Space Details

Key:	PDF
Name:	ICEpdf
Description:	This is the Wiki home for the ICEpdf opensource project. See www.icepdf.org for details.
Creator (Creation Date):	ken.fyten (Dec 10, 2009)
Last Modifier (Mod. Date):	carlo.guglielmin (Apr 15, 2010)

Available Pages

ICEpdf Developer's Guide

This page last changed on Feb 22, 2013 by ken.fyten.

Contents

This guide contains the following sections:

Introduction to ICEpdf

This page last changed on Jan 27, 2010 by mark.collette.

ICEpdf® is a pure Java PDF document rendering and viewing solution. ICEpdf can parse and render documents based on the latest PDF standards (Portable Document Format v1.6/Adobe® Acrobat® 7) with superior rendering accuracy and performance.

ICEpdf is designed to support PDF document viewing within Java applications in a manner not possible with the native Acrobat Reader® application.

Benefits include:

- Seamless integration with Java client applications, allowing complete control over the configuration, exposed functionality and user interface.
- A lightweight static and dynamic memory footprint.
- Easy deployment to any Java platform without the hassles of Java-to-native integration issues.

ICEpdf supports:

- **PDF Viewing:** ICEpdf can easily be integrated into any Java client application to provide PDF document viewing and navigation in a manner not possible with the Acrobat Reader application. ICEpdf includes an embeddable PDF document viewer component for easy integration within Java client applications. ICEpdf can also be used standalone as an industrial strength PDF Viewer application.
- **Multipage view support**: Continuous and side-by-side view types.
- **Text Selection**: Multi-page text selection tool allows users to select and copy text to the system clipboard.
- **Search Highlighting**: Advanced contextual search result and highlighting of found words.
- **PDF Content Conversion:** Convert rendered PDF pages to other formats, such as images, SVG documents, etc.
- **PDF Content Extraction:** Extract PDF document meta-data, text, and images.
- **PDF Link Annotations:** Developers can optionally configure ICEpdf to support interactive link annotations via a mouse. An annotation callback gives developers flexibility in which types of link annotation actions they wish to support.
- **PDF Link Annotation Editing:** Users can now configure the UI to support creation, editing and deletion of Link annotations and their respective URI, Launch or GoTo actions.

Download Bundle Contents

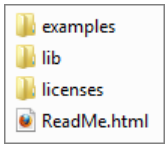
This page last changed on Jan 28, 2010 by patrick.corless.

If you are reading this document, you may have already downloaded and installed ICEpdf. If you haven't, you can get the latest version of ICEpdf from:

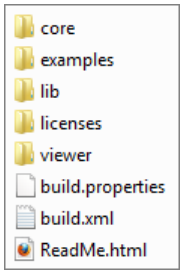
http://www.icepdf.org

ICEpdf is available in both a binary and source-code distribution. With either distribution, begin by unzipping ICEpdf to your preferred location.

If you downloaded the binary distribution, the resulting directory structure should look similar to the structure shown below:



If you downloaded the source distribution, these additional source-code directories will also be present:



Render Core (core)

This directory contains the ICEpdf rendering core source files and Ant build script to build the source.

Examples (examples)

This directory contains examples of common usage scenarios. Further information on these examples can be found in **Common Usage Scenarios**

Libraries (lib)

This directory contains the Java archive (JAR) files which are needed to compile and run ICEpdf.

Jar File	Approximate Size	Description
icepdf-core.jar	703 KB	Contains the core ICEpdf product classes.
icepdf-pro.jar*	127 KB	Professional font library that allows for the reading of embedded font files in PDF documents
icepdf-pro-intl.jar*	2,949 KB	Optional JAR file that contains character collection for predefined CMaps for Chinese, Chinese (Simplified and Traditional), Japanese and Korean. Must be added to the classpath to fully support these languages.

icepdf-viewer.jar	378 KB	This executable JAR file contains the standalone ICEpdf Viewer reference implementation.

*Denotes JAR files that are only available in ICEpdf Pro.

ICEpdf compilation and runtime dependencies:

Jar File	Approximate Size	Description
batik-*.jar	1,100 KB	Optional Batik SVG libraries used for saving PDF pages to SVG format.

Version and license information is described in versions-licenses.html

Viewer Reference Implementation (viewer)

This directory contains the source code for the reference implementation and an Ant build script to build the source.

Supported PDF Features

This page last changed on Jul 28, 2010 by patrick.corless.

ICEpdf supports the following PDF features

- Font support: Embedded font support for Type 1 Fonts (Standard and Multiple Master), TrueType, Font Subsets, Type3, CMaps (predefined and Embedded), Type 0 CID, Type 2 CID, Type 0, Type 1 (CFF), OpenType (True Type Outlines) and OpenType (CFF Type outlines). Font substitution is available for documents that do not use embedded fonts.
- Cross-Reference Table and Cross-Reference Stream support for accelerated document loading.
- Multiple page views: single page, facing page, single page column, and facing page columns.
- Rendering of AcroForm data, push buttons, check boxes, radio buttons, text fields and choice fields.
- Rendering of common Annotation types: markup and text markup; text, free text, and line; square; circle polygon; and polyline types.
- Create, edit and delete Link annotations and their respective URI, Launch or GoTo actions.
- Interactive Link annotation via the following actions types: go to actions, go to resource actions, go to launch actions and URI actions.
- Multi-page text selection.
- Converting rendered PDF pages to images, SVG documents, etc.
- Extracting PDF document meta-data, text, and images.
- PDF document viewing.
- Page navigation.
- Page magnification.
- Page rotation.
- Printing.
- Bookmarks (table of contents entries that represent the chapters and sections in a document).
- Search document text and highlight results.
- Acrobat standard security (40-bit and 128-bit RC4 encryption) for opening password-controlled files (for more information, see **Acrobat Standard Security Support**.
- ICEbrowser PDF Pilot: Extend ICEbrowser® to support PDF document rendering using the included ICEbrowser PDF Pilot (Plugin) component.

PDF Reference Compliance

ICEpdf supports a subset of the PDF Reference, 5th Edition, Version 1.6 from Adobe Systems Incorporated, available at:

http://www.adobe.com/devnet/pdf/pdf_reference.html

The table below lists all the PDF features that ICEpdf supports. The list is based on the PDF Reference, 5th Edition, Version 1.6, from Adobe Systems Incorporated. The section numbers in the table refer to the sections in the PDF Reference.

You can download the reference from: **http://www.adobe.com/devnet/pdf/pdf_reference.html**

Support Feature			Section in PDF Referen	Introdu 1.0	in 1.1	PDF 1.2	Version 1.3	1.4	1.5	1.6
Filters										
ASCIIHexDecode			3.3.1	**X**						
ASCII85Decode			3.3.2	**X**						
LZWDecode			3.3.3	**X**						
FlateDecode			3.3.3			**X**				
RunLengthDecode			3.3.4	**X**						
CCITTFaxDecode										

	Group 4		3.3.5	X						
	Group 3, 1-D		3.3.5	X						
	Group 3, 2-D		3.3.5	X						
JBIG2Decode			3.3.6					X		
DCTDecode (No transformations)			3.3.7	X						
JPXDecode			3.3.8						X	
File Structu										
File Body			3.4.2	X						
Cross-Reference Table			3.4.3	X						
File Trailer			3.4.4	X						
Incremental Updates			3.4.5					X		
Object Streams			3.4.6						X	
Cross-Reference Stream			3.4.7						X	
Encrypt										
Standard Security Handler			3.5.2		X					
Docume Structu										
Document Catalog			3.6.1		X					
	Page Layout		3.6.1							
		Singe Page View	3.6.1	X						
		One Column View	3.6.1	X						

		Two Column Left View	3.6.1	**X**						
		Two Column Right View	3.6.1	**X**						
		Two Page Left View	3.6.1						**X**	
		Two Page Right View	3.6.1						**X**	
Page Tree			3.6.2		**X**					
	Page Objects		3.6.2		**X**					
	Inheritance of Page Attributes		3.6.2		**X**					
Functions			3.9			**X**				
	Type0 (Sampled) Functions		3.9.1			**X**				
	Type2 (Exponential Interpolation) Functions		3.9.2			**X**				
	Type3 (Stitching) Functions		3.9.3			**X**				
Graphic										
Graphics States										
	Device Independent									
		Line Width	4.3.2	**X**						
		Line Cap Style	4.3.2	**X**						

		Line Join Style	4.3.2	**X**						
		Miter Limit	4.3.2	**X**						
		Line Dash Pattern	4.3.2	**X**						
		Alpha Constant	4.3.2					**X**		
Path Construction										
	Cubic Bézier Curves		4.4.1	**X**						
	Sub Paths		4.4.1	**X**						
	Lines		4.4.1	**X**						
	Rectangles		4.4.1	**X**						
Path-Painting Operators										
	Stroking		4.4.2	**X**						
	Filling		4.4.2	**X**						
		Nonzero Winding Number Rule	4.4.2	**X**						
		Even-Odd Rule	4.4.2	**X**						
Clipping Path Operators			4.4.3	**X**						
Color Spaces										
	Device Color Spaces									
		Device Gray	4.5.3		**X**					
		Device RGB	4.5.3		**X**					

		Device CMYK	4.5.3		**X**					
	CIE-Based Color Spaces									
		ICCBased Color Spaces	4.5.4				**X**			
	Special Color Spaces									
		Pattern Color Spaces	4.5.5			**X**				
		Indexed Color Spaces	4.5.5		**X**					
		Separation Color Spaces	4.5.5			**X**				
		DeviceN Color Spaces	4.5.5				**X**			
Patterns			4.6							
	Tiling		4.6.2			**X**				
	Shading		4.6.3				**X**			
External Objects			4.7	**X**						
	Image XObjects		4.7	**X**						
	Form XObjects		4.7	**X**						
Images										
	Decode Arrays		4.8.4	**X**						
	Image Interpolation		4.8.4	**X**						
	Masked Images									
		Stencil Masking	4.8.5	**X**						
		Explicit Masking	4.8.5				**X**			

		Color Key Masking	4.8.5				X			
	Inline Images		4.8.6	X						
	Form XObjects									
		Form Dictionaries	4.9.1	X						
Text										
Text State Parameters										
	Character Spacing		5.2.1	X						
	Word Spacing		5.2.2	X						
	Horizontal Scaling		5.2.3	X						
	Leading		5.2.4	X						
	Text Rise		5.2.6	X						
Simple Fonts*										
	Type 1 Fonts*		5.5.1							
		Standard Type 1 Fonts*	5.5.1	X						
		Multiple Master Fonts*	5.5.1	X						
	TrueType Fonts*		5.5.2	X						
	Font Subsets*		5.5.3	X						
	Type 3 Fonts*		5.5.4	X						
Composite Fonts*										
	CIDFonts*		5.6.3							
		Predefined Cmaps*	5.6.4				X			

		Embedded Cmap Files*	5.6.4				X			
		Type 0 CIDFonts*	5.8				X			
		Type 2 CIDFonts*	5.8				X			
Font Descriptors			5.7	X						
Other Font Programs*										
	Type 0 Fonts*		5.8							
	Type 1(CFF)*		5.8			X				
	OpenType (True Type outlines)*		5.8							X
	OpenType (CFF Type outlines)*		5.8							X
Interac Feature										
Viewer Preferences										
	Hide Tool bar		8.1	X						
	Hide Menu bar		8.1	X						
	Fit Window		8.1	X						
	Center Window		8.1	X						
	Display Document Title		3.1					X		
	Document Page Mode		8.1	X						
		Outlines	8.1	X						

		Optional content group	8.1	**X**						
	Print Scaling		8.1							**X**
Document-Level Navigation										
	Destinations		8.2.1	**X**						
	Document Outline		8.2.1	**X**						
	Link Annotation		8.4.5	**X**						
		Go To actions	8.5.3	**X**						
		Go to resource actions	8.5.3	**X**						
		Go to launch actions	8.5.3	**X**						
		URI actions	8.5.3	**X**						
Annotations†			8.4	**X**						
	Annotation Flags		8.4.2		**X**	**X**	**X**			
	Border Styles		8.4.3			**X**				
	Appearance Streams		8.4.4			**X**				
	Annotation Types		8.4.5							
		Markup Annotations	8.4.5	**X**						
		Annotation States	8.4.5						**X**	
		Text Annotations	8.4.5	**X**						
		Free Text Annotations	8.4.5				**X**			
		Line Annotations	8.4.5				**X**			

		Square and Circle Annotations	8.4.5				**X**			
		Polygon and Polyline Annotations	8.4.5						**X**	
		Text Markup Annotations	8.4.5				**X**			
Interactive Forms†			8.6							
	Button Fields		8.6.3							
		Push Buttons	8.6.3			**X**				
		Check boxes	8.6.3			**X**				
		Radio Buttons	8.6.3			**X**				
	Text Fields		8.6.3			**X**				
	Choice Fields		8.6.3			**X**				

*** ICEpdf Pro version only.**
† Static rendering only.

Supported Platforms

This page last changed on Dec 10, 2009 by ken.fyten.

ICEpdf requires a compliant Java Virtual Machine (JVM) that supports Java 2D and JFC (Swing). Typically, any J2SE platform, version 1.5.0 or greater, meets these requirements. The officially supported platform and JVM combinations are:

Windows

- Sun JDK 5.0
- Sun JDK 6.0
- Sun JDK 7.0 EA

Linux

- Sun JDK 5.0
- Sun JDK 6.0
- Sun JDK 7.0 EA

Solaris

- Sun JDK 5.0
- Sun JDK 6.0
- Sun JDK 7.0 EA

Mac OS X

- Apple JDK 5.0
- Apple JDK 6.0

Visit **http://www.icepdf.org** for more information.

Configuring ICEpdf

This page last changed on Jan 28, 2010 by patrick.corless.

This chapter contains instructions to help you get up and running quickly with ICEpdf. We start by outlining the prerequisites for a standard configuration using a Java Platform, Standard Edition, and Apache Ant to help you build the product from source.

Next we outline runtime configuration settings to best meet the needs of your application requirements and deployment environments:

- Configuration of optional modules at runtime, such as support for the Acrobat Standard Security, optimized font substitution, embedded font support, Java Advanced Imaging (JAI) libraryand SVG support.
- Numerous configuration properties may be adjusted using pre-defined system properties to alter the behavior of ICEpdf for your application, such as cache sizes and behaviors, render quality, settings, etc.
- Viewer Reference Implementation
- ICEpdf Examples

Prerequisites

ICEpdf is a standard Java 2 application, and as such, the only prerequisite to working with ICEpdf is that you must be familiar with Java 2 development. For more information on the Java Platform, Standard Edition (J2SE), refer to **http://java.sun.com/javase/index.jsp**.

To run the ICEpdf example and reference applications, you will need to download and install the following:

- Java 2 Platform, Standard Edition, version 1.5 or higher
- Apache Ant

The following sections provide detailed instructions for downloading the software to set up an environment where you can run the ICEpdf example and reference applications.

Java 2 Platform, Standard Edition

To run ICEpdf, ICEpdf reference implementations or examples, you will need to install a version of the Java Platform JDK, Standard Edition, version 1.5 or higher.

If you already have Java installed on your system, verify your version by typing the following on the command line:

```
java -version
```

To upgrade or install the latest release of the J2SE, visit the Sun web site:

http://java.sun.com/javase/downloads/

Installers and instructions are provided for the various systems that Sun supports. The reference and example application can run on any version of Windows, Unix, Linux, and Mac OS X capable of running J2SE version 1.5 or higher.

Ant

The ICEpdf core, reference applications and examples rely on Ant to build the source code. You will need Ant version 1.6.3 or higher for the build files provided in this ICEpdf release.

If you already have a version of Ant installed, you can verify that you have a recommended version by typing the following on the command line:

```
ant -version
```

To upgrade your current version or install a new version of Ant, visit the following location:

http://ant.apache.org/

If you are not familiar with Ant, detailed instructions for downloading and installing Ant for your environment are available from this location:

http://ant.apache.org/manual/index.html

Building ICEpdf from Source

This page last changed on Dec 10, 2009 by ken.fyten.

ICEpdf is bundled with a PDF viewer reference implementation. If you downloaded the binary distribution of ICEpdf, the viewer application is available as prebuilt JAR file found in **[install_dir]/icepdf/lib/icepdf-viewer.jar**.

If you have downloaded the source code distribution of ICEpdf, it is necessary to build the ICEpdf core and viewer source code using the provided build scripts. The Ant help command can be used to see a list of available build targets. To build the ICEpdf core and viewer JARs simply execute the command **ant** in the **[install_dir]/icepdf/** directory.

This command will build and copy the icepdf-core.jar and icepdf-viewer.jar to the **[install_dir]/icepdf/lib/** directory.

Customizing the Viewer

This page last changed on Jan 26, 2010 by patrick.corless.

Customizing the Viewer RI

The viewer Reference Implementation can be configured using properties defined at runtime. These properties allow for easy manipulation of the Viewer RI functionality without having to modify and compile the viewer source code.

When the Viewer RI is run for the first time a small text file is created in the user home directory. This file pdfviewerri.properties, stores the configuration options for the application. This file can be optionally overridden by using the application switch "-loadproperties path" to load a properties file with different default values.

Summary of Configuration Properties.

Property	Type	Description
General		
application.viewerpreferences.hidemenubar	boolean	Hides the menubar. Default value false.
application.viewerpreferences.hidetoolbar	boolean	Hides the top toolbar. Default value false.
application.viewerpreferences.fitwindow	boolean	When enabled shows the document with fit to width if the document does not already specify a default view. Default value, true.
application.menuitem.show.keyboardshortcuts	boolean	Enables/disables menubar key events, default is true.
application.alwaysShowImageSplashWindow	boolean	Enables/disables the splash window, default is no
application.chromeOnStartup	string	Start with with chrome look and feel, default value is yes.
application.showLocalStorageDialogs	string	Yes is default value. Show dialog to use before writing properties to disk.
application.datadir	string	Directory to write properties file to in users home, default .icesoft/icepdf-viewer.
Toolbar - General		
application.toolbar.show.pagenav	boolean	Show page navigation controls; first, previous, next and last. Default is true.
application.toolbar.show.fit	boolean	Shows page fit controls; normal, hight and width. Default is true.
application.toolbar.show.zoom	boolean	Shows page zoom controls. Default is true.

application.toolbar.show.rotate	boolean	Shows page rotation controls. Default is true.
Toolbar - Zoom		
application.zoom.factor.default	number	Float value of default page zoom. Default if 1.0
application.zoom.range.default	string	Comma separate list of float values of zoom combo box. Default value is 0.05,0.1,0.25,0.5,0.75,1.0,1.5,2.0,3.0,4.0,8.0
Toolbar - Utility		
application.toolbar.show.utility	boolean	show utility toolbar and children. Default is true.
application.toolbar.show.utility.save	boolean	Show the save control, default is true.
application.toolbar.show.utility.open	boolean	Show the open control, default is true.
application.toolbar.show.utility.search	boolean	Show the search control, default is true.
application.toolbar.show.utility.upane	boolean	Show the utility pane control, default is true.
application.toolbar.show.utility.print	boolean	Show the print control, default is true.
Toolbar - Tools		
application.toolbar.show.tool	boolean	Show the tools toolbar and children. Default is true.
application.toolbar.show.annotation	boolean	Show the annotation toolbar and children. Default is true.
Utility Pane		
application.utilitypane.show.bookmarks	boolean	Show the document outline panel tab, default is true.
application.utilitypane.show.search	boolean	Show the search panel tab, default is true.
application.utilitypane.show.annotation	boolean	Show the annotation properties pane, default is true.
Status Bar		
application.statusbar	boolean	Show the status bar and child component, default is true.
application.statusbar.show.statuslabel	boolean	Show the status label, default is true.
application.statusbar.show.viewmode	boolean	Show the view mode controls, default is true.

Logging

This page last changed on Feb 02, 2010 by patrick.corless.

How to Setup Logging

ICEpdf uses the Java Util Logging system. The following is a snippet from the ./lib/logging.properties which is our default logging configuration file.

```
# Specify the handlers to create in the root logger
# (all loggers are children of the root logger)
# The following creates two handlers
handlers = java.util.logging.ConsoleHandler, java.util.logging.FileHandler

# Set the default logging level for the root logger
.level = ALL

# Set the default logging level for new ConsoleHandler instances
#java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.level = INFO

# Set the default logging level for new FileHandler instances
java.util.logging.FileHandler.level = INFO

# Set the default formatter for new ConsoleHandler instances
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

org.icepdf.core.pobjects.fonts.level = INFO
org.icepdf.level = INFO

javax.level = OFF
java.level = OFF
win.level = OFF
sun.level = OFF
awt.level = OFF
```

Use the following system property to enable the logging file when developing with ICEpdf.

```
-Djava.util.logging.config.file="c:/svn/ossrepo/icepdf/trunk/icepdf/lib/logging.properties"
```

To see all of ICEpdf debugging output change the following value java.util.logging.ConsoleHandler.level from **INFO** to **FINEST**.

Optional Components

This page last changed on Dec 10, 2009 by ken.fyten.

This section provides details for the following optionally supported components:

- Optimized Font Substitution
- Embedded Font Support
- Acrobat Standard Security Support
- Java Advanced Imaging (JAI) Library for Enhanced Image Support
- Batik Library for SVG Support

Acrobat Standard Security Support

This page last changed on Jan 27, 2010 by mark.collette.

Acrobat standard security (40-bit and 128-bit RC4 encryption) includes password protection and setting change and display permissions, such as printing and content extraction, for a PDF file.

> **Useful Information**
> As of JDK 1.5, the Bouncy Castle jars are no longer needed for ICEpdf to support Adobe Standard Security. If you prefer the Bouncy Castle Cryptography libraries over the standard JRE implementation, they can be downloaded here.

If you use a different JCE 1.2.1-compliant security provider, you must set the system property **org.icepdf.core.security.jceProvider** appropriately. See **System Properties** for details.

> **Warning**
> You must also code your application to respect the security settings. For example, you must disable printing if a PDF file is set with "No Printing" permissions. The ICEpdf Viewer reference implementation has been coded to respect security permissions, and you can use its source code as a model for your own application.

Optimized Font Substitution

This page last changed on Jan 24, 2011 by patrick.corless.

Some PDF documents do not embed the fonts required to render the document in the document itself. For ICEpdf to display these documents accurately, it must substitute an available system font for any non-embedded fonts used in the document. The following optional but recommended configurations can be used to improve ICEpdf's ability to find and select a system font that best matches the non-embedded font specified by the PDF document.

Supporting the 14 Standard Adobe Fonts

To enable the best possible font substitution for the 14 standard Adobe fonts, ICEsoft recommends that you download and install the fonts provided by Ghostscript, which are freely available at:

http://prdownloads.sourceforge.net/gs-fonts/ghostscript-fonts-std-8.11.tar.gz

http://prdownloads.sourceforge.net/gs-fonts/ghostscript-fonts-other-6.0.tar.gz

The steps to install the fonts are platform-specific:

Windows

1. Unzip the download file and extract the **.pfm** files to a temporary directory.
2. Launch Fonts from the Control Panel.
3. Select **File > Install New Font**.
4. Navigate to the temporary directory, select the fonts, and click **OK**.

Other Platforms
For Linux, Solaris, or Mac OS X platforms, see your system documentation for information on installing the fonts.

- The GhostScript fonts include fonts for different operating systems and when installing them on Windows or other platforms you may be presented with an error dialogue that some font did not correctly install. This is normal, fonts that could be installed will be installed regardless of the message.
- Once new system fonts have been stalled it is important to delete the ICEpdf font cache file to insure that ICEpdf will rescan the system for valid fonts the next time it is run. Delete the file ~/.icesoft/icepdf-viewer/pdfviewerfontcache.properties to clear the font cache.

Embedded Font Support

This page last changed on Jan 27, 2010 by mark.collette.

ICEpdf Open Source

ICEpdf Open Source uses java.awt.Font when reading system font files for substitution. ICEpdf Open Source, by default, disables using java.awt.Font for reading embedded font files, because a malformed font file can crash the JVM. The system property `org.icepdf.core.awtFontLoading=true` can be set to enable java.awt.Font embedded font loading.

ICEpdf Pro

ICEpdf Pro allows for unprecedented font reproduction and rendering speed. ICEpdf Pro is a commercial product and requires `icepdf-pro.jar` and `icepdf-pro-intl.jar` libraries are on the application class path.The following embedded font types are supported:

•	Type 0	•	Type 1
•	Type 2	•	Type3
•	TrueType	•	OpenType

If your application requires support for Asian languages such as Chinese (Simplified and Traditional), Japanese, Korean, you should add the optional JAR `icepdf-pro-intl.jar` to your application classpath to ensure maximum rendering quality and accuracy of the respective language's characters.

For more information on ICEpdf Pro, visit **http://www.icepdf.org**.

Java Advanced Imaging (JAI) Library for Enhanced Image Support

This page last changed on Jul 28, 2010 by patrick.corless.

If you want to be able to view JPEG2000, CCITTFax Group 3 1-D and Group 3 2-D images in ICEpdf, you require the optional JAI library, which is available at **https://jai.dev.java.net/**

More detailed installation instruction scan be found **http://java.sun.com/products/java-media/jai/INSTALL-jai_imageio_1_0_01.html**

JAI also provides more robust image handling of JPEG (DCTDecode) images, which the Java SDK library cannot always decode correctly.

JAI is available for Windows, Linux, Mac OS X, and Solaris, and takes advantage of native acceleration when available. However, you can use the JAI libraries on any platform if you add the following JAR files to your classpath: `jai_core.jar`, `jai_codec.jar`, `jai_imageio.jar`

Batik Library for SVG Support

This page last changed on Dec 10, 2009 by ken.fyten.

If you want to export PDF files as SVG files in the ICEpdf Viewer, you need the Batik SVG library, which is included in the libs folder of the bundle. The following jars must be on the class path:

- **batik-awt-util.jar**
- **batik-dom.jar**
- **batik-svg-dom.jar**
- **batik-svggen.jar**
- **batik-util.jar**
- **batik-xml.jar**

System Properties

This page last changed on Jul 28, 2010 by patrick.corless.

Many system properties are available for configuring ICEpdf. They can be set programmatically or on the command line. Programmatically, the syntax is as follows:

```
System.getProperties().put("org.icepdf.core.minMemory", "3M");
```

On the command line, the syntax is as follows:

```
java -Dorg.icepdf.core.minMemory=3M ...
```

The **Dynamic** column indicates whether changing the value of the property at runtime has any effect. The possible values are:

- No - has no effect at runtime
- Yes - always has effect at runtime
- new **<*class**>* - a new instance of the class must be created to see the effect at runtime
- N/A - not applicable

Property	Type	Description	Dynamic
General			
org.icepdf.core.security.jceProvider	String	Specifies the classname of the security provider to use for encrypted documents. The provider must be Sun Java JCE 1.2.1 compliant. Default value is **org.bouncycastle.jce.provider.BouncyCastleProvider**.	No
Memory Management			
org.icepdf.core.minMemory	string	Sets the amount of Java heap memory to reserve as a safety buffer to prevent OutOfMemoryExceptions from occurring. If the amount of used Java heap memory is greater than (Max Memory - org.icepdf.core.minMemory), the memory manager will flush pages in the page cache until the required amount of Java heap memory is available. The default is 5MB.	No
org.icepdf.core.maxSize	integer	Specifies the maximum number of pages that should be cached at	No

		one time. The default is 0, which results in as many pages as will fit in the available memory being cached. A value of 1 effectively disables the page cache.	
org.icepdf.core.purgeSize	integer	Sets the number of pages that are purged from the page cache each time the memory manager attempts to free memory. If the MemoryManager detects that a considerable percentage of time is being spent purging pages, it will dynamically and temporarily raise this value. Default value is 5 pages.	No
Caching			
org.icepdf.core.imagecache.enabled	boolean	If true, images are cached to the hard drive. This saves memory, and the cost of execution time. The default value is true.	No

Note: For the Rendering Quality properties below, **target** can be set for both **print** and **screen**. For dynamic changes to these System Properties to take effect, you must call `org.icepdf.core.util.GraphicsRenderingHints.reset()`.

Property	Type	Description	Dynamic
Rendering Quality			
org.icepdf.core.awtFontLoading	boolean	When enabled the java.awt.Font will be used to try and load embedded font files. Default value is false. Has no effect on ICEpdf Pro.	No
org.icepdf.core.scaleImages	boolean	If true, images with a pixel width greater than 1000 will be scaled down, to improve memory usage. The dynamic scaling does its best to preserve image quality, but text clarity can be impacted. Default value is true.	No

org.icepdf.core.paint.disableAlpha	boolean	If true, all alpha or transparency painting will be suspended. This property can be enabled when printing with in the intension of reducing the spool size.	Yes
org.icepdf.core.paint.disableClipping	boolean	If true, all clipping will be suspended. This property should only be used if content is missing from printed output. This property can be turned on for printing and turned off once the print job has finished.	Yes
org.icepdf.core.**target**.alphaInterpolation	string	Sets the JVM's alpha interpolation rendering hint. The default value for **print** is **VALUE_INTERPOLATION_QUALITY**. The default value for **screen** is **VALUE_INTERPOLATION_QUALITY**. The other supported value is **VALUE_ALPHA_INTERPOLATION_DEFAULT**	Yes
org.icepdf.core.**target**.antiAliasing	string	Sets the JVM's antialiasing of all images and text. The default value for **print** and **screen** is **VALUE_ANTIALIAS_ON**. Other supported values are **VALUE_ANTIALIAS_DEFAULT** and **VALUE_ANTIALIAS_OFF**.	Yes
org.icepdf.core.**target**.background	string	Sets whether a Page will draw a background fill color before drawing the Page contents. According to the PDF standard, a white background should be drawn. When printing on white paper, for some printers with poor drivers, it is best to not draw a background at all. The default value is **VALUE_DRAW_WHITE_BACKGROUND**. The other supported value is **VALUE_DRAW_NO_BACKGROUND**.	Yes

org.icepdf.core.**target**.colorRender	string	Sets the JVM's color render rendering hint. The default value for **print** and **screen** is **VALUE_COLOR_RENDER_QUALITY**. The other supported values are **VALUE_COLOR_RENDER_DEFAULT**.	Yes
org.icepdf.core.**target**.dithering	string	Sets the JVM's dither rendering hint. The default value for **print** and **screen** is **VALUE_DITHER_ENABLE**. Other supported values are **VALUE_DITHER_DEFAULT** and **VALUE_DITHER_DISABLE**.	Yes
org.icepdf.core.**target**.fractionalmetrics	string	Sets the JVM's fractional metrics rendering hint. The default value for **print** and **screen** is **VALUE_FRACTIONALMETRICS_ON**. Other supported values are **VALUE_FRACTIONALMETRICS_DEFAULT** and **VALUE_FRACTIONALMETRICS_OFF**.	Yes
org.icepdf.core.**target**.interpolation	string	Sets the JVM's interpolation rendering hint. The default value for **print** and **screen** is **VALUE_INTERPOLATION_BICUBIC**. The other supported values are **VALUE_INTERPOLATION_BILINEAR** and **VALUE_INTERPOLATION_NEAREST_NEIGHBOR**.	Yes
org.icepdf.core.**target**.rendering	string	Sets the JVM's render rendering hint. The default value for **print** and **screen** is **VALUE_RENDER_QUALITY**. The other supported values are **VALUE_RENDER_DEFAULT** and **VALUE_RENDER_SPEED**	Yes
org.icepdf.core.**target**.stroke	string	Sets the JVM's stroke rendering hint. The default value for **print** and **screen** is **VALUE_STROKE_NORMALIZE**. The other supported values are	Yes

		VALUE_STROKE_PURE and **VALUE_STROKE_DEFAULT**.	
org.icepdf.core.**target**.textAntiAliasing	string	Sets the Font rendering engine's antialiasing rendering hint. The default value is **true**. The other supported value is **false**.	Yes

Property	Type	Description	Dynamic
Page Views			
org.icepdf.core.views.buffersize.vertical	string	Sets the vertical ratio that the current viewport height will be multiplied by to create a screen buffer. The default value is **1.0**. Using a larger ratio will increase the amount of memory needed by the page view.	No
org.icepdf.core.views.buffersize.horizontal	string	Sets the horizontal ratio that the current viewport width will be multiplied by to create a screen buffer. The default value is **1.0**. Using a larger ratio will increase the amount of memory needed by the page view.	No
org.icepdf.core.views.refreshfrequency	integer	Specifies the interval between refreshes of the view buffer when content is being rendered. The default value is 200 milliseconds.	No
org.icepdf.core.annotations.linkinteractive.enabled	boolean	If true, link annotation actions can be activated using the system mouse. Default value is true.	No
org.icepdf.core.views.pagepaper.color	string	Default page paper color before PDF content is painted. Default color value is #FFFFFF.	No
org.icepdf.core.views.pageborder.color	string	Default page border color. Default color value is #000000.	No

org.icepdf.core.views.pages.shadow.color	string	Default page shadow color. Default color value is #333333.	No
org.icepdf.core.views.background.color	string	Default color value is #808080.	No
org.icepdf.core.views.pages.textSelectionColor	string	Sets the color used for text selection painting. Default color value is #0077FF.	No
org.icepdf.core.views.pages.textHighlightColor	string	Sets the color used for search highlight painting. Default color value is #FFF600.	No

Common Usage Scenarios

This page last changed on Jan 28, 2010 by mark.collette.

ICEpdf is a powerful and flexible PDF rendering and viewing library which supports usage scenarios related to the parsing, inspecting, rendering, and interactive viewing of PDF documents.

This section provides detailed instructions on how to use ICEpdf to support the following use-cases:

- PDF Viewer Application - Using the ICEpdf Viewer Reference Implementation (RI), as a stand-alone application, to provide robust PDF viewing support to Java platforms.
- PDF Viewer Component - Integrating the fully-featured PDF document viewer component into your Java Swing client application or applet, to instantly provide seamless PDF document navigation and viewing capabilities in your application.
- **PDF Content Conversion** - Converting PDF Page Renderings from a PDF document to another format, such as images (.jpg, .gif, .png, etc.) or an SVG (Scalable Vector Graphics) document.
- **PDF Content Extraction** - Extracting PDF document meta-data, text, and images.
- Search API - Programmatically finding, and highlighting, specified search terms.
- Automated Annotation Creation - Create Link annotation based on text search results for batch processing of new Link Annotations across multiple documents.

Using the PDF Viewer Application

This page last changed on Jan 27, 2010 by mark.collette.

ICEpdf includes a complete standalone PDF Viewer Application reference implementation (RI) that can be used as a PDF document viewing solution on any compliant Java platform.

While the ICEpdf Viewer application provided represents a complete commercial-quality PDF document viewing solution, it is also intended to be used as a learning aid on how to use various ICEpdf features, as well as a 'head-start' for developers whose requirements closely match the existing ICEpdf Viewer's capabilities and who simply need to refine the application to meet their needs.

The Viewer application leverages the same MVC architecture, SwingViewBuilder and SwingController classes as the embeddable viewer component. Additional functionality has been implemented to provide more complete PDF Viewer functionality, similar to Adobe Acrobat Reader.

The source code for this viewer application is available in the package **`org.icepdf.ri.viewer`**.

The application uses the following classes in addition to the PDF Viewer Component to implement a standalone viewer application:

- WindowManager
- FontPropertiesManager
- PropertiesManager

These classes can be found in the `*org.icepdf.ri.viewer` package.

Using the PDF Viewer Component

This page last changed on Jun 21, 2012 by patrick.corless.

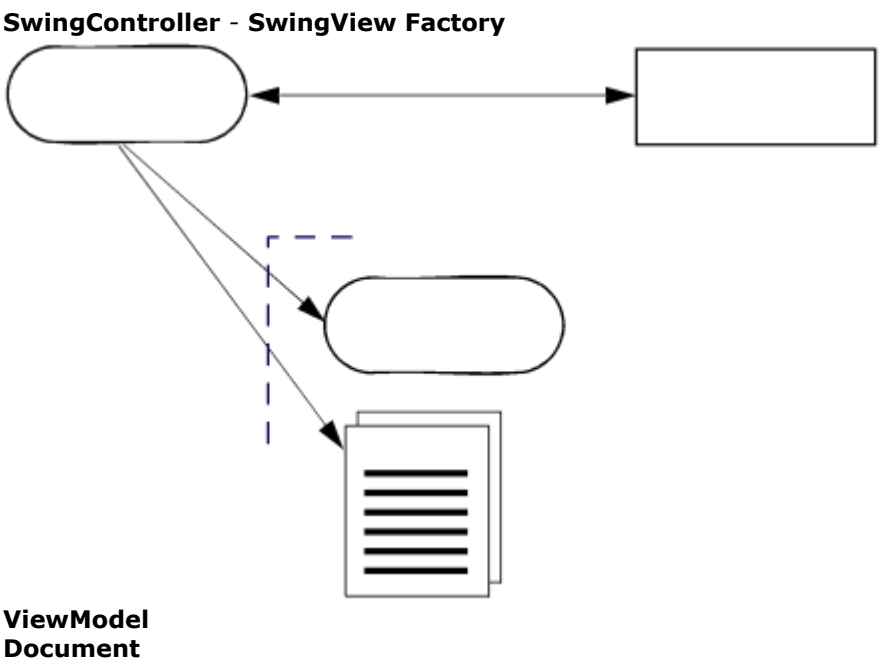
The ICEpdf library can also be used to create a full-featured PDF Viewer component which can be inserted into any Java application. For more information on the viewer component features, see Reference Implementations.

The PDF Viewer application is a reference implementation (RI) application, meaning that all source code used to implement the application is available to developers to modify as required.

The PDF Viewer RI uses the Model-View-Controller (MVC) design pattern for communication between the user, the GUI and the PDF document data. The PDF Viewer's data model is implemented by the ViewModel class. The view, which presents the user interface, is implemented using standard Java Swing components and is constructed by the SwingViewBuilder class. The controller, which interacts between the user, view and data model is represented by the SwingController class.

This relationship can be seen in Figure 1. The combination of the MVC design and the SwingViewBuilder and SwingController classes provides a very powerful and easily adaptable approach to PDF Viewer GUI development. Developers using ICEpdf can readily customize the Viewer user-interface with a very shallow learning curve and minimal coding effort.

Figure 1 - ICEpdf MVC Implementation



Creating a Viewer Component

The **`org.icepdf.core.ri.common.SwingController`** class provides convenience methods for the most common UI actions, such as rotating the document, setting the zoom level, etc. The **`org.icepdf.core.ri.common.SwingViewBuilder`** class is responsible for creating the PDF Viewer component panel populated with Swing components configured to work with the SwingController.

When using the SwingViewBuilder and SwingController classes, it is usually not necessary to use the Document object directly. The SwingController class does this for us.

The following code snippet illustrates how to build a PDF Viewer component:

```
String filePath = "somefilepath/myfile.pdf";

// build a controller
SwingController controller = new SwingController();
```

```
// Build a SwingViewFactory configured with the controller
SwingViewBuilder factory = new SwingViewBuilder(controller);

// add copy keyboard command
ComponentKeyBinding.install(controller, viewerComponentPanel);

// add interactive mouse link annotation support via callback
controller.getDocumentViewController().setAnnotationCallback(
      new org.icepdf.ri.common.MyAnnotationCallback(
            controller.getDocumentViewController()));

// Use the factory to build a JPanel that is pre-configured
//with a complete, active Viewer UI.
JPanel viewerComponentPanel = factory.buildViewerPanel();

// Create a JFrame to display the panel in
JFrame window = new JFrame("Using the Viewer Component");
window.getContentPane().add(viewerComponentPanel);
window.pack();
window.setVisible(true);

// Open a PDF document to view
controller.openDocument(filePath);
```

Note
The SwingViewBuilder class provides numerous methods that enable developers to quickly create custom viewer user interfaces (UIs) by including only those UI controls that are required, customizing existing UI controls, etc.

- Refer to **`org.icepdf.core.ri.common.SwingViewBuilder`** in the JavaDoc API documentation and Customizing the SwingViewBuilder for more information.
- See ICEpdf Viewer Application (RI) for a complete example.

Converting PDF Page Renderings

This page last changed on Jan 27, 2010 by mark.collette.

The Document class provides functionality for rendering PDF content into other formats via a Java2D graphics context. As a result, rendering PDF content to other formats is a relatively simple process with very powerful results. ICEpdf also supports Java headless mode when rending PDF content, which can be useful for server side solutions.

An example of how to extract PDF document content to SVG is available in the SVG class found in the package **`org.icepdf.ri.util`**. The following is an example of how to save page captures in PNG format.

Building a Page Capturing Class

Create a file called **`PageCapture.java`** similar to the following:

```
import org.icepdf.core.exceptions.PDFException;
import org.icepdf.core.exceptions.PDFSecurityException;
import org.icepdf.core.pobjects.Document;
import org.icepdf.core.pobjects.Page;
import org.icepdf.core.util.GraphicsRenderingHints;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.awt.image.RenderedImage;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

public class PageCapture {

  public static void main(String[] args) {

    // Get a file from the command line to open
    String filePath = args[0];

    // open the file
    Document document = new Document();
    try {
      document.setFile(filePath);
    } catch (PDFException ex) {
      System.out.println("Error parsing PDF document " + ex);
    } catch (PDFSecurityException ex) {
      System.out.println("Error encryption not supported " + ex);
    } catch (FileNotFoundException ex) {
      System.out.println("Error file not found " + ex);
    } catch (IOException ex) {
      System.out.println("Error IOException " + ex);
    }

    // save page captures to file.
    float scale = 1.0f;
    float rotation = 0f;

    // Paint each pages content to an image and
    // write the image to file
    for (int i = 0; i < document.getNumberOfPages(); i++) {
      BufferedImage image = (BufferedImage) document.getPageImage(
          i, GraphicsRenderingHints.PRINT, Page.BOUNDARY_CROPBOX, rotation, scale);
      RenderedImage rendImage = image;
      try {
        System.out.println(" capturing page " + i);
        File file = new File("imageCapture1_" + i + ".png");
        ImageIO.write(rendImage, "png", file);
      } catch (IOException e) {
        e.printStackTrace();
```

```
        }
        image.flush();
      }

      // clean up resources
      document.dispose();
    }
  }
```

The Import Statements

The **`org.icepdf.core.`*** packages are always required. The **`java.`*** packages are necessary for saving the page captures to image.

The Static Main Method

The following lines create a new Document object and open a PDF document specified by a URL.

```
Document document = new Document();
try {
  document.setFile(filePath);
} catch(PDFException ex) {
  System.out.println("Error parsing PDF document " + ex);
} catch(PDFSecurityException ex) {
  System.out.println("Error encryption not supported " + ex);
} catch(FileNotFoundException ex) {
  System.out.println("Error file not found " + ex);
} catch (IOException ex) {
  System.out.println("Error IOException " + ex);
}
```

This will take care of loading the PDF document and catch any errors that may be thrown in the process.

Before page content can be captured, it is necessary to set the zoom and rotation used to render the page's content. For this example, we are using a scale factor of 100% and will be using the default rotation of zero degrees.

```
float scale = 1.0f;
float rotation = 0f;
```

The page content can now be saved to a file.

```
for (int i = 0; i < document.getNumberOfPages(); i++) {
  BufferedImage image = (BufferedImage) document.getPageImage(
    i, GraphicsRenderingHints.PRINT, Page.BOUNDARY_CROPBOX, rotation, scale);
  RenderedImage rendImage = image;
  try {
    System.out.println(" capturing page " + i);
    File file = new File("imageCapture1_" + i + ".png");
    ImageIO.write(rendImage, "png", file);
  } catch (IOException e) {
    e.printStackTrace();
  }
  image.flush();
```

```
}
```

The code iterates through all the pages in the document and gets an image rendering of each page which is then written to a file. The last step is to free up the resources used by the Document class during the rendering process by calling the **`dispose()`** method.

```
document.dispose();
```

You now have a simple class that can save PDF page captures to disk.

Extracting PDF Document Content

This page last changed on Dec 10, 2009 by ken.fyten.

The Document class can allow alternate access to the content in a PDF document. It is possible to extract document meta-data, text, and images.

Extracting Meta-Data

ICEpdf supports extracting document meta-data via the API that is available on the document hierarchy classes in the **`org.icepdf.core.pobjects`** package. The main entry-point into the document meta-data is the Document class.

See **Content Extraction Examples** for an example that illustrates extracting meta-data from a document.

Also, see the API documentation for the **`org.icepdf.core.pobjects`** package for more information on what types of data are available.

Extracting Text

This page last changed on Jan 27, 2010 by patrick.corless.

Text extraction is possible for most PDF documents. There are, however, some limitations with how a document text is encoded and the type of font used to render the text.

> **Note**
> If a document is encrypted, the document permissions should be checked to make sure that content extraction is allowed.

The following code demonstrates how to extract text from the first page of a PDF document. The call to document.getPageText(..) returns the PageText data structure which contains child LineText, WordText and GlyphText. A call to pageText.toString() will return all the text for the current page.

```
// load the file
URL documentURL = new URL("your url");
Document document = new Document();
document.setUrl( documentURL);

try {
   // create an output file
   FileOutputStream fileOutputStream = new FileOutputStream( "extracted.txt");
   PageText pageText = document.getPageText(0);
   if (pageText != null && pageText.getPageLines() != null) {
      fileOutputStream.write(pageText.toString().getBytes());
   }
   fileOutputStream.close();
} catch (IOException e) {
   e.printStackTrace();
} finally {
   // clean up the document resources document.dispose();
}
```

Extracting Images

This page last changed on Jan 27, 2010 by mark.collette.

Image extraction is possible for all PDF documents.

> **Note**
> If a document is encrypted, the document permissions should be checked to make sure that content extraction is allowed.

The following code demonstrates how to extract images from the first page of a PDF document. The images on the first page of the document are extracted into a vector of **`Image`** objects using the Document **`getPageImages(int pgNumber)`** method. The image vector is then iterated with each image entry being saved to disk as a separate image file.

```
// load the file
URL documentURL = new URL("your url");
Document document = new Document();
document.setUrl(documentURL);

// Get the images for a single page
Enumeration tmpImages = document.getPageImages(0).elements();

// Save the images as JPEGs
int count = 0;
while (tmpImages.hasMoreElements()){
  Image image = (Image) tmpImages.nextElement();
  // create new buffered image to paint to.
  BufferedImage bufferedImage = new BufferedImage(
    image.getWidth(this), image.getHeight(this), BufferedImage.TYPE_INT_RGB);
  Graphics2D g2d = bufferedImage.createGraphics();
  g2d.drawImage(image, 0, 0, image.getWidth(this), image.getHeight(this), this);
  RenderedImage rendImage = bufferedImage;
  try {
    // Save as JPEG
    File file = new File("newimage_" + count + ".jpg");
    ImageIO.write(rendImage, "jpg", file);
  } catch (IOException e) {
    e.printStackTrace();
  }
  g2d.dispose();
  bufferedImage.flush();
}
// Clean up document resources
document.dispose();
```

Search API

This page last changed on Feb 04, 2010 by patrick.corless.

DocumentSearchController Interface Notes

The DocumentSearchController Interface defines the functionality of search API that is designed to work with a Page Object's PageText hierarchy. Each Page of a PDF document that contains text will have a none Null value when a call to page.getPageText() is made. The PageText hierarchy is made up child LineText, WordText and GlyphText. During the Page initialization process the content stream is parsed and the PageText Object and child objects are created. The resulting structure is synonymous with normal text processing; LineText represents a line of text and is made ups of child WordText objects for each word. The data structure makes is possible to properly search and extract text in a contextual manner.

The DocumentSearchControllerImpl is our reference implementation for searching for text in PDF page content streams. If a search indexing system such as Lucene is required it can easily be implemented and tied in with ICEpdf using the DocumentSearchController interface. The Search API was designed to be extensible and simple to use. The following sudo code shows how a typical search would be started and executed.

1. Get instance of SearchController form the SwingController
2. Call clearAllSearchHighlight to remove previous search terms from controller.
3. Add one or more Search Terms to the SearchController
4. Execute SearchController.search to search for the specified terms

Search controllers job is quite simple, search a page or clear a page's search results and search terms. Search terms are global to the Search controller and a set of SearchTerms will be used for subsequent page searches. When a call to one of SearchControllers clear method is called all search terms are removed. Working examples of the document searching can be found by using the Viewer RI (http://www.icepdf.org/demo/jws/icepdf.jnlp) and the Search example ([http://wiki.icefaces.org/display/PDF/Search+Example|../display/PDF/Search+Example]).

Search Terms

The SearchTerms class allows a user to specify what they want the search controller to search for. The search term has three immutable instance vars to configure case sensitivity and whole word searches.

- **Term** - string word or phrase to use as the search key.
- **Case-sensitive** - true indicates a case sensitive search of the given term, false ignores case search dependency when searching for the term.
- **Whole word** - True indicates that a search match must match the search term exactly in length. False indicates that whole word most match.

The new search API implementation allows for multiple search terms. For example, consider the following three term search:

```
// clear previous search terms and highlighting
searchController.clearAllSearchHighlight();
// use the add search term call to add multiple terms
searchController.addSearchTerm(term1, isCaseSensitive, isWholeWord);
searchController.addSearchTerm(term2, isCaseSensitive, isWholeWord);
searchController.addSearchTerm(term3, isCaseSensitive, isWholeWord);
```

Specify the page to search, for the terms. The search results are highlighted on that page.

```
// Search/highlight page y, with previously specified search terms
searchController.searchHighlightPage(y);
```

The RI search panel is setup by default to use single term searches. The search checkbox option "Cumulative" can be enabled to suppress, between search commands, the call to

```
searchController.clearAllSearchHighlight();
```

A working examples of the search API can be found in Examples.

Annotation Creation

This page last changed on Feb 04, 2010 by patrick.corless.

Annotation Creation

Link Annotations and their respective actions can be created edited and delete from a PDF document page. The Page Object has three new methods that allow developers to manipulate a PDF documents annotations:

- Page.addAnnotation(Annotation):Annotation
- Page.createAnnotation(Rectangle, AnnotationState):Annotation
- Page.deleteAnnotation(Annotation):void

Annotations are added at the document page level. The PDF specification recommends that duplicate Annotation object are not shared between pages and that a distinct instance of each should be created for every page.

Annotations and their child Action object must created using their respective factory classes. The factory class insures that the new instance of these object are properly setup so that they can be property inserted or removed from the PDF document structure and saved to file.

Annotation Factory

The Annotation factory call AnnotationFactory.buildAnnotation(..) returns a new instance of the specified annotation type which can be inserted into the Document Page Object via the addAnnotation method.

```
LinkAnnotation linkAnnotation = (LinkAnnotation)
            AnnotationFactory.buildAnnotation(
             document.getPageTree().getLibrary(),
             AnnotationFactory.LINK_ANNOTATION,
             new Rectangle(), annotationState);
```

Action factory

The Action factory call Annotation.buildAction(..) returns an new instance of the specified action type. Action must be added to Annotation object in a similar way as Annotations are to pages to insure that the Action objects are wired correctly to the document structure and their parent annotation object.

- Annotation.addAction(Action):Annotation
- Annotation.deleteAction(Action):void

The following code snippet is an example of creating a new Launch Action using the ActionFactory.

```
LaunchAction action = (LaunchAction) ActionFactory.buildAction(
        library,
        ActionFactory.LAUNCH_ACTION);
```

Annotation State

AnnotationState object is state object intended to aid in the creation of multiple page annotations that have similar look and feel as well as to save annotation edit states within the viewer RI. AnnotationState properties can be copied from one instance to another using the apply(AnnotationState):void method. The following snippet shows how to construct a new annotation state.

```
AnnotationState annotationState = new AnnotationState(Annotation.VISIBLE_RECTANGLE,
        LinkAnnotation.HIGHLIGHT_INVERT, 1f,
```

```
        BorderStyle.BORDER_STYLE_SOLID, Color.GRAY);
```

Using the Non-visual vs. the Visual API

Creating annotation in a headless environment can be done using Page and Annotation classes, no special considerations need to be taken into account. However if the annotation updates are to occur though a GUI the ICEpdf PageView objects must be made aware of any changes to the document structure.

The first step in update the UI of a new annotation or action is to get a reference to the current views page objects with the following call which returns a list of all pages that make up the document view:

```
java.util.List<AbstractPageViewComponent> pageComponents =
        controller.getDocumentViewController()
          .getDocumentViewModel().getPageComponents();
```

The list of AbstractPageViewComponent is very similar to a list of a PDF documents Page Object. The PageViewComponents are actually JComponents and contain all the interactive state logic. But adding annotation to AbstractPageViewComponent is the same as adding new annotations to Page objects.

```
PaveViewComponent.pageViewComponent.addAnnotation(linkAnnotation);
PaveViewComponent.pageViewComponent.removeAnnotation(linkAnnotation);
```

The distinction between visual and none visual annotation creation is very important. If you use the non-visual method in a visual environment you will not see the annotation until the document is saved and opened. You can however use the visual method to create annotation in a non-visual environment but you will be loading a lot of extra classes that are not needed.

The source-code for the example is located at install_dir/icepdf/examples/annotation/NewAnnotationPostPageLoad.java.

The source-code for the example is located at install_dir/icepdf/examples/annotation/NewAnnotationPrePageLoad.java.

Saving Document Changes

Annotation and action document changes are stored in a StateManager class. In most cases developers will not have to modify the StateManager class directly, it was designed to be used by the Page, PageComponent and Annotation object directly. Saving document change is quite simple:

```
document.saveToOutputStream(OutputStream out):long
```

When a file is saved all changed are appended to the PDF file as an incremental update.

Viewer RI

This page last changed on Jan 27, 2010 by patrick.corless.

The ICEpdf Viewer is a reference implementation (RI) of a standalone PDF viewer application. You can use it as is, or as a starting point for your own custom application.

The application uses the SwingViewBuilder object to create the GUI elements in the viewer application, such as the toolbar and menu system. These GUI elements, including the page view, are controlled by the SwingController object which produces a rich viewer application that can be used as is in most implementations.

The source-code for the ICEpdf Viewer Application is located in the **`[install_dir]/icepdf/viewer/`** directory.

Starting the Viewer

This page last changed on Jan 27, 2010 by mark.collette.

Starting the ICEpdf Viewer Application

1. Ensure that JDK 1.5.0 or higher has been installed.

2. Add the relevant JAR files to the classpath. You need at least the following:

- **icepdf-core.jar**
- **icepdf-viewer.jar**

To enable ICEpdf Pro and full PDF font support, install the following:

- **icepdf-pro.jar**
- **icepdf-pro-intl.jar**

To enable exporting to SVG, you also require the following Batik JAR files:

- **batik-awt-util.jar**
- **batik-dom.jar**
- **batik-svg-dom.jar**
- **batik-svggen.jar**
- **batik-util.jar**
- **batik-xml.jar**

For more information, see **Batik Library for SVG Support**.

3. Run "**java org.icepdf.core.ri.viewer.Main** [option <value>]"

back to top

Starting as an Executable JAR

The **icepdf-viewer.jar** file is an executable JAR file, so you can also start the Viewer Application with JDK 1.5.0 or greater as follows:

```
java -jar icepdf-viewer.jar
```

back to top

Command Line Options

Option	Description
-loadfile *filename*	Starts the ICEpdf Viewer and displays the specified local PDF file. Use the following syntax: **-loadfile c:/examplepath/file.pdf**
-loadurl *url*	Starts the ICEpdf Viewer and displays the PDF file at the specified URL. Use the following syntax: **-loadurl http://www.examplesite.com/file.pdf**

To start the ICEpdf Viewer without command line options or Batik support:
"**java -classpath icepdf-core.jar;icepdf-viewer.jar org.icepdf.ri.viewer.Main**"

back to top

Settings Directory

The ICEpdf Viewer stores its settings in the directory:

<user_dir>/.icesoft/icepdf_viewer

...where **<user_dir>** is the platform-specific directory specified by the **user.home** system property.

back to top

Using the Viewer

This page last changed on Jan 27, 2010 by mark.collette.

Using the ICEpdf Viewer

This section describes how to use the ICEpdf Viewer application to do the following:

- Open PDF files
- Understand the ICEpdf Viewer work area
- Navigate and manipulate the view of a PDF document
- Export, save, and print the contents of a PDF document
- Exit a document and the application

Opening PDF Files

You can open a PDF file from the local file system or from a URL.

To open a local PDF file

1. Launch your ICEpdf application (if it is not already running).
2. Click **File > Open > File** (in the menu).
3. In the **Open** dialog box, select a PDF file and click **Open**.

To open a PDF from a URL

1. Launch ICEpdf.
2. Select **File > Open > URL** (in the menu).
3. In the **Open URL** dialog box, type the URL of a PDF file in the form `http://www.icepdf.org/resources/DevelopersGuide.pdf` and click **OK**.

To open a PDF using Drag and Drop

You can open a PDF file by dragging it from your file system onto the Viewer application window. You can also drag and drop multiple files at once.

Furthermore, some applications allow you to drag and drop URL text strings. If you drag a URL pointing to a PDF file onto the Viewer application, that file is opened. The text string must contain an `http://` prefix and a ***.pdf** reference.

> **Note**
> If a file is password protected and security support has been configured, a password dialog is displayed. You have three attempts to enter the correct password, after which an error message is displayed. If the file has other security settings, such as "No Printing", the ICEpdf Viewer respects those settings.

Opening Multiple PDF Files

The ICEpdf Viewer implements a Single Document Interface (SDI) for viewing PDF files. Multiple PDF files can be viewed at the same time, each in its own application window. Some MDI functionality has been included in the Window menu, which allows the user to switch between each open Viewer window, minimize all Viewer windows, and bring all Viewer windows to the front.

An additional ICEpdf Viewer window is opened automatically when you open another PDF file.

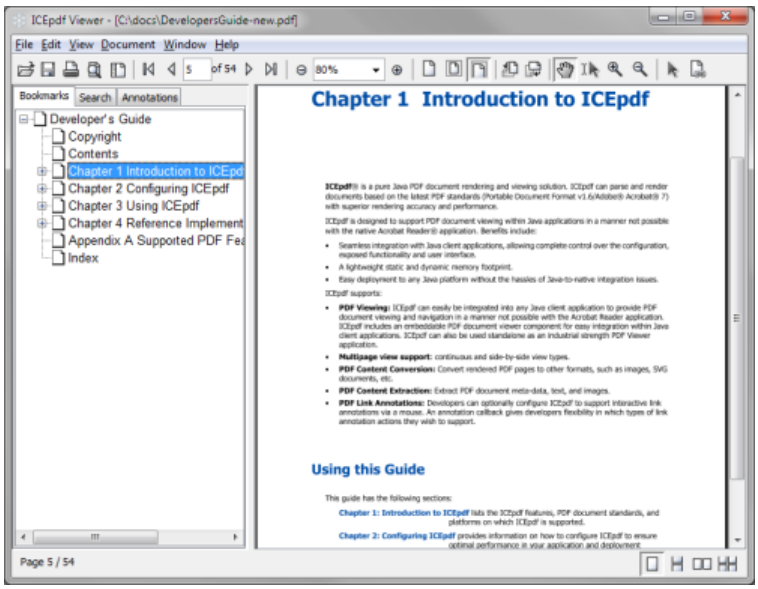
back to top

The ICEpdf Viewer Work Area

The ICEpdf Viewer consists of the following four distinct work areas:

1. **Main toolbar** - provides controls and buttons to navigate and work with a PDF file.
2. **Document pane** - displays a PDF file.
3. **Utility pane** - has a bookmark tab which lets you browse a PDF file using a documents bookmarks and a search tab which enables you to search for text in a file.
4. **View toolbar** - allows you to change the way a document's pages are displayed in the document pane.

Figure 2 ICEpdf Viewer Work Area



Main Toolbar

The toolbar has shortcuts for many common functions, but you can hide it to have a larger display area. To show or hide the toolbar, select **View > Show Toolbar** or **View > Hide Toolbar**. The toolbar provides access to the view and navigation functionality.

Figure 3 ICEpdf Main Toolbar

From left to right, the toolbar contains:

• Open Document • Save As... • Print Document • Search Document • Show/ Hide Utility Pane	• First Page • Previous Page • Next Page • Last Page	• Zoom Out • Zoom In	• Actual Size • Fit in Window • Fit Width	• Rotate Left • Rotate Right	• Pan Tool • Text Selection Tool • Zoom In Tool • Zoom Out Tool	• Annotation Select Tool • Link Annotation Tool

View Toolbar

The view toolbar contains four buttons that allow you to change how a PDF document is displayed.

From left to right:

* Facing Page View Continuous
* Facing Page View Non-continuous
* Single Page View Continuous
* Single Page View Non-continuous

About the Utility/Bookmark Pane

You can show or hide the utility pane, which contains tabs for a document's bookmarks and for the search function. If the file does not have any bookmarks, the bookmarks tab is not displayed.
To show or hide the utility pane, select **View > Show Utility Pane** or **View > Hide Utility Pane**. Alternatively, if the toolbar is displayed, you can click the **Show/Hide Utility Pane** button.

back to top

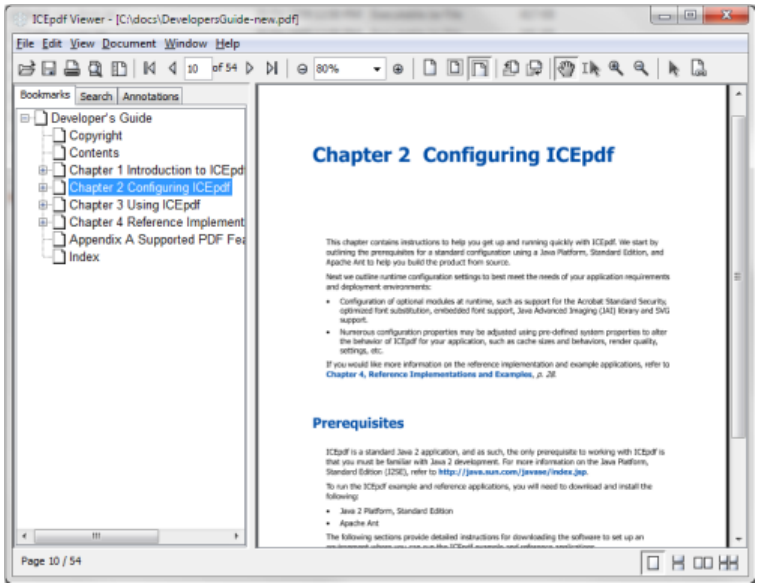
Navigating Within a PDF File

Using Bookmarks

If a PDF file has been created with bookmarks, you can use the bookmarks like a table of contents to choose what you want to view.
Bookmarks are displayed on a tab in the utility pane on the left side of the Viewer. However, they are not displayed if **View > Hide Utility Pane** has been selected.

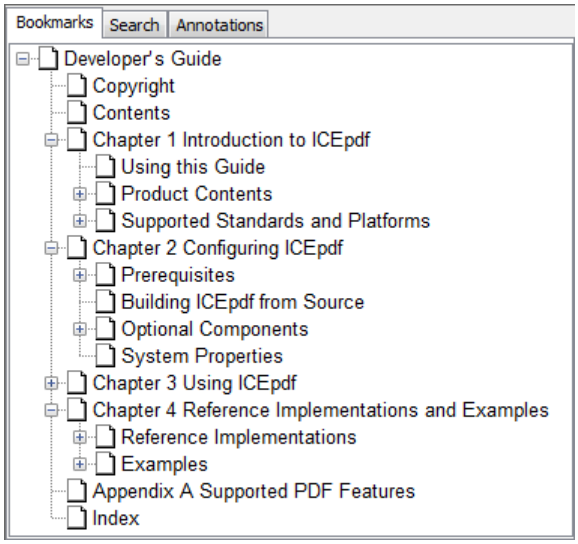
Viewing a Bookmark Topic

To view a bookmark topic, click the bookmark topic name:



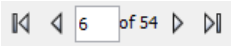
Viewing More Bookmark Topics

If a bookmark has a **+** symbol beside it, you can expand a parent bookmark topic to reveal its children. Similarly, click the - symbol next to a bookmark to collapse a parent bookmark and hide its children.



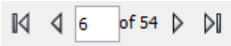
Viewing the Next or Previous Page

* To view the next page of a PDF file, select **Document > Next Page**.
* To view the previous page of a PDF file, select **Document > Previous Page**. Using the toolbar, select the Next Page or Previous Page buttons to navigate.



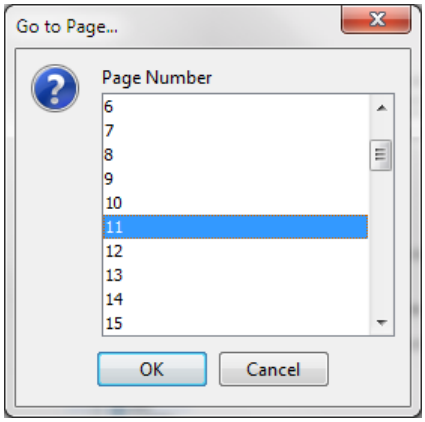
Viewing the First or Last Page

* To view the first page of a PDF file, select **Document > First Page**.
* To view the previous page of a PDF file, select **Document > Last Page**. Using the toolbar, select the First Page or Last Page buttons to navigate.

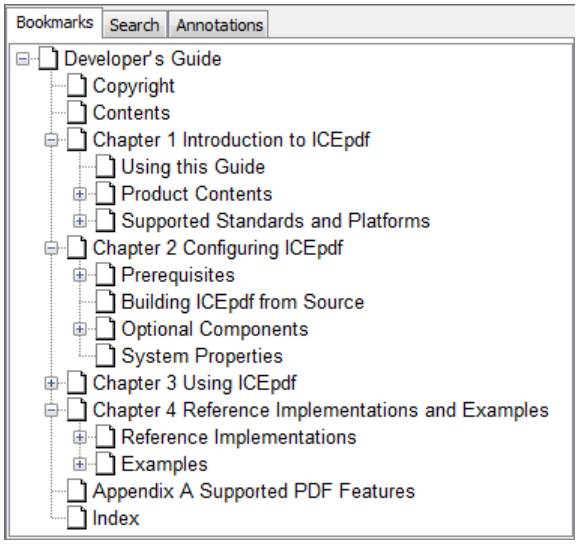


Viewing a Specific Page

To view a specific page of a PDF file:
1. Select **Document > Go To Page**.



2. In the **Go To Page** dialog, select a page number and click **OK**.
Using the toolbar, type a page number in the page number field and press **Enter**.



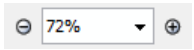
top

back to top

Document Manipulation

Zooming In and Out

You can zoom in on a page to get a closer view, and you can zoom out for a smaller view:
* To zoom in on a page, select **View > Zoom In**. You can zoom in repeatedly to get a closer view.
* To zoom out from a page, select **View > Zoom Out**. You can zoom out repeatedly to get a smaller view.
Using the toolbar, select the Zoom In or Zoom Out buttons. You can select a magnification percentage from the Zoom drop-down selection box, or type a value in the Zoom box and press **Enter**.



Select the Zoom In Tool or Zoom Out Tool to simply click to zoom in or out on an area of the PDF file

Viewing a PDF File in its Original Size or to Fit the Window

* To view the PDF file in its actual size, select **View > Actual Size**. The actual size is the size of the page as it was originally created.
* To view the PDF file so that the entire page is visible, select **View > Fit in Window**.
* To view the PDF file so that the width of the page fills the window size, select **View > Fit Width**.
Using the toolbar, click the Actual Size, Fit in Window, and Fit Width buttons.

Rotating a PDF File

If the PDF file is displayed sideways, you can rotate it so that it is easier to view or read.
To rotate the PDF file, select **View > Rotate Left** or **View > Rotate Right**. The view is rotated 90 degrees clockwise or counter-clockwise each time you select it. Using the toolbar, click the Rotate Left and Rotate Right buttons.

back to top

Keyboard and Mouse Manipulation

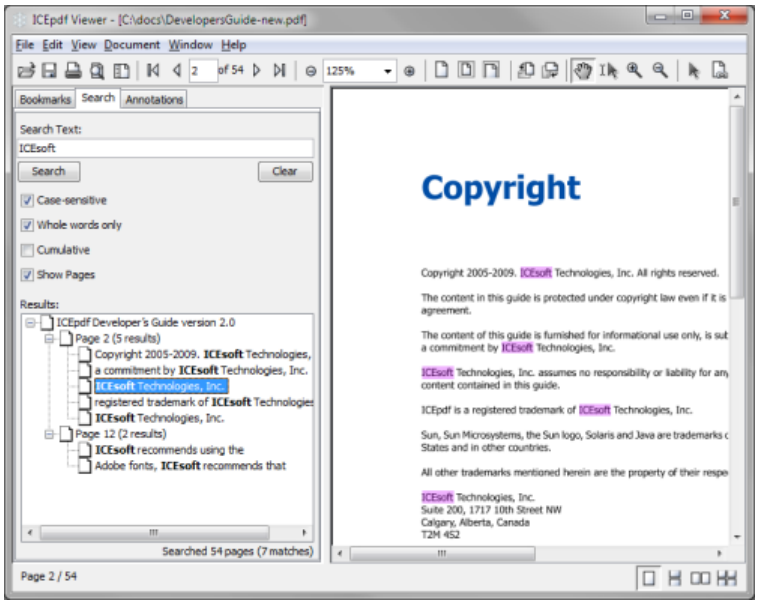
While viewing a PDF, you can use the following keyboard and mouse actions:

Keyboard or Mouse Action	Result
Page Up / Page Down	Scrolls the current page up/down. If the top/ bottom of the page is reached, the view changes to the previous/next page in the document.
Arrow keys	Scrolls the display in the direction of the arrow key pressed. If the top/bottom of the page is reached, the view changes to the previous/next page in the document.
Mouse wheel	Scrolls the display up/down. If the top/bottom of the page is reached, the view changes to the previous/next page in the document.
Mouse click	If the Zoom In tool is selected, zooms in. If the Zoom Out tool is selected, zooms out.
Mouse click and drag	If the Pan Tool is selected (the default), pans the display in the direction the mouse is dragged.

back to top

Searching for Text

If a PDF file contains text, you can search for text strings within it using the Search function.
The Search function is displayed on a tab in the utility pane on the left side of the Viewer. However, it is not displayed if the Utility Pane is hidden. In this case, select **Document > Search** or click the Search Document button to open the Search tab.

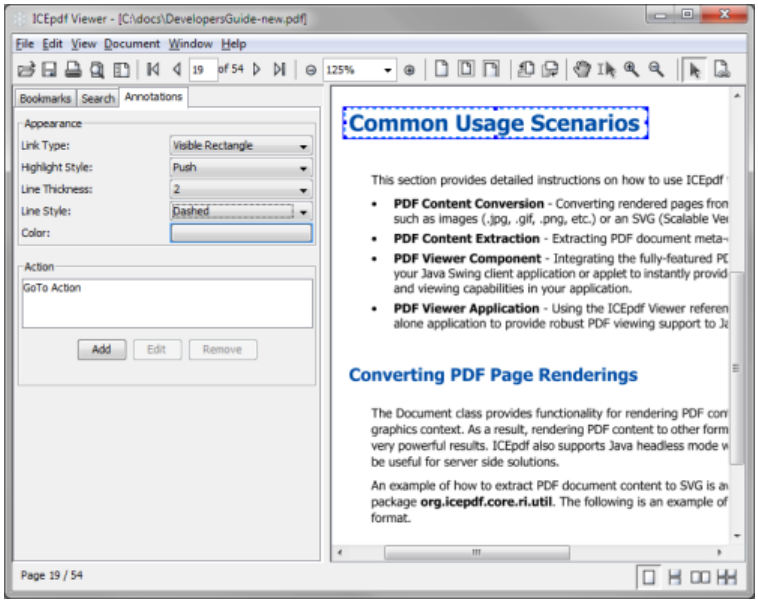


To search for a text string, type it in the **Search Text** field and click **Search**. The results are displayed in the Results field, indicating how many hits were found on each page of the document. You can then click on an entry in the Result field to go directly to that page.

back to top

Working with Annotations

The Viewer RI is able to display and add / edit / remove Link annotations from a PDF document.
To view, edit, or delete an existing annotation, first use the Select Tool to select an existing annotation.
Once an annotation is selected:
* It's properties will be displayed in the Annotations pane of the Utility Panel. These can be edited directly in the Annotations pane.
* It can be moved via dragging it to a new location on the page with the mouse.
* It can be deleted/removed by pressing the **Delete** key, or the **Ctrl-D** / **Cmd-D** key (OS X).

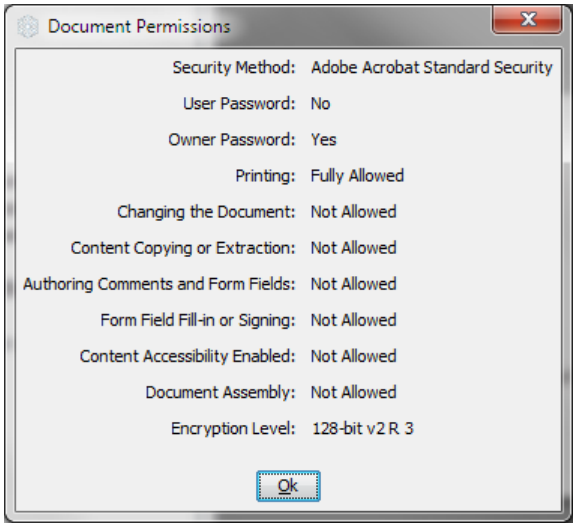


To create a new annotation, use the Link Annotation Tool. With the tool selected, the mouse can be used to select a rectangular region on the page that will form the extent of the new link annotation. Once the region is defined, the Annotations pane in the Utility Panel can be used to define its attributes.

back to top

Displaying Document Permissions and Information

To display security information about an open PDF document, such as the security method and which security features are enabled, select **File > Document Permissions**.



To display other information about an open PDF document, such as the creator and modification dates, select **File > Document Information**.
top

back to top

Exporting a PDF File to SVG

SVG support is available if you have configured the optional Batik SVG library. For more information, see *Batik Library for SVG Support*.

To export a PDF file as an SVG file

1. If Batik is configured, select **File > Export SVG**.
2. In the Export as SVG dialog, specify a location and filename and click **OK**.

back to top

Exporting a PDF File to Text

You can export the text in a PDF to a text file.

To export a PDF file as a text file:

1. Select **File > Export Text**.
2. In the Export Document Text dialog, specify a location and filename and click **OK**.

back to top

Printing a PDF File

To print a PDF file with a printer dialog, select **File** > **Print.**

Alternatively, from the toolbar, click the  (**Print** button) to print a PDF file without displaying a print dialog.

To set the paper, orientation and margin settings before printing the file, select **File > Print Setup**.

back to top

Saving a PDF File

To save a copy and any annotation related changes of the PDF file you are viewing:

1. Select **File > Save As**.
2. In the **Save As** dialog box, navigate to a location for the file, type a unique filename and click **OK**.

Alternatively, from the toolbar, click the  (**Save** button) to save a PDF files changes.

back to top

Closing a PDF File

To close the PDF file you are viewing, leaving the application open, select **File > Close**.

back to top

Exiting the ICEpdf Viewer Application

To exit from the ICEpdf Viewer and close any open files, select **File > Exit**.

back to top

Examples

This page last changed on Jan 27, 2010 by patrick.corless.

ICEpdf API Examples

ICEpdf includes a comprehensive set of examples and reference implementations in source code form to enable rapid learning and successful use of the product.

The examples are simplified applications that demonstrate how to use a specific feature or capability.

- Annotation Example
- Applet Example
- Export to SVG
- Page Capture Example
- Multi-page Tiff Capture
- ICEfaces Example
- Viewer Component Example
- Content Extraction Examples
- Print Services Example
- Search Example

Annotation Example

This page last changed on Jan 27, 2010 by mark.collette.

About the Annotation Examples

The annotation examples demonstrate how to use the ICEpdf API to interact with document annotations and how to create new annotations programmatically.

There are three individual extraction examples:

- **`MyAnnotationCallback.java`**
  - Shows how the AnnotationCallback interface can be used to interpret how annotation actions can be handled, when activated in the user interface.
  - The example also shows how annotation border painting can be manipulated, to show users which annotations they have already clicked on.
- **`NewAnnotationPostPageLoad.java`**
  - Uses the Search API to find and create annotations with the given search criteria. The annotation creation occurs after the document is viewable.
- **`NewAnnotationPrePageLoad.java`**
  - Uses the Search API to find and create annotations with the given search criteria. The annotation creation occurs before the document pages become viewable.

The source-code for the example is located in the **[install_dir]/icepdf/examples/annotation/** directory.

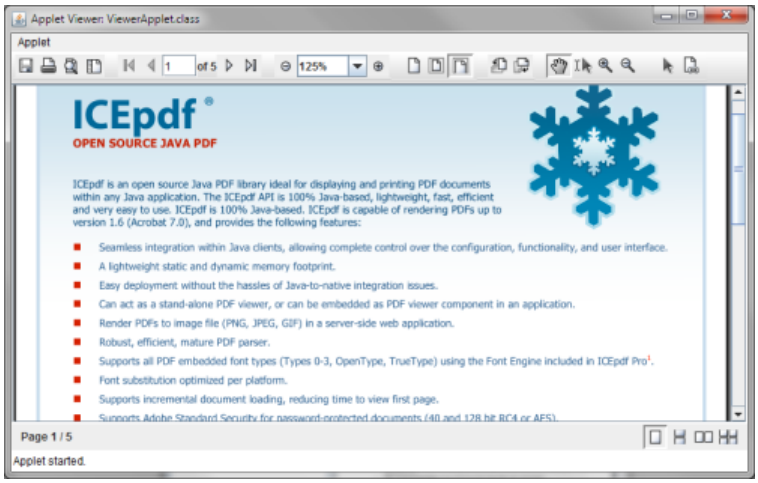
Applet Example

This page last changed on Jan 27, 2010 by mark.collette.

About the Applet Example

The Applet example demonstrates deployment of the ICEpdf Viewer as a Java Applet. This example is very similar to the Viewer Component Example but differs by extending the JApplet class.

The Applet example can be built with the provided ant build script. The default ant target is icepdf.applet.jar. Once the build runs successfully all needed jars and the respective HTML files are copied to the './dist' folder.



Signing the Jars
The build script does not attempt to sign the jar files, however most Applet deployment scenarios require jar signatures. More information on how to sign jars, with your own certificate, can be found here: http://java.sun.com/docs/books/tutorial/deployment/jar/signing.html

The example PDFApplet.html specifies the PDF file to open the param 'URL' and has a default value of 'http://www.icepdf.org/pdf/ICEpdf-Datasheet.pdf'. Make sure you have Internet access before attempting to run the demo, to avoid a document loading error.

The source-code for the example is located in the **[install_dir]/icepdf/examples/applet/** directory.

Export to SVG

This page last changed on Jan 21, 2013 by patrick.corless.

Export to SVG Example

The export SVG code snippet below shows ho easy it is to convert a PDF document to the SVG file format.

```
try {

    if (pdfDocument != null &&
            (pageNumber >= 0 && pageNumber < pdfDocument.getNumberOfPages())) {
        // Get a DOMImplementation
        DOMImplementation domImpl = GenericDOMImplementation.getDOMImplementation();
        // Create an instance of org.w3c.dom.Document
        org.w3c.dom.Document document = domImpl.createDocument(null, "svg",
                null);
        // Create an instance of the SVG Generator
        SVGGraphics2D svgGenerator = new SVGGraphics2D(document);
        // Ask the test to render into the SVG Graphics2D implementation

        float userRotation = 0;
        float userZoom = 1;
        PDimension pdfDimension = pdfDocument.getPageDimension(pageNumber, userRotation, userZoom);
        svgGenerator.setSVGCanvasSize(pdfDimension.toDimension());

        // paint the page to the Batik svgGenerator graphics context.
        pdfDocument.paintPage(pageNumber, svgGenerator,
                GraphicsRenderingHints.PRINT,
                Page.BOUNDARY_CROPBOX,
                userRotation, userZoom);

        // Finally, stream out SVG to the standard output using UTF-8
        // character to byte encoding
        boolean useCSS = true;          // we want to use CSS style attribute
        //          File f=new File("a.svg");
        //          Writer out = new OutputStreamWriter(new FileOutputStream(f), "UTF-8");
        svgGenerator.stream(out, useCSS);
    }
} catch (org.apache.batik.svggen.SVGGraphics2DIOException e) {
    logger.log(Level.SEVERE, "Error creating svg document.", e);
}
```

Tooltip
Make sure to always call document.dispose() when you are done processing a PDF document. Dispose cleans resources used by the document object.

Page Capture Example

This page last changed on Jan 27, 2010 by mark.collette.

About the Page Capture Examle

The Page Capture example demonstrates how to use the ICEpdf Document class to capture PDF page renders as image files. Once a file has been opened, the document pages can be iterated over, to generate an image capture of each page.

```
// save page caputres to file.
float scale = 1.0f;
float rotation = 0f;

// Paint each pages content to an image and write the image to file
for (int i = 0; i < document.getNumberOfPages(); i++) {
    BufferedImage image = (BufferedImage)
            document.getPageImage(i,
                    GraphicsRenderingHints.SCREEN,
                    Page.BOUNDARY_CROPBOX, rotation, scale);
    RenderedImage rendImage = image;
    // capture the page image to file
    try {
        System.out.println("\t capturing page " + i);
        File file = new File("imageCapture1_" + i + ".png");
        ImageIO.write(rendImage, "png", file);

    } catch (IOException e) {
        e.printStackTrace();
    }
    image.flush();
}
```

Tooltip
Make sure to always call document.dispose() when you are done processing a PDF document. Dispose cleans resources used by the document object.

The full source-code for the example is located in the **[install_dir]/icepdf/examples/capture/** directory.

Multi-page Tiff Capture

This page last changed on Jan 27, 2010 by mark.collette.

About the Multi-page Tiff Capture Example

The multi-page TIFF capture example captures all document pages and adds them to a multi-page group-4 fax TIFF graphics file. This differs from the Page Capture Example by saving in black and white, instead of in RGB colour, as well as saving every page of the PDF in a single TIFF graphics file, instead of a different graphics file for each page.

> **Library Dependency**
> This example requires that the Java Advanced Imaging Image I/O Tools jar is on the classpath, in addition to the typical jars one may use with ICEpdf, even including the regular Java Advanced Imaging jars.

The source code for the example is located in the **[install_dir]/icepdf/examples/captureMultiple/** directory.

ICEfaces Example

This page last changed on Feb 04, 2010 by patrick.corless.

About the ICEfaces Example

ICEfaces PDF Viewer Application utilizes the ICEfaces framework and ICEpdf core to render PDF documents in a Rich Web application. ICEfaces 1.8 or greater is needed to compile this example. The PDF rendering is controlled by the Servlet ***org.icepdf.examples.jsf.viewer.servlet.PdfRenderer*** and all of the document controls and UI features are implemented in JSF using the ICEfaces framework.

Update the Apache Ant build.properties variable *common.build.file* to point to the location of build-common.xml located in the **[install_dir]/icefaces/samples/etc/** directory. The common ICEfaces build script is very flexible having build targets for most major Servlet containers. For example ">ant tomcat6.0" will build a Tomcat 6.0 compatible ware in the **[install_dir]/icepdf/examples/icefaces/dist/** directory.

The source code for the example is located in the **[install_dir]/icepdf/examples/icefaces/** directory.

ICEfaces is available at **http://www.icefaces.org/downloads/**.

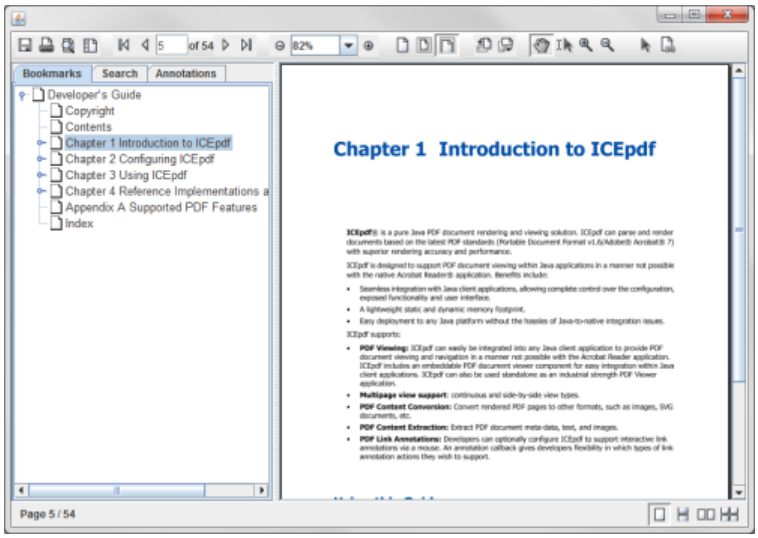
Viewer Component Example

This page last changed on Jan 27, 2010 by mark.collette.

About the Viewer Component Example

The Viewer Component example demonstrates how to use the SwingController and SwingViewBuilder classes to create a PDF viewer component which can be used in a larger swing application. The viewer component is similar to the Viewer RI but lacks a parent JFrame and menu bar.

The source-code for the example is located in the **[install_dir]/icepdf/examples/component/** directory.



Running the Viewer Component Example

1. Ensure that JDK 1.5.0 or higher has been installed.

2. Add the relevant JAR files to the classpath. You need at least the following:

   - icepdf-core.jar
   - icepdf-viewer.jar

To enable ICEpdf Pro and full PDF font support, install the following:

   - icepdf-pro.jar
   - icepdf-pro-intl.jar

3. Run "java ViewerComponentExample"

Content Extraction Examples

This page last changed on Jan 27, 2010 by mark.collette.

About the Content Extraction Examples

The Content Extraction examples demonstrate how to use the ICEpdf Document class to extract meta-data, text, and images from within a PDF document.

There are three individual extraction examples:

- **`PageImageExtraction.java`** - Extracts the images from the first page of the loaded PDF document.
- **`PageMetaDataExtraction.java`** - Extracts document meta-data such as title, author, creator, creation date, etc.
- **`PageTextExtraction.java`** - Extracts the text from the first page of the loaded PDF document if any exists.

The source-code for the example is located in the **[install_dir]/icepdf/examples/extraction/** directory.

Print Services Example

This page last changed on Jan 28, 2010 by patrick.corless.

The Print Services example demonstrates how the PrintHelper class can be used to print PDF documents using Java Print Services. The example also shows how it is possible to select and modify available printers and their settings.

This example also shows how to change the default paper size which is handy for users living outside of North America. Here is a code snippet of how most print jobs are setup.

```
Document pdf = new Document();
pdf.setFile(filePath);
SwingController sc = new SwingController();
DocumentViewController vc = new DocumentViewControllerImpl(sc);
vc.setDocument(pdf);
// create a new print helper with a specified paper size and print
// quality
PrintHelper printHelper = new PrintHelper(vc, pdf.getPageTree(),
        MediaSizeName.NA_LEGAL, PrintQuality.DRAFT);
// try and print pages 1 - 10, 1 copy, scale to fit paper.
printHelper.setupPrintService(selectedService, 0, 9, 1, true);
// print the document
printHelper.print();
```

The source-code for this example is located in the **[install_dir]/icepdf/examples/printServices/** directory.

Search Example

This page last changed on Jan 27, 2010 by patrick.corless.

About the Search Highlight Example

The Search example demonstrates how the Search Controller API can be used to search for text in a PDF document. Text that has been found is drawn in a highlighted state to easily identify the results. Search criteria can also be set for case-sensitive and whole word searches.

The following code snippet shows how to add search terms tot he searchController and then iterate over any hits.

```
DocumentSearchController searchController =
        controller.getDocumentSearchController();
// add a specified search terms.
for (String term : terms) {
    searchController.addSearchTerm(term, false, false);
}
// search the pages in the document or a subset
Document document = controller.getDocument();
// list of founds words to print out
ArrayList<WordText> foundWords;
for (int pageIndex = 0; pageIndex < document.getNumberOfPages();
     pageIndex++) {
    foundWords = searchController.searchPage(pageIndex);
    System.out.println("Page " + pageIndex);
    if (foundWords != null){
        for (WordText wordText : foundWords){
            System.out.println("    found hit: " + wordText.toString());
        }
    }
}
```

The source-code for this example is located in the **[install_dir]/icepdf/examples/search/** directory.

Advanced Topics

This page last changed on Jan 27, 2010 by patrick.corless.

This section provides examples for numerous ICEpdf advanced techniques.

Customizing the SwingViewBuilder

This page last changed on Jan 27, 2010 by mark.collette.

The **`org.icepdf.ri.common.SwingViewBuilder`** class constructs a set of GUI components which are pre-configured to work with the **`org.icepdf.ri.common.SwingController`** class to manipulate and reflect the status of view of a rendered PDF document.

The `SwingViewBuilder` methods, `buildViewerFrame` and `buildViewerPanel`, provide the high-level entry points for the GUI construction process, with each constructing complete PDF Viewer user interfaces. In addition, numerous other `build...` methods are available that construct specific components and subcomponents.

See the JavaDoc for the SwingViewBuilder class for an overview of the various build... methods.

For example, the `buildCompleteMenubar()` method will construct and return a complete application menu bar, including File, View, Document, Window, and Help menus. The `buildFileMenu()` method will construct and return the File menu, including menu items for Open, Close, Save As, Export, Print, etc. You may use the `SwingViewBuilder` methods at any level that is appropriate for your application to construct the various user interface components that you require.

If you prefer to customize the GUI components that the `SwingViewBuilder` creates, you will need to adopt one of the following strategies:

1. Directly modify the `SwingViewBuilder` source code, or copy the `SwingViewBuilder` class into a new class of your own and make required changes to the new class. Using this approach will ensure that any changes or enhancements made to the default Viewer user interface in future versions of ICEpdf will not impact your application. You will need to integrate any changes you wish to adopt manually.
2. Subclass the SwingViewBuilder class and override the build... methods that you need to modify. The advantage of this approach is that your application will automatically incorporate any changes or additions to the Viewer user interface provided in subsequent releases of ICEpdf for those methods that you haven't overridden. For example, if you are satisfied with the default Viewer Component user interface, but need to modify the Help->About menu-item to display a custom About dialog for your application, you would extend the SwingViewBuilder class and override the buildAboutMenuItem() method to return a menu item that displays your custom About dialog. If a future version of ICEpdf includes additional Tools, such as a Text Selection Tool, your user interface will automatically adopt the new toolbar and menu changes to add support for the new Tool without requiring any manual integration of the new code.

Window Management

This page last changed on Dec 10, 2009 by ken.fyten.

Window events, such as closing a window, must be handled by your application, not the ICEpdf framework. Your application needs to track windows and viewports from creation to destruction.

The package **`org.icepdf.core.ri.viewer`** contains reference code for creating window management.

Adding a Custom Utility Tool

This page last changed on Jan 27, 2010 by mark.collette.

The ICEpdf Viewer RI provides a tabbed utility pane component that is used to display a document outline (bookmarks) when available and a document search capability. This utility pane can also be configured to support additional custom utility tab components as required. The SwingViewBuilder method **`JTabbedPane buildUtilityTabbedPane()`** is responsible for creating the tabbed utility pane for the outline and search user interfaces. This method can be easily modified to add your own custom utility user interface.

The **`org.icepdf.core.ri.common.SearchPanel`** component provides a reference implementation that demonstrates implementing a basic search interface for finding text within a PDF document.

Building a Custom Page View

This page last changed on Jan 27, 2010 by mark.collette.

By default, the ICEpdf Viewer application can display PDF documents in any of the predefined views specified in the PDF Reference. The viewer application will first see if the document specifies which document view to use. If not specified, then the viewer selects the view that was last used.

The source code for all of the view layout types can be found in the package **`org.icepdf.core.ri.common.views`**. Each page is represented by a PageViewComponentImpl object which can be easily used to build
custom view types.

The page view implementation uses a secondary MVC to manage the multiple views efficiently. All supporting classes can be found in the package **`org.icepdf.core.views`**. The support view types are as follows:

- Single Page - Displays one page at a time.
- One Column - Displays the pages, continuously, in one column.
- Two Column Left - Displays the pages, continuously, in two columns, with odd-numbered pages on the left.
- Two Page Left - Displays the pages, two at a time, with odd-numbered pages on the left.

Implementing a SecurityCallback

This page last changed on Jan 27, 2010 by mark.collette.

When the Document class encounters a PDF document that is encrypted with Acrobat standard security, it first tries to open the PDF file with an empty password string. If the Document class fails to validate the empty password, the application must have a mechanism to request the password. You can use the **`org.icepdf.core.SecurityCallback`** interface to do this.

The interface has one method, which is called by the Document class to retrieve a document's password. You can implement the SecurityCallback interface in numerous ways to meet the needs of your application. For example, the package **`org.icepdf.core.ri.common`** contains reference code for the SecurityCallback in the class **`MyGUISecurityCallback`**.

Consider the following code which would allow a user to type in the documents password if needed.

```
// new document instance
Document document = new Document();
// setup a security callback before opening an encrypted document.
document.setSecurityCallback(new SecurityCallback(){
   public String requestPassword(Document document) {
      System.out.println(
         "This document is Encrypted please type the document password:");
      String input = "";
      // get users  password
      try {
         BufferedReader stdin =
            new BufferedReader(new InputStreamReader(System.in));
         input = stdin.readLine();
      } catch (IOException e) {}
      return input;
   }
});
// finally open the document.
document.setFile(filePath);
```

The anonymous inner class allows a developer to handle how they want to ask a user for the password. In this case the command line is used, but alternatively a dialog or some other input form could have been use.

Implementing an AnnotationCallback

This page last changed on Jan 27, 2010 by mark.collette.

ICEpdf can optionally be configured to support interacting with link annotations. The Viewer RI and Pilot RI, by default, come configured with their own implementations of the **`org.icepdf.core.AnnotationCallback`**. They differ only by how they handle external URI Actions. The Viewer RI will load an external URI with the Operating System's default web browser, while the Pilot RI will load external URI links in a new ICEbrowser viewport.

ICEpdf only supports annotations via the mouse pointer; there is no keyboard support at this time. When the mouse is moved over a portion of a PDF document which is marked as an annotation, the mouse cursor will change into a hand with a pointing finger. When a user clicks on the annotation, ICEpdf will draw any effect specified by the selected annotation, but it will not execute the annotation action; it instead passes the selected annotation to the AnnotationCallback. It is up to the AnnotationCallback implementation to process the annotation's actions.

A default implementation of an AnnotationCallback can be found in **`org.icepdf.core.ri.common.MyAnnotationCallback`**.

The following code snippet shows how to configure the SwingController with an AnnotationCallback that would allow the user to interact with annotations in a PDF document.

```
// build a component controller
SwingController controller = new SwingController();

SwingViewBuilder factory = new SwingViewBuilder(controller);

JPanel viewerComponentPanel = factory.buildViewerPanel();

// add interactive mouse link annotation support via callback
controller.getDocumentViewController().setAnnotationCallback(
        new org.icepdf.ri.common.MyAnnotationCallback(
                controller.getDocumentViewController()));

JFrame applicationFrame = new JFrame();
applicationFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
applicationFrame.getContentPane().add(viewerComponentPanel);

// Now that the GUI is all in place, we can try openning a PDF
controller.openDocument(filePath);
```

Printing

This page last changed on Jan 28, 2010 by patrick.corless.

Printing a PDF document with ICEpdf is a highly configurable task that allows users to print using a wide range of Java technologies. To aid developers in printing, the package **`org.icepdf.core.ri.common`** contains the PrintHelper class that implements Java 2 Printable and Pageable interfaces. The source code for this class is available for users who want to gain a greater understanding of the printing process or modify the printing behavior.

A simple standalone print example can be found here Print Services Example

Font Management

This page last changed on Dec 10, 2009 by ken.fyten.

ICEpdf uses a font manager to manage fonts that exist on the host operating system. The FontManager class can be found in the package **`org.icepdf.core.fonts`**.

When the font manager's **`readSystemFonts()`** method is called, it tries to read all fonts on the host operating system and stores the name, family and path information of the readable font. The reading of all font programs can be time consuming and in most usage scenarios needs only to be done once as operating system fonts do not change regularly. As a result, the manager can import and export the collected font information using the **`getFontProperties()`** and **`setFontProperties()`** methods respectively.

A basic `FontPropertiesManager` class is available in the package **`org.icepdf.core.ri.util`**.

Adding Font Paths to the FontManager

This page last changed on Jan 27, 2010 by mark.collette.

If you want to specify additional font paths to be ready by the FontManager class, it is possible, using the **`readSystemFonts()`** method.

The following code demonstrates how to add more font paths to the FontManager's internal list of paths:

```
String[] extraFontPaths = new String[] {"f:\\windows\\fonts\\", "f:\\winnt\\fonts\\"};
FontManager fontManager = FontManager.getInstance();
fontManager.readSystemFonts(extraFontPaths);
```

Memory Management and Caching

This page last changed on Jan 27, 2010 by mark.collette.

An Adobe PDF file consists of numerous compressed streams. As a PDF file is opened and additional pages are viewed, more streams are decompressed, and the memory required to display the content grows significantly.

When the amount of free Java heap memory becomes low, the ICEpdf Memory Manager frees up memory by disposing of the PDF objects associated with previously viewed pages, until the required amount of memory is recovered.

> **Note**
> Memory is considered low if the Runtime maxMemory value, minus the current amount of used Java heap, is less than the specified **`org.icepdf.core.minMemory`** value.

When an image stream is encountered, it is first represented as a byte stream and then encoded into a viewable image. This process can be time consuming and memory intensive. The image is stored in memory until the Memory Manager disposes of its parent page resources, at which time the encoded image may be written to disk and the representation of the image in memory is flushed. If needed later, the cached image is read from disk, which may be more efficient than re-decoding the byte stream and re-encoding the image.

> **Note**
> In some environments, file caching may not be desirable. If necessary, you can turn off caching completely by setting the system property **`org.icepdf.core.imagecache.enabled`** to false. However, you should not disable file caching if your PDF documents contain large images or several images unless you have a large Java Heap available (512 MB or more).

For more information on the ICEpdf system properties, see **System Properties**.

Internationalization

This page last changed on Jan 28, 2010 by patrick.corless.

The ICEpdf Viewer RI provides support for internationalization so that it can easily be adapted (localized) to various languages and regions. Internationalization is implemented using standard Java 2 Internationalization mechanisms.

Preliminary support for Danish, German, Spanish, Finish, French, Italian, Dutch, Norwegian, Portuguese and Swedish. This includes the default menus and toolbars but does not include dialogs and the utility panels. In the future support for dialogs and utility pane is expected.

The ICEpdf Viewer RI stores language bundles in the package, **`org.icepdf.ri.resources`**.