOW_append_page(String, byte, int)`
- `IMGLOW_delete_page(String, int, int)`
- `IMGLOW_delete_page(DataInputStream, int, int)`
- `IMGLOW_detect_blank_page(int, char, int, int, int, int, char, char)`
- `IMGLOW_extract_page(java.io.DataInputStream, int, int)`
- `IMGLOW_extract_page(String, int, int)`
- `IMGLOW_get_pages(java.io.DataInputStream)`
- `IMGLOW_get_pages(String)`

Printing Images

This section describes how to print images using RasterMaster. The features includes altering the size of an image and image printing.

All Snowbound products print to any device with a valid Windows printer driver installed.

If a color or gray scale image is printed to a 1-bit or bi-level printer such as an HPLASERJET type printer, the image is dithered or reduced to 1-bit per pixel automatically.

The capabilities of the printer are detected by the Snowbound libraries to determine if the image must be dithered. The technique employed is Stucky error diffusion. This technique simulates grayscale by the placement of dots. The higher the resolution of the printer, the better the effect.

For examples on how to use the various printing methods included in the RasterMaster SDK, please see The Print sample and The SilentPrint sample. For the full method reference, please see the RasterMaster Java API documentation.

Printing Large Documents

Printing large documents, such as documents approaching or exceeding 100 pages, or documents set to a resolution of 300 DPI or above, requires additional memory resources. These additional resource requirements can affect the performance of both servers and clients.

Servers

Depending on the current Java Virtual Machine memory configuration for the Imaging server, the need for additional resources may cause an out of memory error. Increasing the Java Virtual Machine maximum heap size to the Imaging server during start up can help avoid memory errors. Specify the minimum and maximum heap size by passing the parameter -Xms and -Xmx to the Imaging server during start up, where the amount of memory is in megabytes or giga-bytes you want to allocate to the Imaging server.

Clients

Each of the different browsers handles the increased resource demands uniquely. In some cases, when printing documents that require additional resources, the document may print with blank pages, fail to respond, or require the browser or computer to be restarted.

Solution

A workaround for this problem may be to download the document locally and then print it. If the document does not have any annotations or the document is to be printed without annotations, the original document can be downloaded and printed. However, if a document is to be printed with annotations, a TIFF version of the document can be downloaded and printed.

Annotations

This section describes annotations for RasterMaster Java. The Viewing and Annotations sample is an image viewer that demonstrates the following image manipulations: altering the viewable size, changing the orientation, changing the size and resolution, making corrections, and flipping through pages of a multi-page file.

This sample also demonstrates several annotation methods, such as adding edits, bitmaps, lines, sticky notes, and more through a variety of methods. These annotations can be in both text and graphic format.

Overview of Annotations for the Java Platform

The Snowbound Software annotation toolkit is a Java class library for adding text, lines, highlights, and other annotations to bitmap images. The annotations are read and saved as a separate file so they can be added or edited independently of the bitmap image.

The display allows normal zooming, panning, and scrolling of the bitmap image with annotations drawn on top. The toolkit provides a number of built-in user interface methods for adding and editing annotations.

The Annotation toolkit works like the imaging library; simple Java methods are used to perform the functionality. The developer can layer the annotations. Currently, annotations may be placed on the image, read, written to a file, deleted, resized, and moved.

To create a new Annotation object or read from a file, start by using the standard SnowAnn constructor below.

```
SnowAnn Sann = new SnowAnn(simage.getWidth(), simage.getHeight());
```

Annotation Classes for the Java Platform

The annotation classes for the Java platform are listed below. For more information on the annotation classes, please see the RasterMaster Java API documentation.

- `Snow.SnowAnn`
- `Snow.ANN_GRAPHIC_STRUCT`
- `Snow.SANN_POINT`

- Snow.SANN_RECT

Overview of Built-In Annotation Editing

There are different user interface capabilities available when using built-in annotations. They are:

- Adding objects
- Editing the text in edit and Post-it objects
- Moving objects
- Resizing objects
- Deleting objects

See the sample DemoApp.java (Samples\com\snowbound\samples\annapp) for code which demonstrates how to add built-in annotation event handling to your application.

Using Built-In Annotation Editing

This section describes how to add, edit, move, resize, and delete annotation objects.

ADDING NEW OBJECTS

To add a new object, set the graphic_id parameter to one of the object numbers defined in Snow.SnowAnn .

Type	Object	Value
public static final int	SANN_FILLED_RECT	= 1
public static final int	SANN_HIGHLIGHT_RECT	= 2
public static final int	SANN_RECTANGLE	= 3
public static final int	SANN_LINE	= 4
public static final int	SANN_ELLIPSE	= 5
public static final int	SANN_FILLED_ELLIPSE	= 6
public static final int	SANN_FREEHAND	= 7

public static final int	SANN_BITMAP	= 8
public static final int	SANN_POSTIT	= 9
public static final int	SANN_POLYGON	= 10
public static final int	SANN_FILLED_POLYGON	= 11
public static final int	SANN_ARROW	= 12
public static final int	SANN_EDIT	= 13
public static final int	SANN_BUBBLE	= 15

CREATING ANNOTATION TEXT AND OBJECTS

Object	**Action or Result**
Edit and Post-it	Drag the mouse to draw a rectangle to the desired size. An editable text box appears when the mouse is released.
Most rectangles	Drag the mouse to draw a rectangle to the desired size.
Polygons	Mouse click at the vertices of each point for the polygon. A small box is drawn at the first point. The polygon is finished when the last point is on this box.

MOVING OBJECTS

To move an annotation object:

1. Left-click on the object to display a pop-up window.
2. Select **move**. A box is drawn around the object allowing you to position it elsewhere on the screen.
3. Left-click again to position the object where you want it.

RESIZING OBJECTS

To resize an annotation object:

1. Left-click on the object to display a pop-up window.
2. Select **resize**. The object is highlighted and boxes on the corners and middle of the top, bottom, left, and right edges are drawn.

3. Click and drag any of the boxes to resize the object.

DELETING OBJECTS

To delete an annotation object:

1. Left-click on the object to display a pop-up window.
2. Select **delete**. The object is removed from the page and from the internal list of objects.

Using Double Byte Characters with RasterMaster Java Annotations

Double byte characters such as Asian character sets are supported in the RasterMaster Java annotation library. To use double byte annotations, set the `double_byte` field to true as in the following example:

```
sann = new SnowAnn();sann.double_byte = true;
```

When creating any type of text annotation object, space will be reserved to allow for double byte characters. Double byte characters can be typed in the active text controls.

When saving out to an annotation file, the double byte characters will be preserved. For the XML annotation file format, a tag for double byte characters will also be written to the file. When reading in this file, the `double_byte` field will be read from the file and updated. If you are saving and reading the binary RasterMaster Java annotation file format, you will have to set the `double_byte` field to true to allow the library to read and save the file properly.

If you are having trouble using double byte characters, please see the Troubleshooting section.

Methods Used for Viewing and Annotations

The Viewing and Annotations sample is an image viewer that demonstrates the following image manipulations: altering the viewable size, changing the orientation, changing the size and resolution, making corrections, and flipping through pages of a multi-page file.

This sample also demonstrates several annotation methods, such as adding edits, bitmaps, lines, sticky notes, and more through a variety of methods. These annotations can be in both text and graphic format.

For information about the methods used in The Viewing and Annotations Sample, please see the RasterMaster Java API documentation.

Please see the RasterMaster Java API documentation for detailed information about these methods.

Methods:

- IMG_decompress_bitmap(String, int)
- IMG_deskew_bitmap(int)
- IMG_despeckle_bitmap(int)
- IMG_display_bitmap(java.awt.Graphics, int, int, int, int)
- IMG_display_bitmap_aspect(java.awt.Graphics,java.awt.Container, int, int, int, int, int)
- IMG_erase_rect(int,int,int,int,int,int,int)
- IMG_flip_bitmapx(void)
- IMG_flip_bitmapy(void)
- IMG_get_deskew_angle(int[], int, int)
- IMG_invert_bitmap(void)
- IMG_merge_block(Snow.Snowbnd, int, int, int)
- IMG_print_applet_pages(java.awt.Container,int)
- IMG_resize_bitmap(int, int)
- IMG_rotate_bitmap(int)
- IMG_save_bitmap(String, int)
- IMG_scroll_bitmap(java.awt.Container, java.awt.Event)
- IMG_window_level(int, int, int)
- SANN_read_ann(DataInputStream, int)
- SANN_set_coprect(int, int, int, int)
- SANN_display_annotations(java.awt.Graphics,java.awt.Container,int, int,int,int)
- GetClientRect(java.awt.Container, Snow.SANN_RECT)
- SANN_highlight_object(java.awt.Container, Snow.SANN_RECT, int)
- ui_delete(java.awt.Container)
- ui_startmove(java.awt.Container)
- ui_startresize(java.awt.Container)
- ui_wm_lbuttondown(java.awt.Container, int, int, byte[])
- ui_wm_mousemove(java.awt.Container, int, int, byte)
- ui_wm_lbuttonup(java.awt.Container, int, int, byte[])
- SANN_write_ann(byte[], int, byte[])

- `SANN_activate_all_objects(void)`
- `SANN_deactivate_all_objects(void)`
- `SnowAnn Sann = new SnowAnn(int, int);`
- `Snowbnd()`

Scanning

This section describes how to do scanning in RasterMaster Java. The Scanning sample gives a quick demonstration of how to use the scanning methods.

Using the Java Scanning Interface

The Java scanning interface is used in conjunction with Snowbound's RasterMaster Imaging SDK DLL. The file used is called scandll.dll. This file is the interface between the DLL and Snowbound's RasterMaster Java.

Installing the scandll.dll

The easiest way to put this in your path is to place the file in your \Windows\System32 directory. The RasterMaster native DLL should also be in your path.

Using the scanInterface.class for JNI Declarations

When writing a Java application that uses the scanning DLL, you must always use the `scanInterface` class. This makes the scanning calls to the native product. Since this has the JNI declarations of the scanning calls, these calls mirror the calls in the native library.

Please see the RasterMaster Java API documentation for detailed information about these methods.

Methods:

- `IMG_scan_acquire()`
- `IMG_scan_acquire_feeder()`
- `IMG_scan_feeder_close()`
- `IMG_scan_open_source()`
- `IMG_scan_get_source_list()`
- `IMG_scan_pages()`
- `IMG_scan_get_cap()`
- `IMG_scan_set_cap()`
- `IMG_scan_set_caps()`
- `IMG_scan_setup()`
- `IMG_get_scanned_image_bytes(int)`

- IMG_get_scanned_image(int)

Troubleshooting

This section describes solutions to issues that you may run across when installing and using RasterMaster Java.

Receiving an Error Code When Loading, Saving, or Converting a Document

If you receive an error code when trying to load, save or convert a document with RasterMaster Java, please check the list of error codes and their descriptions in Supported Error Codes to determine the source of the issue.

If you still cannot resolve the issue after looking at the list of error codes, please submit a support ticket at http://support.snowbound.com and include the error code and the document that you were trying to load, save or convert when you got the error message.

Output Document Differs from Original Document

When you convert a document and the output document is different from the original document, please create a support ticket at http://support.snowbound.com and attach the document along with a screenshot of the output document in RasterMaster Java. Some examples of the differences that may occur include missing data, missing text, distorted graphics or displaying an incorrect or different font. We can test the document with the latest release of RasterMaster Java which contains fixes which may resolve your issue.

Output Document Displays Incorrect or Missing Characters

When you convert a document, the output document may display incorrect or missing characters if the document contains special characters which are commonly found in Non-English languages such as Chinese, Japanese and Thai. Make sure that your system is properly configured to support these characters. For more information, please see Snowbound Software's Font Configuration Guide, available at https://snowbound.com/support/manuals.

Output Document Has Much Larger File Size than the Original Document

The file size of your output document may be much larger than the original document if you are converting or merging a PDF, Word document or other document into a raster image.

The Supported File Formats document (available on the Snowbound website) shows a list of file formats and their supported bit depths. Please see the following suggestions to reduce the file size:

Use one of the following methods prior to decompression to reduce the output dots per inch (DPI) or bit depth. For the full method reference, please see the RasterMaster Java API documentation.

- `IMGLOW_set_document_input(int dpi, int bits_pix, int format)` converts PDF, Word, Excel, PCL and AFP formats. Please see IMGLOW_set_document_input(int, int, int) for more information.
- `IMGLOW_set_pcl_input(int dpi, int bits_pix)` converts PCL formats only. Please see `IMGLOW_set_pcl_input(int, int)` for more information.
- `IMGLOW_set_pdf_input(int dpi, int bits_pix)` converts PDF formats only. Please see `IMGLOW_set_pdf_input(int, int)` for more information.

Use one of the following color reduction methods to reduce 8-bit and 24-bit images to smaller gray scale images. For the full method reference, please see the RasterMaster Java API documentation.

- `IMG_color_gray()` converts 24-bit color images to 8-bit gray scale images. Please see `IMG_color_gray(void)` for more information.
- `IMG_resize_to_gray(int xsize, int ysize)` resizes a 1-bit black and white image to a (smaller) 8-bit grayscale image. Please see `IMG_resize_to_gray(int, int)` for more information.
- `IMG_diffusion_mono()` converts 4, 8 or 24 bit images to 1-bit per pixel bi-level images using the Stucky error diffusion technique. Please see `IMG_diffusion_mono(void)` for more information.

Output Document Has Much Lower Quality than the Original Document

The first step in obtaining better quality conversions is to check the original images or documents in the application where they originated such as Adobe Acrobat or Microsoft Word.

If the original application is not available, try another viewer. If the quality is bad for the original image, then you may not be able to obtain any better quality in the conversion.

If this is a simple bitmap format such as TIFF, you can try adjusting the sharpening, contrast and brightness. To improve the quality of a 1-bit file format document, first convert it to gray scale using `IMG_promote_8()`.

If the source image or document is vector or it is composed of text and drawing commands such as PDF, Word, or AFP, you can set the conversion to be 1-bit or increase the dots per inch using `IMGLOW_set_document_input(Dpi, bits_pix, format)`. For the full method reference, please see the RasterMaster Java API documentation.

Please note that a higher resolution will almost always yield a better quality conversion but will probably result in a larger output file size.

Also it is common to convert to TIFF_G4. This is a 1-bit per pixel format. Make sure that if you are converting from a color image, that it is mostly text. There will be a noticeable loss in quality for pictures or color graphics such as logos when converting a color image to a 1-bit per pixel format. Text will usually work well.

If your document requires color, then try saving to a color output format such as TIFF_Packbits, TIFF_ JPEG or TIFF_LZW.

The output quality of your output document may be much lower than the original document if you convert a high quality image to a low quality output such as TIFF_G4_FAX. Please see the following suggestions to improve the quality of the output document:

Choose a different output format such as TIFF_LZW to produce an output document with sufficient pixel depth. Supported File Formats shows a list of file formats and their supported bit depths.

Use one of the following methods prior to decompression to increase the output dots per inch (DPI) or bit depth. For the full method reference, please see the RasterMaster Java API documentation.

- `IMGLOW_set_document_input(int dpi, int bits_pix, int format)` converts PDF, Word, Excel, PCL and AFP formats. Please see IMGLOW_set_document_input(int, int, int) for more information.
- `IMGLOW_set_pcl_input(int dpi, int bits_pix)` converts PCL formats only. Please see IMGLOW_set_pcl_input(int, int) for more information.
- `IMGLOW_set_pdf_input(int dpi, int bits_pix)` converts PDF formats only. Please see IMGLOW_set_pdf_input(int, int) for more information.

Use the color promotion method, `IMG_promote_24(void)`, to promote 1-bit, 4-bit and 8-bit images to 24-bit images.

Document Takes Too Long to Process

If your document takes a long time to process, use the HangDetector sample. This sample converts one specified document. If a document does not convert within three seconds, then this sample halts the conversion and prints a timeout message. Timeout speed can be adjusted to log in a way that works work best for you.

Identifying an Unknown File Format

To identify an unknown file format, you can use the `IMGLOW_get_filetype(String)` method in RasterMaster Java. This method returns a number that corresponds to a file type as defined in the RasterMaster® SDK library. Please see Supported File Formats for a list of the Snowbound file type constants.

Characters not Displaying Correctly with Smart Quotes in RTF and MS Word Documents

An issue may occur with documents created by Microsoft Word when using the "Smart Quotes" auto-correction feature. Microsoft will insert Unicode versions of quote, apostrophe, dash, and bullet characters. This displays well on Microsoft systems with the appropriate fonts, but the document tends to display with boxes for those characters on Unix because those Unicode characters are not in the fonts on that system. Also, some RTF and MS Word documents may word wrap incorrectly after an apostrophe. For example, the word member's may display as member at the end of a line and the s would display at the beginning of the next line.

The current version of the MS Office option for RasterMaster Java does not include the special handling needed to treat smart quotes correctly. A future version of RasterMaster Java should resolve this issue.

Until the issue is resolved in RasterMaster Java, you can resolve this issue by disabling "Smart Quotes" in Microsoft when creating documents. Specific instructions differ depending on your MS Word version. If you Google for disable Microsoft Word smart quotes, you will find how to do this for your version of MS Word.

To create a document that displays better on Unix systems, please follow the steps below:

1. Open the document in MS Word.
2. Make sure that Smart Quotes are disabled if they are not already.
3. Select the character that is not displaying properly or do a search replace/all for that character.

4. Insert the equivalent ASCII character using Insert >Symbol.
5. Save the document.

Receiving a `-3 Corrupted File` Error code

If you receive a `-3 corrupted file` error code, the input document may have become corrupt. To resolve this issue, open the document in an editor and write it back out again.

Some PDF document generators do not properly specify all of the information needed in a document. To resolve this issue in PDF documents that are generated by custom applications, open the document in Adobe Acrobat and then save it. You should then be able to process the newly saved document in RasterMaster Java.

Overlapping Scrollbar Arrow

When invoking the `IMG_display_bitmap_aspect` method with both vertical and horizontal scrollbars enabled, the scroll arrow for the scrollbars may overlap in some cases. You can use one of the following ways to work around this issue:

- Adjust the height and width of the window
- Adjust the zoom factor.
- Adjust the paint method used to display the bitmap and the scroll bars. Please see the sample Paint method below:

```
public void paint(Graphics x_g)

{

Dimension p_dimension; Insets p_insets;

if (g_clearFlag)

{

p_dimension = getSize(); p_insets = getInsets();

p_dimension.width -= (p_insets.right + p_insets.left); p_dimension.height -=
(p_insets.top + p_insets.bottom); x_g.setColor(getBackground());

x_g.fillRect(0, 0, p_dimension.width, p_dimension.height);
x_g.setColor(getForeground());
```

```
g_clearFlag = false;

}

if (g_snowbnd != null)

{

int p_stat = 0; p_dimension = getSize(); p_insets = getInsets();

x_g.translate(p_insets.left, p_insets.top); g_snowbnd.fit_to_width(0);
g_snowbnd.fit_to_height(0);

p_dimension.width -= (p_insets.right + p_insets.left); p_dimension.height -=
(p_insets.top + p_insets.bottom); boolean hsbBeforePaint = g_snowbnd.hsb !=
null;

boolean vsbBeforePaint = g_snowbnd.vsb != null; p_stat = g_snowbnd

.IMG_display_bitmap_aspect(x_g, this,

0,

0,

p_dimension.width, p_dimension.height, 75);

boolean hsbAfterPaint = g_snowbnd.hsb != null; boolean vsbAfterPaint =
g_snowbnd.vsb != null;

boolean scrollBarAddedInDisplay = (hsbAfterPaint && !hsbBeforePaint)

|| (vsbAfterPaint && !vsbBeforePaint); if (scrollBarAddedInDisplay)

{

/* paint again with new scollbar set up */ p_stat = g_snowbnd

.IMG_display_bitmap_aspect(x_g, this,

0,

0,

p_dimension.width, p_dimension.height, 75);

/* validate the container with the scroll bars */ validate();

}

if (p_stat < 0)
```

```
{
System.out.println("Error display bitmap aspect: stat = "
+ p_stat);
}
}
super.paintChildren(x_g);
}
```

Overlay Resources Not Pulled into APF or MODCA Document

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

Searching for Text in a Snowbound Software Generated PDF

If you are having trouble finding text in a Snowbound Software generated searchable PDF, please use Adobe Reader version 9.4 to do the search.

Yellow, Red or Other Light Colored Content Disappear When Converting to Black and White Output

If yellow, red or other light colored content disappear when converting to black and white output, adjust the threshold. Increasing the threshold value will cause lighter content to be preserved during the conversion. To remove light-colored artifacts, the threshold value should be decreased.

Some TIFF_JPEG Files Produced By RasterMaster Java Do Not Open In Third Party Image Viewers

When converting a file to TIFF_JPEG with RasterMaster Java, you may see that the resulting TIFF is blank when opened in a third party image viewer such as MS Office Document Imaging or Windows Photo Viewer. In some cases, the viewer may also display an error message indicating that the TIFF is corrupted or too large.

This issue typically occurs when the file in question is converted at 1-bit black and white or 8-bit grayscale, rather than 24-bit color. By default, RasterMaster Java will often choose 1 bit/pixel to ensure better performance during conversion. Formats which are converted to 1-bit by default include AFP, DOC, XLS, and PCL. For these formats, we recommend that you call the `IMGLOW_set_document_input()` method prior to decompression if you wish to convert to TIFF_JPEG. Please see the following example:

```
status = s.IMGLOW_set_document_input(200, 24, Defines.DOC) status =
s.IMG_decompress_bitmap("C:\\input\\file.doc", page); status =
s.IMG_save_bitmap("C:\\output\\file.tif", Defines.TIFF_JPEG)
```

In the above example, the input DPI and bit-depth are set to 200 and 24 respectively for the input DOC file before being converted to TIFF_JPEG.

Please also note that while color input images (e.g. JPEG, TIFF, PNG) will be converted to 24 bits/pixel by default, you still might see this issue for black and white or grayscale input images. In the case where your input file is a black and white or grayscale image, you can promote the bit-depth to 24 by calling `IMG_promote_24()` after decompression. Please see the following example:

```
status = s.IMG_decompress_bitmap("C:\\input\\file.png", page); status =
s.IMG_promote_24();

status = s.IMG_save_bitmap("C:\\output\\file.tif", Defines.TIFF_JPEG)
```

This should produce an output TIFF that can be viewed by most third party image viewers.

Converting Text to JPEG on Lower Resolution Screen

JPEG is not the best compression to use for converting text. JPEG works for pictures and not images with hard black to white transitions. For display on a lower resolution screen and to see

better results when zoomed, try resizing the image with `IMG_create_thumbnail(int, int)`, then convert to JPEG.

XLS or XLSX Page Content Truncated

Your XLS or XLSX page content may be truncated because XLS and HTML formats do not include the page size in the document like Word and PDF. It can be set explicitly similar to how you can set the page size when printing. To set the page size to avoid truncated content, use the `IMGLOW_set_document_input (int dpi, int bits_pix, int format , int width, int height)` method as shown in the example below.

To explicitly set the page size to US Letter 8.5"x11", use:

```
IMGLOW_set_document_input(300, 24, Defines.EXEL, 8.5, 11.0)
```

If you have more than one XLSX file open in the evaluation version of RasterMaster Java, you may get a `-38 EXCEPTION_ERROR` error.

The workaround is to only open one XLSX document at a time in the evaluation version.

Getting a `ClassNotFoundException` Error

A return status of -38 EXCEPTION_ERROR indicates an exception was thrown. This usually includes a stack trace with information about what caused the exception. If the stack trace includes `java.lang.NoClassDefFoundError`: this indicates the issue is a missing .jar file. The name of the class following `java.lang.NoClassDefFoundError`: can be used to determine which .jar file cannot be found. Check your java CLASSPATH carefully to ensure the directory containing the .jar is in the path. Please feel free to contact Snowbound Support at support.snowbound.com for help determining which .jar is missing.

Getting Different Results When Using the Same Version of RasterMaster Java and the Same Document on Different Systems

The following are the two common reasons for different results when using the same version of RasterMaster Java and the same document on different systems:

1. You may have different fonts configured and available for Java. Make sure that your system is properly configured to support the fonts that you are using. For more information, please see Snowbound Software's Font Configuration Guide.
2. You may have different versions of Java installed. Please contact Snowbound support at http://support.snowbound.com for solutions if your system configurations are identical and you still see a significant difference.