Outside In Clean Content SDK
Technical Note

PDF Extraction and Analysis Support

Version 1.2 – June 2, 2010

Change History

Version 1.0 – January 18, 2008
Initial version

Version 1.1 – November 10, 2008
Updated to include description of image extraction support

Version 1.2 – June 2, 2010
Updated to include description of features added with the release of Clean Content 2010.1

Overview

The Adobe Portable Document Format (PDF) is a rich and complex file format that has become a de facto standard for the exchange of electronic information. The ability to extract useful content from PDF documents is a requirement of many software applications. This document outlines the PDF extraction features supported by the Outside In Clean Content SDK.

PDF Extraction Features

File Identification and Version Coverage

Clean Content supports identification and processing of all versions of PDF through version 1.8 and applicable extensions of the file format. This is the version associated with Acrobat Professional 9. Future versions that may be released later than 1.8 require validation upon release. Clean Content will use the PDF signature and the PDF version explicitly stated in the PDF Catalog in order to identify the correct version and extension level.

In addition to recognizing the standard and required PDF signatures at the top of a PDF file, Clean Content will detect and support PDF documents that include a 128-byte non-standard header at the top of the file. This header may be found in PDF documents that have been processed in a Mac environment.

Malformed Document Support

There are many ways in which a PDF document can violate the file format specification. Two common causes include document truncation and the dropping of specific byte values from the document. Both of these likely occur during the transport of PDF documents. A third cause of malformed documents occurs when PDF generation software simply creates documents that do not follow the specification at some level. However, in many cases the Acrobat Reader can successfully recover the malformed document, occasionally displaying a warning though often simply ignoring the problem area. In some ways, this has allowed PDF generation software to create technically malformed PDF without awareness since Acrobat can load them fine.

Like Acrobat itself, Clean Content includes numerous recovery features that enable the extraction of content from otherwise malformed PDF documents. The first recovery feature involves the ability to regenerate an internal PDF structure that defines the PDF object hierarchy. This structure is often missing or broken in malformed documents. The second recovery feature is the ability to detect, trap, and recover from invalid PDF syntax. For example, if the processor encounters a violation of

the allowable PDF operations within a page, it will flag a warning to that effect and recover processing on the following page. Lastly, the component has specifically addressed some of the most common violations in order to recover immediately within the page where the error occurred. Each recovery is reported through the Clean Content logging feature.

Text Content and Unicode Character Mapping

The Clean Content PDF component includes robust support for converting PDF text content into Unicode. The process of converting PDF text to Unicode, while seemingly straightforward, is actually fairly complex. Each PDF Text operation includes a string of bytes that represent a series of character codes. The character code is used to select the glyph to be drawn from the current font. Converting each character code to its Unicode equivalent requires that the encoding of the font is well defined or that a Unicode mapping of its character codes is included in the PDF document. There is also the possibility that the characters are tagged with an alternate text replacement definition that overrides the text operation.

Most PDF creation tools follow the recommendations of the PDF specification with regard to creating PDF that allows consuming applications to correctly convert character content to Unicode. In fact, this is a requirement for a class of PDF documents known as Tagged PDF. Unfortunately, there are numerous cases where a PDF document can be created in such a way that it prevents the consuming application from being able to guarantee the mapping of characters to Unicode or any other known character set. In such cases, even though the glyphs are displayed perfectly, there is no way of knowing what character the glyph represents short of using OCR or some other heuristic technique. In many of these cases the consuming application can make certain assumptions about the character encoding that allows most characters to be correctly converted. However, even in such cases, the application cannot be sure that the mapping is correct.

The Clean Content PDF component supports correct conversion of character content to Unicode for all well-defined font encodings or when a Unicode map is provided. When the encoding is poorly defined the component will apply reasonable assumptions that improve the likelihood that the characters are converted correctly. Additionally, in an effort to identify poorly defined fonts, Clean Content will log debug information for font definitions that flag the most common cases of potentially incorrect character conversion.

It is extremely unfortunate, but the reality is that PDF generation tools have managed to create documents that have incorrect encoding definitions for nearly every example listed below.

The list below describes the specific character mapping use cases that can exist related to PDF font definitions.

To Unicode Map - The font definition includes an explicit mapping of character codes to Unicode. When a *To Unicode Map* is provided it will override any other encoding definition for the purpose of converting character codes to Unicode. In this case the encoding will still be used to correctly establish character metrics (width, encoding length …) but not for character mapping. In general, this is a very well defined use case that normally results in correct character conversion. However, there are examples where applications generate partial and even incorrect Unicode maps so even this case cannot always be trusted.

Named Encoding - The font definition uses one of the predefined simple encodings that include: *StandardEncoding, MacRomanEncoding, WinAnsiEncoding, PDF-DocEncoding,* and

MacExpertEncoding. This is a well-defined encoding that will result in valid character mapping unless the font definition has misrepresented the encoding and uses glyphs that don't match the stated encoding.

Named Encoding with Differences – This is an extension of the *Named Encoding* case that allows a font definition to leverage a simple encoding listed above while making adjustments that swap out some character codes for different glyphs. This is a well-defined encoding that requires that all differences be stated using the list of named characters in the *Adobe Glyph List*. If any of the differences use names that are not defined in the *Adobe Glyph List* then the case will be treated as a *Named Encoding with Unknown Differences.*

Named CMap with Known Collection – This is a well-defined encoding that is used in composite fonts, particularly for Chinese, Japanese, and Korean character sets. The named encoding matches one of the pre-defined PDF CMaps listed in table 5.15 of the PDF specification and called out below. These fonts leverage one of the Adobe registered character collections and its associated mapping to Unicode. Supported registered collections include the Adobe-GB1, Adobe-CNS1, Adobe-Japan1, and Adobe-Korea1 collections. The list of supported pre-defined Cmaps includes all of those documented in the PDF specification through version 1.7.
Simplified Chinese

Simplified Chinese

GB-EUC-H	Microsoft Code Page 936 (lfCharSet 0x86), GB 2312-80 character set, EUC-CN encoding
GB-EUC-V	Vertical version of GB-EUC-H
GBpc-EUC-H	Mac OS, GB 2312-80 character set, EUC-CN encoding, Script Manager code 19
GBpc-EUC-V	Vertical version of GBpc-EUC-H
GBK-EUC-H	Microsoft Code Page 936 (lfCharSet 0x86), GBK character set, GBK encoding
GBK-EUC-V	Vertical version of GBK-EUC-H
GBKp-EUC-H	Same as GBK-EUC-H but replaces half-width Latin characters with proportional forms and maps character code 0x24 to a dollar sign ($) instead of a yuan symbol (¥)
GBKp-EUC-V	Vertical version of GBKp-EUC-H
GBK2K-H	GB 18030-2000 character set, mixed 1-, 2-, and 4-byte encoding
GBK2K-V	Vertical version of GBK2K-H
UniGB-UCS2-H	Unicode (UCS-2) encoding for the Adobe-GB1 character collection
UniGB-UCS2-V	Vertical version of UniGB-UCS2-H
UniGB-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-GB1 character collection; contains mappings for all characters in the GB18030-2000 character set

UniGB-UTF16-V	Vertical version of UniGB-UTF16-H

Chinese (Traditional)

B5pc-H	Mac OS, Big Five character set, Big Five encoding, Script Manager code 2
B5pc-V	Vertical version of B5pc-H
HKscs-B5-H	Hong Kong SCS, an extension to the Big Five character set and encoding
HKscs-B5-V	Vertical version of HKscs-B5-H
ETen-B5-H	Microsoft Code Page 950 (lfCharSet 0x88), Big Five character set with ETen extensions
ETen-B5-V	Vertical version of ETen-B5-H
ETenms-B5-H	Same as ETen-B5-H but replaces half-width Latin characters with proportional forms
ETenms-B5-V	Vertical version of ETenms-B5-H
CNS-EUC-H	CNS 11643-1992 character set, EUC-TW encoding
CNS-EUC-V	Vertical version of CNS-EUC-H
UniCNS-UCS2-H	Unicode (UCS-2) encoding for the Adobe-CNS1 character collection
UniCNS-UCS2-V	Vertical version of UniCNS-UCS2-H
UniCNS-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-CNS1 character collection; contains mappings for all the characters in the HKSCS-2001 character set and contains both 2- and 4-byte character codes
UniCNS-UTF16-V	Vertical version of UniCNS-UTF16-H

Japanese

83pv-RKSJ-H	Mac OS, JIS X 0208 character set with KanjiTalk6 extensions, Shift-JIS encoding, Script Manager code 1
90ms-RKSJ-H	Microsoft Code Page 932 (lfCharSet 0x80), JIS X 0208 character set with NEC and IBM®extensions
90ms-RKSJ-V	Vertical version of 90ms-RKSJ-H
90msp-RKSJ-H	Same as 90ms-RKSJ-H but replaces half-width Latin characters with proportional forms
90msp-RKSJ-V	Vertical version of 90msp-RKSJ-H
90pv-RKSJ-H	Mac OS, JIS X 0208 character set with KanjiTalk7 extensions, Shift-JIS encoding, Script Manager code 1
Add-RKSJ-H	JIS X 0208 character set with Fujitsu FMR extensions, Shift-JIS

	encoding
Add-RKSJ-V	Vertical version of Add-RKSJ-H
EUC-H	JIS X 0208 character set, EUC-JP encoding
EUC-V	Vertical version of EUC-H
Ext-RKSJ-H	JIS C 6226 (JIS78) character set with NEC extensions, Shift-JIS encoding
Ext-RKSJ-V	Vertical version of Ext-RKSJ-H
H	JIS X 0208 character set, ISO-2022-JP encoding
V	Vertical version of H
UniJIS-UCS2-H	Unicode (UCS-2) encoding for the Adobe-Japan1 character collection
UniJIS-UCS2-V	Vertical version of UniJIS-UCS2-H
UniJIS-UCS2-HW-H	Same as UniJIS-UCS2-H but replaces proportional Latin characters with half-width forms
UniJIS-UCS2-HW-V	Vertical version of UniJIS-UCS2-HW-H
UniJIS-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-Japan1 character collection; contains mappings for all characters in the JIS X 0213:1000 character set
UniJIS-UTF16-V	Vertical version of UniJIS-UTF16-H

Korean

KSC-EUC-H	KS X 1001:1992 character set, EUC-KR encoding
KSC-EUC-V	Vertical version of KSC-EUC-H
KSCms-UHC-H	Microsoft Code Page 949 (lfCharSet 0x81), KS X 1001:1992 character set plus 8822 additional hangul, Unified Hangul Code (UHC) encoding
KSCms-UHC-V	Vertical version of KSCms-UHC-H
KSCms-UHC-HW-H	Same as KSCms-UHC-H but replaces proportional Latin characters with half-width forms
KSCms-UHC-HW-V	Vertical version of KSCms-UHC-HW-H
KSCpc-EUC-H	Mac OS, KS X 1001:1992 character set with Mac OS KH extensions, Script Manager Code 3
UniKS-UCS2-H	Unicode (UCS-2) encoding for the Adobe-Korea1 character collection
UniKS-UCS2-V	Vertical version of UniKS-UCS2-H

UniKS-UTF16-H	Unicode (UTF-16BE) encoding for the Adobe-Korea1 character collection
UniKS-UTF16-V	Vertical version of UniKS-UTF16-H

CMap with Known Collection- This is a well-defined encoding that provides an explicit mapping of character codes to one of the registered CJK character collections.

Identity Encoding with Known Collection – This is a well-defined encoding that uses character codes that map directly to one of the registered CJK character collections.

Implicit Encoding based on Font Name – This is a well-defined encoding based on using the font name to determine the encoding. It specifically allows for correctly mapping the ZapfDingbats and Symbol fonts that are included in what is commonly referred to as the base14 fonts. Clean Content also includes support for the Wingdings font under this use case.

Named Encoding with Unknown Differences – This type of encoding cannot be trusted to result in correct character conversion to Unicode. It indicates that the font declaration leverages a simple named encoding but includes a list of differences containing character names that do not match a name from the *Adobe Glyph List.* It is not all that uncommon for some PDF generation tools to obfuscate the glyph names when embedding a subset of a font. While this does not adversely affect the display of the glyph when the document is viewed, it does prevent the consuming application from determining what character the glyph represents. It turns out that many applications, though renaming the character to an ambiguous name when font sub-setting, have maintained the original simple encoding. Clean Content applies this assumption to improve the likelihood that the characters are correctly converted. Unfortunately use of this encoding may often result in incorrect character conversion.

Named CMap with Unknown Collection – This type of encoding indicates a named pre-defined encoding combined with a character collection that is not known to the component. The component will make an assumption that the target collection is Unicode. The testing of Clean Content against a large set of documents has not resulted in any known real life examples of this use case but its possibility is accounted for.

CMap with Unknown Collection – This character encoding is extremely uncommon, but cannot be trusted for correct character conversion. It indicates that the font is mapping character codes to values in a collection that is not known to the component and therefore not an Adobe registered character collection. A few examples of this have been seen in CJK documents using embedded fonts to support symbol characters.

Identity Encoding with Unknown Collection – The identity encoding is commonly used in a composite font to directly select a glyph from the font using a 2-byte code. The character collection is often declared as the Adobe-Identity collection, which simply indicates that the character code can be used to directly select the glyph but does not indicate the character set being used. This encoding cannot be trusted to result in a correct conversion of text to Unicode because the character collection being leveraged is not a well-defined collection. When encountering this encoding, Clean Content will assume that the target collection is Unicode in order to improve the likelihood of correct character conversion.

Symbolic Font with Unknown Encoding – This encoding is reasonably common and cannot be trusted. It indicates that the font contains symbol characters that are not part of the simple encoding that would otherwise be in place. However, it is also common that the majority of the characters do match the default standard encoding. Clean Content will fall back to the default encoding in order to improve the likelihood that characters are correctly converted under this encoding.

Word Boundary Structure Enhancement

The origin of PDF as a page description language that was focused on accurate and consistent display across many platforms and devices has contributed to a significant lack of structure within PDF documents. The text drawn on a page of PDF can often be characterized as a sequence of glyphs drawn at x/y locations on the page canvas without any regard for structure. For example, there is usually no information to indicate where a paragraph starts and ends, or whether content is part of a header, footer, or footnote, table, column, or caption. Even worse, the content may not even contain spaces between words. The Word Boundary Structure Enhancement feature is specifically designed to address the lack of spaces between words by monitoring the PDF drawing operations in an attempt to infer word boundaries and generate space characters accordingly.

Clean Content infers the location of spaces by continuously projecting where the next text operation would be drawn if a space were added to the previous text operation. If the next character falls at or beyond a tolerance factor of that location then a space is inferred. If the previous character was already a space, soft hyphen, or hard hyphen, then a space is not inferred since those are already valid word boundaries.

Care is taken to ensure that this algorithm is effective regardless of the rotation, skewing, scaling, font size, kerning, and character spacing applied to the text. Numerous special cases are addressed. Initial caps applied to the first character of a paragraph, superscripting and subscripting, shifting from a small font to a large font, are all examples where the distance from one piece of text to another has special consideration.

There are three situations where this algorithm is disabled.

- If the document is marked as tagged PDF then this algorithm is not applied. By definition, tagged PDF is required to be well formed with respect to including sufficient structure to establish word boundaries without the need to infer spaces.
- This feature is also disabled when a font definition lacks the character metrics required to establish the width of each character. By specification, a font is required to supply the character widths unless it is one of the Base 14 fonts that have predefined width tables. However, though uncommon and not recommended, it is possible for a PDF document to rely entirely on Acrobats ability to locate and leverage an operating environment dependent font for its character metrics. In these cases Clean Content cannot be certain of the exact location of each character and disables this algorithm to avoid inferring spaces incorrectly.
- Lastly, Clean Content does not attempt to infer spaces for text drawn in a vertical writing mode, instead relying on the PDF creator to include valid space characters.

Line Boundary Structure Enhancement

As mentioned earlier, PDF generally is lacking in document structure. Clean Content includes a feature designed to infer line elements based on the location of text pieces. Line structure by itself is

of limited value, but it is a requirement of several features that depend upon it. Specifically, word boundary detection, hyphenation adjustment, overlapping text filtering, and Arabic/Hebrew text re-ordering all rely on line detection to varying degrees. Providing the line structure also has the benefit of making the extracted output slightly more manageable to read. It is not necessary to treat line elements as a word-boundary because inferred spaces will be added at the end of the line where applicable.

Hyphenation Adjustment

Hyphenation adjustment is an optional feature designed to remove soft and hard hyphens found at the end of a line. This feature is designed to cleanup the extracted output by removing hyphens that were likely the result of an automatic hyphenation process. Unfortunately, the vast majority of hyphens found at the end of a line in PDF are output as hard hyphens even though they originated as syllable hyphens in the authoring application. This makes it impossible to distinguish between what was intended to be a hard or soft hyphen.

This feature is optional and is off by default because consuming applications with linguistic analysis features may prefer to differentiate soft and hard hyphens using a combination of linguistic analysis and the line position of the hyphen rather than have Clean Content make this determination based solely on line position. This is true because hard hyphens can be valid at the end of a line and should not be removed in such cases.

Tagged PDF and Paragraph Boundary Support

The PDF file format includes a feature called tagged PDF that allows PDF authoring applications to include a rich structure hierarchy that tags the text of the document beneath a hierarchy of elements. For example, the document may include tags that declare the content of headers, footers, hyperlinks, tables, paragraphs, formatted text spans, and many other structures. However, the use of this feature is often application specific and often lacks a consistent form of structure. For this reason Clean Content only leverages the use of paragraph elements when generating the extracted output. Paragraph elements will be passed through to the extracted output but all other elements will be ignored. Paragraph elements should be treated as valid word boundaries.

Overlapped Text Filtering

It is not uncommon for PDF creators to produce multiple layers of identical text for the purpose of accomplishing specific graphic effects. For example, text shadowing, 3D text, embossing, and other effects may be accomplished by writing each character multiple times, with slight position, transformation, and font attribute changes. Unfiltered extraction of this situation results in duplicate characters, words, or lines in the extracted output that is far from ideal for any subsequent text analysis. This feature attempts to detect this type of overlapping and clean up the extracted output so that only a single instance of each overlapped character is generated, thus improving the search-ability of the output. This feature is turned off by default due to the performance cost of testing every line for overlapping conditions, and because specific use cases exist where only some of the overlapped characters within a line fall within the overlapping tolerance resulting in only partial cleanup of the line.

Reading Order Extraction of Middle Eastern Scripts (i.e. Arabic and Hebrew)

Most applications produce text that is read from right to left (i.e. Arabic and Hebrew) by drawing it from left to right. In such cases, extracting the text in the order that it is drawn will result in lines of text being in reverse reading order when extracted. This feature will automatically detect the use of characters found in scripts that read from right to left and then validate or re-order the characters within the line to match the proper reading order. This algorithm includes consideration for mixed mode writing that includes words read from both right to left and left to right. There are specific mixed mode use cases where the word ordering cannot be properly established without linguistic consideration not included in this algorithm. In such cases the text of each word will be correctly ordered but specific words within the line may be out of order. This algorithm is dependent on the accuracy of the line boundary algorithm since re-ordering must be done a line at a time.

Comment/Annotation Support

The PDF format includes the ability to layer annotations of many forms over the top of a page. Clean Content supports extracting annotations that contain text, file attachments, and URL links. Clean Content also identifies the type of annotation extracted. Certain types of annotations include an appearance stream that describes how the text is drawn while others simply include the text and leave the display format to the viewing application. Both types are supported by Clean Content during extraction. Also, text found within appearance stream based annotations can be highlighted when implementing Acrobat highlighting support referenced later in this document. Annotations that do not contain text, attachment content, or URL's are intentionally not extracted.

The annotations that will be extracted by Clean Content when they include some supported form of content include the following types: 3D, Caret, Circle, File Attachment, Free Text, Highlight, Ink, Line, Link, Polygon, PolyLine, Printer Mark, Square, Squiggly, Stamp, Strike Out, Text, TrapNet, Underline, Watermark, Widget, Popup, Sound, Movie, Screen. Only text, URL's, and attachment content will be extracted with the associated annotations, Movie and sounds objects are not extracted in the current release of Clean Content.

Standard Document Property Support

The PDF file format can include document properties in two different ways. The first is through the document information dictionary with specific entries for Title, Author, Subject, Keywords, Creator, Producer, Creation Date, Modification Date, and a field that indicates whether the document was trapped (a technique used to address certain visual artifacts that occur in some printing environments). This feature supports extraction of these properties into a property collection during the extraction process. The second method of handling document properties is outlined below.

XMP Document Property Support

In addition to the standard document property storage format, PDF allows an XML metadata stream to be attached to nearly any content component in the document. The format of the XML is XMP (Extensible Metadata Platform) making it very customizable. In PDF, document property metadata is commonly attached to the PDF Document Catalog. Clean Content supports exporting the XMP metadata stream that is attached to the document catalog to a stream accessible to the consumer for

further processing. An XML header is added to the XMP stream in order to allow the XMP to become a valid XML stand-alone file. Optional control over the extraction of the XMP stream is provided through the Clean Content options that control embedded content extraction.

Clean Content only reports and exports the XMP stream attached to the PDF document catalog. XMP streams can also be attached to many other content elements including pages and images but these XMP streams often lack any useful properties and are not the standard method of including document wide properties. Support for extraction of these additional XMP streams has been intentionally disabled for the sake of performance and clarity. This decision can be reconsidered should valid use cases be encountered.

Acrobat Highlighting Support

The Acrobat Professional and Acrobat Reader applications provide a useful API that enables web applications to highlight text found in a PDF document when the document is viewed in Acrobat and is served as a URL. This is done by including a reference to an additional file as a parameter to the URL. This additional file is referred to as a highlight file that specifies the text locations to be highlighted. Enabling this feature requires a component that can extract the text from a PDF document while properly tracking the page and highlight location for every piece of text in the file.

Clean Content includes the ability to provide the highlight locations for every piece of PDF text that can be highlighted. This allows applications that include Clean Content to develop a powerful search and highlight feature for PDF documents. The SDK includes sample source code that demonstrates how to generate the Acrobat highlight file and further demonstrates an implementation of this feature through the SDK demo application. A detailed description of the PDF highlight algorithm and Clean Content usage notes is documented in the *PDF Highlight Algorithm Tech Note* included in the Clean Content SDK.

File Attachments

The PDF file format supports the ability to attach documents of any type to a PDF document. Clean Content supports exporting attachments to stand-alone files as well as the ability to recursively extract content from attached files. The ability to recursively extract content requires that the attachment be in one of the formats generally supported by Clean Content. Support is included for files attached to the document as well as files attached using a comment.

PDF Packages and Portfolios

Acrobat Professional includes the ability to combine a set of PDF documents into a PDF package. This feature was substantially enhanced and renamed Portfolios with in Acrobat 8 and beyond. The resulting PDF includes a cover sheet for the package and provides the ability to access each packaged document independently. Similar to attachments, Clean Content supports exporting the embedded files to stand-alone files and the ability to recursively extract content from all files in the package that are supported by Clean Content.

Encrypted File Support

The PDF file format includes many security features, one of which is the ability to encrypt the document. PDF allows many different encryption algorithms. The encryption may leverage a user or owner password or may simply encrypt the document with a default password. Clean Content

automatically detects and decrypts documents leveraging RC4 or AES encryption that have been encrypted with a default password and standard security handler. This covers most cases where an encrypted PDF document can be opened without a user password. If a PDF document uses an unsupported encryption algorithm or requires a password then Clean Content will flag the document as encrypted and log a warning to that affect.

Clean Content leverages the Java™ Cryptography Extensions (JCE) for AES decryption. Due to import control restrictions the version of the JCE policy files that are bundled with the JRE that ships with Clean Content allow 'strong' but limited cryptography. If a PDF document is encrypted with AES 256 bit encryption it can only be decrypted if the Java Runtime Environment leveraged by Clean Content is updated to include the Unlimited Strength Java™ Cryptography Policy Files. These files can be downloaded from the Sun Developer Network and installed into the appropriate JRE location as documented in the download.

Raster Image Extraction

Raster images within the PDF file format can be implemented using a variety of methods. Unlike many other formats, PDF does not rely upon industry standard image file formats for embedding raster images. Instead, PDF has its own internal structures that combine to provide raster image support. The description of an image generally involves a stream of image data, a set of filters applied to the image data, a color space associated with the image, and additional options that specify the format of the image data.

The goal of the Clean Content image extraction feature is to allow images to be extracted from the source document into an industry standard file format so that the image can be repurposed as needed. An ancillary goal is to limit the amount of conversion on the image data itself when exporting the image to a stand-alone file. Limiting the conversion of image data is done to maximize performance, avoid device dependent color conversion, and to maintain the original form of image data so that stegongraphic detection may be implemented on the original image data.

As of version 1.7 of the PDF file format (Acrobat 8/9), there are nine different stream filters and multiple encryption filters that could be applied to the image data any number of times or not at all. The nine stream filters include ASCIIHexDecode, ASCII85Decode, LZWDecode, FlateDecode, RunLengthDecode, CCITTFaxDecode, JBIG2Decode, DCTDecode, and JPXDecode. Additionally, each filter may have specific options that tune the filter in various ways. Also note that PDF supports TIFF and PNG predictor filters as options on Flate and LZW. TIFF prediction will be maintained when going to TIFF but PNG predictors will applied to the data since TIFF does not support them. The final filter remaining after applying decryption, ASCII decoding, and reduction to at most one filter, determines the industry file format that will be used for extraction as outlined in the table below.

Final Image Filter	**Target Image File Format**
No Filter (Raw image data)	TIFF (with no compression)
LZWDecode	TIFF (with LZW compression)
FlateDecode	TIFF (with Flate compression)
RunLengthDecode	TIFF (with Packbits compression)

CCITTFaxDecode	TIFF (with CCITT compression)
JBIG2Decode	JBIG2 stand-alone file (JB2)
DCTDecode	TIFF (with JPEG compression)
JPXDecode	JPEG2000 (JPX)

This approach minimizes the amount of decompression and color conversion, often allowing for a straight copy of the image data to the target file.

The PDF file format supports a wide variety of color spaces that can be applied to raster images. These include DeviceGray, DeviceRGB, DeviceCMYK, CalGray, CalRGB, Lab, ICCBased, Indexed, Separation, and DeviceN. The Indexed color space may index into any of the other color spaces except Indexed, Separation, and DeviceN effectively allowing for 7 Index color spaces and a total of 16 different color spaces. Both JPX and JBIG2 inherently define the applicable color space within the image data. When converting to TIFF it is necessary to either select a matching TIFF color space or, in some cases, transform the image data and/or color space to a reasonable alternative. The table below outlines the color space conversion details when exporting to TIFF.

PDF Color Space	**TIFF Color Space**
DeviceGray	Grayscale (WhiteIsZero or BlackIsZero depending on applicable PDF image parameters)
DeviceRGB	RGB
DeviceCMYK	CMYK
CalGray	Grayscale (WhiteIsZero or BlackIsZero depending on applicable PDF image parameters) - Note, the resulting image is effectively device dependent rather than explicitly calibrated.
CalRGB	RGB - Note, the resulting image is effectively device dependent rather than explicitly calibrated.
Lab	ICCLab
ICCBased	The number of color components determines the TIFF color space. 1=Grayscale, 3=RGB, 4=CMYK. The embedded ICC Profile is transferred to the TIFF file.
Separation	Grayscale – the image data is copied without conversion and pixel values are effectively matched to a shade of gray by applying a grayscale color space. This may result in unintended color application.

DeviceN	Grayscale - the image data is copied without conversion and pixel values are effectively matched to a shade of gray by applying a grayscale color space. This may result in unintended color application.
Indexed DeviceGray	PaletteRGB with grayscale color entries.
Indexed DeviceRGB	PaletteRGB
Indexed DeviceCMYK	CMYK - image data is expanded to continuous tone and then recompressed using Flate because TIFF does not support this palette type.
Indexed CalGray	PaletteRGB with grayscale color entries.
Indexed CalRGB	PaletteRGB
Indexed Lab	Lab - image data is expanded to continuous tone and then recompressed using Flate because TIFF does not support this palette type.
Indexed ICCBased	Grayscale, Palette RGB, or CMYK depending on the number of color components . Grayscale and CMYK image data is expanded to continuous tone and then recompressed using Flate because TIFF does not support this palette type.

PDF allows images to be defined as either a named image resource or as an inline image. Clean Content currently only exports named image resources. This is done because inline images are intended to be used for only very small images (less than 4K) and are commonly used for graphic elements like table borders or bullets. Ignoring inline images is a reasonable way to avoid exporting a substantial amount of images that are not intended to be repurposed. This decision may be revisited based on customer feedback.

It is reasonably common to find images within PDF that are stored upside down. Such images are often flipped vertically when drawn on the page using a PDF transformation operation that is independent of the image data. A single named image resource may be drawn any number of times using any number of transformations making resolution of this issue less than exact. Clean Content does not currently consider such transformations when extracting the image to a stand-alone file. This decision may be revisited based on customer feedback.

Macros and Code (Javascript) Detection

PDF allows JavaScript to be executed through numerous mechanisms. This feature of PDF has been identified as a possible security risk. Clean Content will detect and flag the existence of JavaScript under the Macros and Code target. Document level JavaScript actions are found in the DocumentCatalog.Names.JavaScript name tree that maps name strings to JavaScript Actions. JavaScript defined in this manner typically is executed when the document is opened, providing

methods that may be called from other JavaScript in the document. JavaScript action dictionaries can be found as the K, F, V, and C entries in the Additional Actions dictionary that is referenced from a Form Field Dictionary AA entry. JavaScript action dictionaries may also be found in the WC, WS, DS, WP, and DP, entries of the Additional Actions dictionary referenced from the Document Catalog dictionaries AA entry. The JS entry of an Action Dictionary of type Rendition is another mechanism for PDF to include JavaScript. Clean Content will report the existence of JavaScript found in any of these locations.

Fast Save Data (Incremental Updates) Detection

The PDF file format supports a feature known as Incremental Updates that allows PDF objects to be replaced with new versions or added to the file by appending the object and certain structure information to the end of the document. This is done to allow quickly saving small changes (i.e. replace or add one page) and in some cases to limit the complexity of modifying a PDF for limited changes (i.e. add a Watermark). The result of this feature is that a reasonable percent of PDF documents contain obsolete content and data. In many cases entire pages that have been visually removed or replaced are still stored and can be extracted from the document. Clean Content will detect the existence of obsolete content and report it as 'Fast Save Data'.

Document Outline Extraction

The PDF format includes the ability to describe a document outline that provides a hierarchy of outline items that reference particular destinations within the document. This feature works in tandem with a feature called “Structure Hierarchy” that effectively bookmarks locations within a document. The text of the outline is stored independently from the destination of the bookmark. Clean Content will extract the content and structure of the document outline during the extraction process.

Page and Embedded File Thumbnail Image Extraction

Each page of a PDF document may optionally include a thumbnail image. A thumbnail image may also be associated with a file embedded within a PDF. Clean Content extraction will produce a thumbnail element prior to the applicable page or embedded file during the extraction process. The thumbnail element includes an embeddedcontent element of type ‘Image’ that represents the thumbnail image. The form of the extracted image is consistent with the Raster Image Extraction feature described earlier.

Article Thread Information Extraction

PDF allows for the definition of article threads that provide the data needed to navigate an article that is logically connected but visually disconnected in the document. A page identifier and a rectangle on that page define the location of each bead of an article. The actual content of each article is extracted as part of the page on which it appears. Article threads may also contain information such as the title, author, subject, keywords, and creation date. Clean Content will generate a collection element of type ‘Article Thread’ and generate an articlethread element for each article in the document. The articlethread element will include string elements with the appropriate type that define the title, author, subject, and keywords, and will include a date element that defines the creation date. Note that each of these is optional and will only be generated when defined in the document.

Interactive Forms Extraction

PDF documents provide rich support for interactive forms. Fields are defined with a hierarchical structure. Clean Content detects and extracts form data by generating a collection element with a type of 'FormFields'. The collection is then populated with formfield elements that may in turn contain child formfield elements. Each formfield may contain various string elements that define the FIELDTYPE, FIELDNAME, FIELDALTNAME, and FIELDMAPPINGNAME. If the field is populated with a text value it is extracted as a text element that contains that applicable value.

XFA Forms Export to stand-alone file

PDF 1.5 added support for interactive forms based on XFA (XML Forms Architecture). The result is that an XFA resource may be stored inside the PDF document. XFA supports a more dynamic interactive forms architecture than standard PDF interactive forms. Clean Content treats the XFA resource as embedded content, allowing it to be identified and exported to a stand-alone file. This includes piecing together the XFA from multiple XDP packets as allowed within PDF.

FDF Extraction/Inspection

Adobe FDF (Forms Data Format) is a stand-alone file format that is a subset of the PDF file format. This format is designed to act as a transport format that allows exporting form fields, and importing form fields, new pages, and annotations into an existing PDF document. Clean Content 2010.1 added support for extraction and analysis of FDF documents. The extraction functionally matches PDF in all other features respects.