# Web-Friendly PDFs

with ngPDF


www.itextpdf.com

# PDFs as first-class citizens of the Web with ngPDF

The web has changed the way we look at documents. Increasingly, content is consumed on devices with wildly differing form factors. The need for content to adapt to diverging screen sizes will only increase as mobile devices displace traditional desktops and laptops. It is a common misconception that PDFs cannot be responsive. However, since the addition of Tagged PDF to PDF 1.4, PDF readers have had the possibility to reflow text depending on the screen size.


Be that as it may, users today have come to expect their documents to be accessible and legible on any of their devices, preferably through the use of a web browser. The excellent reputation of PDF for accurate representation of documents on screen as well as on paper, has caused it to be perceived to be out-of-place on the web.

At iText, we aim to be at the frontier of PDF technology. Through our continued involvement in the PDF Association and the ISO committee, we are working hard to make PDF a better citizen of the web. An algorithm recently developed by the PDF Association, makes great strides for turning well-crafted PDFs into responsive documents. The algorithm, as implemented in ngPDF and powered by iText 7, takes the idea of text reflow in PDF one step further, by automatically turning them into first-class HTML documents without manual intervention.

In this white paper co-authored with Dual Lab, we touch upon the history of semantic PDF; how we got from Tagged PDF to ngPDF today. The first part is accessible to a broader audience, the second part takes a deep dive into the ngPDF internals. Additionally, a tech preview of ngPDF is available at www.ngpdf.com. We'd very much like for you to give it a spin and let us know what you think.

# Contents

# 1

# Making PDFs a good citizen in the Open Web world

## 1.1 Introduction

One of the key propositions of the Portable Document Format is a reliable visual representation on all devices and operating systems. It is based on the fixed page layout model, where all text characters, all raster images and vector drawings have absolute positioning on the page.

This is very different from the Web world, where only the web browser knows how to lay out the web content on the screen of the user device. On the other hand, the internal XML-like structure of the HTML pages along with the actively developing WAI-ARIA standard does a very good job of presenting the semantics of the content.

The same idea of defining the semantics of PDF content was introduced back in PDF 1.4 as the concept of structure elements, the so-called Tagged PDF. In short, Tagged PDF defines the semantic hierarchy (often called semantic structure) of the complete PDF document, where each terminal structure element (for example, a paragraph or a span) in this hierarchy references the graphical elements on the page (or several pages) that represent this structure element.

Although this notion of Tagged PDF and its structure tree were introduced into PDF specifications more than 10 years ago, not many end users were aware of this additional data that could be present in their documents. One of the important applications of Tagged PDF today is the “Read out loud” feature that allows accessibility software to accurately describe the content of PDF documents to visually impaired users.

However, only accessibility specialists and PDF engineers were mainly inspecting and manipulating the structure tree of Tagged PDF documents.

## 1.2 Derivation algorithm

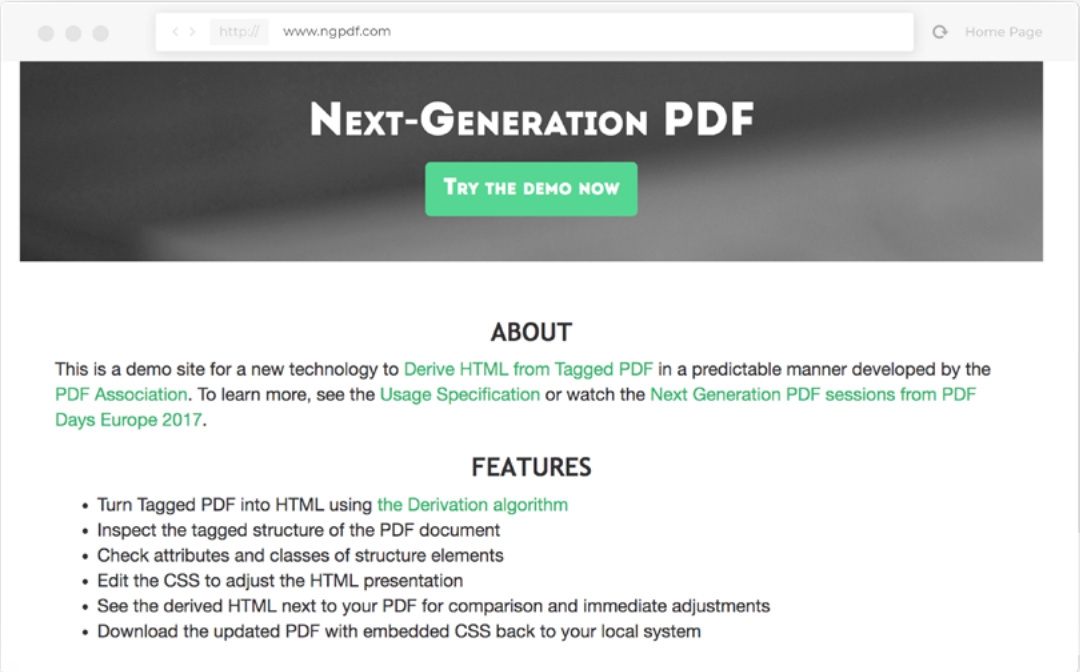
The Derivation algorithm was developed by the PDF Association to provide a reliable way for generating an HTML presentation of Tagged PDF documents based on their semantics. In short, this algorithm extracts the content of the PDF document based on its visual presentation and recombines it into HTML using the logical structure of the PDF. Although this might sound rather simple, it took several years for the group of PDF experts to specify all the details and make this approach well-defined. The full details of the Derivation algorithm can be found in the PDF Association publication "Deriving HTML from PDF".

## 1.3 How ngPDF does this

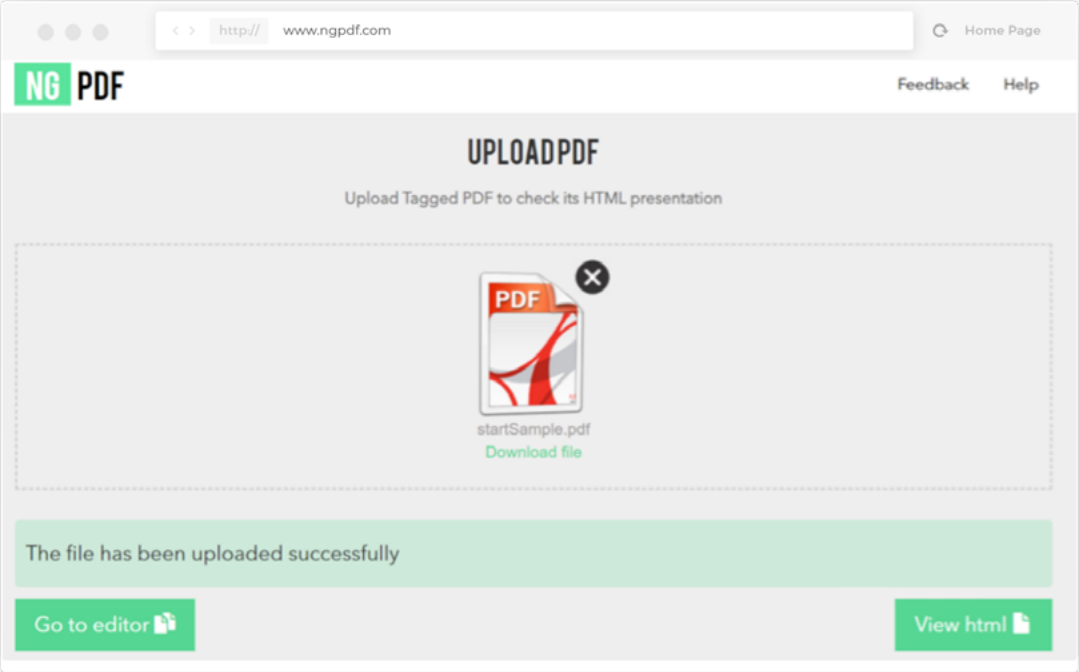
ngPDF is a web application freely available at www.ngpdf.com that allows users to upload any Tagged PDF and inspect the resulting HTML, as well as many other useful properties of Structure hierarchy right in their browser.

Here is the step by step illustration of how it works:

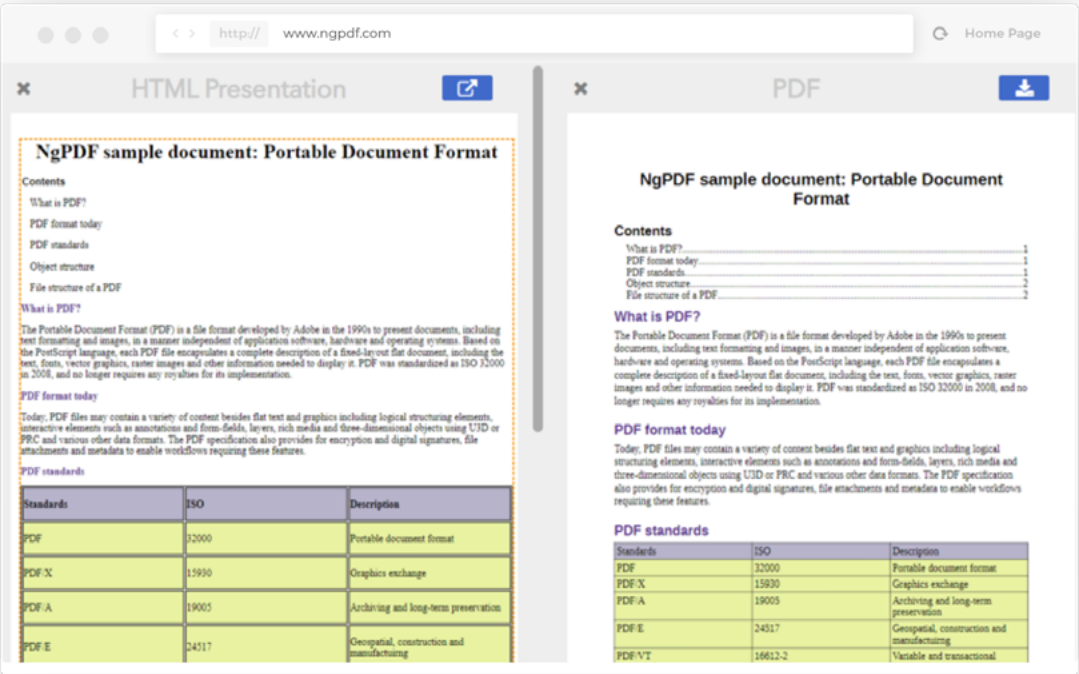
**Step 1:** Go to www.ngpdf.com and click the button "**Try the demo now**"



**Step 2:** Upload your PDF document. Note that it must be Tagged, **see the next section** if you do not know how to check this. Or simply choose a sample provided by the application.




**Step 3:** Click to "**View HTML**" to see the results of the derivation algorithm right away.
Or click "**Go to editor**" to compare the HTML and PDF visual presentations side by side.



## 1.4 What's inside: iText API

The derivation algorithm implemented inside ngPDF is based on the iText 7 library, which provides all the necessary ingredients for the correct derivation from PDF to HTML.

Below we list several code samples to demonstrate how iText does this. Note that the samples use Java, but equivalent functionality is also available in C#.

We can't just derive HTML from any old PDF. A first step in the implementation of the derivation algorithm, must be to check if the PDF is tagged. This is trivial to achieve with iText 7, as shown in Code Sample 1.

In a second step, the PDF document is then analyzed to gather information on its structure. The ngPDF implementation of the derivation algorithm uses iText to iterate over the document structure tree (see Code Sample 2). All structural elements and their attributes are then stored for processing so corresponding HTML can be emitted later.

Finally, after the output HTML document structure is known, ngPDF can proceed to extract the actual PDF contents. We define an event listener that knows how to process both text content and images as demonstrated in Code Sample 3. Every time `PdfCanvasProcessor` encounters an object, the event listener knows how to extract the underlying text or image.

### 1.4.1 Open the document and check if it is tagged

```
PdfDocument pdfDocument = new PdfDocument(new PdfReader(documentPath));
boolean documentIsTagged = pdfDocument.isTagged();
```

### 1.4.2 Iterate through the structure tree of the document and retrieve all structure element properties and attributes

```
PdfDocument pdfDocument = new PdfDocument(new PdfReader(pdfDocumentPath));
PdfStructTreeRoot root = pdfDocument.getStructTreeRoot();
iterateOverChildren(root);

private void iterateOverChildren(IStructureNode node) {
   if (node.getKids() != null) {
       for (IStructureNode child : node.getKids()) {
           if (child instanceof PdfMcr) {
               // This ID can be used to extract the associated page content
               int mcid = ((PdfMcr) child).getMcid();
           } else if (child instanceof PdfStructElem) {
               inspectAttributes((PdfStructElem) child);
           }
           iterateOverChildren(child);
       }
   }
}

private void inspectAttributes(PdfStructElem element) {
   // Example on how to resolve effective role (taking /RoleMap into consideration)
   String role = pdfDocument.getTagStructureContext().resolveMappingToStandardOrDomainSpecificRole(
           element.getRole().getValue(), element.getNamespace()).getRole();

   PdfObject attrObj = element.getAttributes(false);
   if (attrObj != null) {
       PdfDictionary attrDict;
       if (attrObj instanceof PdfArray) {
           attrDict = ((PdfArray) attrObj).getAsDictionary(0);
       } else {
           attrDict = (PdfDictionary) attrObj;
       }
       // Example on how to fetch Bbox attribute
       Rectangle bBox = attrDict.getAsRectangle(PdfName.BBox);
   }
}
```

### 1.4.3 Find the corresponding marked content sequences inside the page contents and extract its text or image content

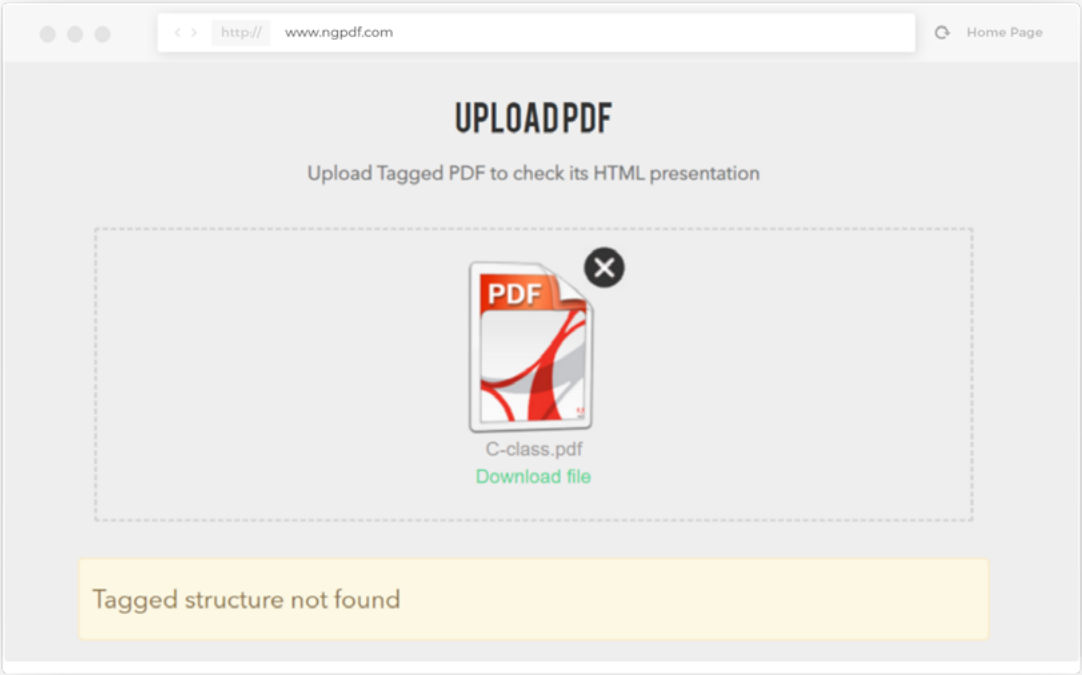
```
PdfDocument pdfDocument = new PdfDocument(new PdfReader(pdfDocumentPath));
MarkedContentEventListener listener = new MarkedContentEventListener();
new PdfCanvasProcessor(listener).processPageContent(pdfDocument.getPage(1));
Map<Integer, String> mcidToContent = listener.getMcidContent();

class MarkedContentEventListener implements IEventListener {
   private Map<Integer, ITextExtractionStrategy> contentByMcid = new HashMap<>();
   private Map<Integer, Collection<PdfImageXObject>> imagesByMcid = new HashMap<>();

   public Map<Integer, String> getMcidContent() {
       Map<Integer, String> content = new HashMap<>();
       for (int id : contentByMcid.keySet()) {
           content.put(id, contentByMcid.get(id).getResultantText());
       }
       return content;
   }

   @Override
   public void eventOccurred(IEventData data, EventType type) {
       switch (type) {
           case RENDER_TEXT:
               TextRenderInfo textInfo = (TextRenderInfo) data;
               int textMcid = textInfo.getMcid();
               if (textMcid != -1) {
                   ITextExtractionStrategy textExtractionStrategy = contentByMcid.get(textMcid);
                   if (textExtractionStrategy == null) {
                       textExtractionStrategy = new LocationTextExtractionStrategy();
                       contentByMcid.put(textMcid, textExtractionStrategy);
                   }
                   textExtractionStrategy.eventOccurred(data, type);
               }
               break;
           case RENDER_IMAGE:
               ImageRenderInfo imageRenderInfo = (ImageRenderInfo)data;
               int imageMcid = imageRenderInfo.getMcid();
               if (imageMcid != -1) {
                   Collection<PdfImageXObject> images = imagesByMcid.get(imageMcid);
                   if (images == null) {
                       images = new ArrayList<>();
                       images.add(imageRenderInfo.getImage());
                   }
               }
           default:
               break;
       }
   }

   @Override
   public Set<EventType> getSupportedEvents() {
       return new HashSet<>(Arrays.asList(EventType.RENDER_TEXT, EventType.RENDER_IMAGE));
   }
}
```

# 2

# How to understand that a PDF is Tagged

### 2.1 Use ngPDF to view the logical structure and its properties

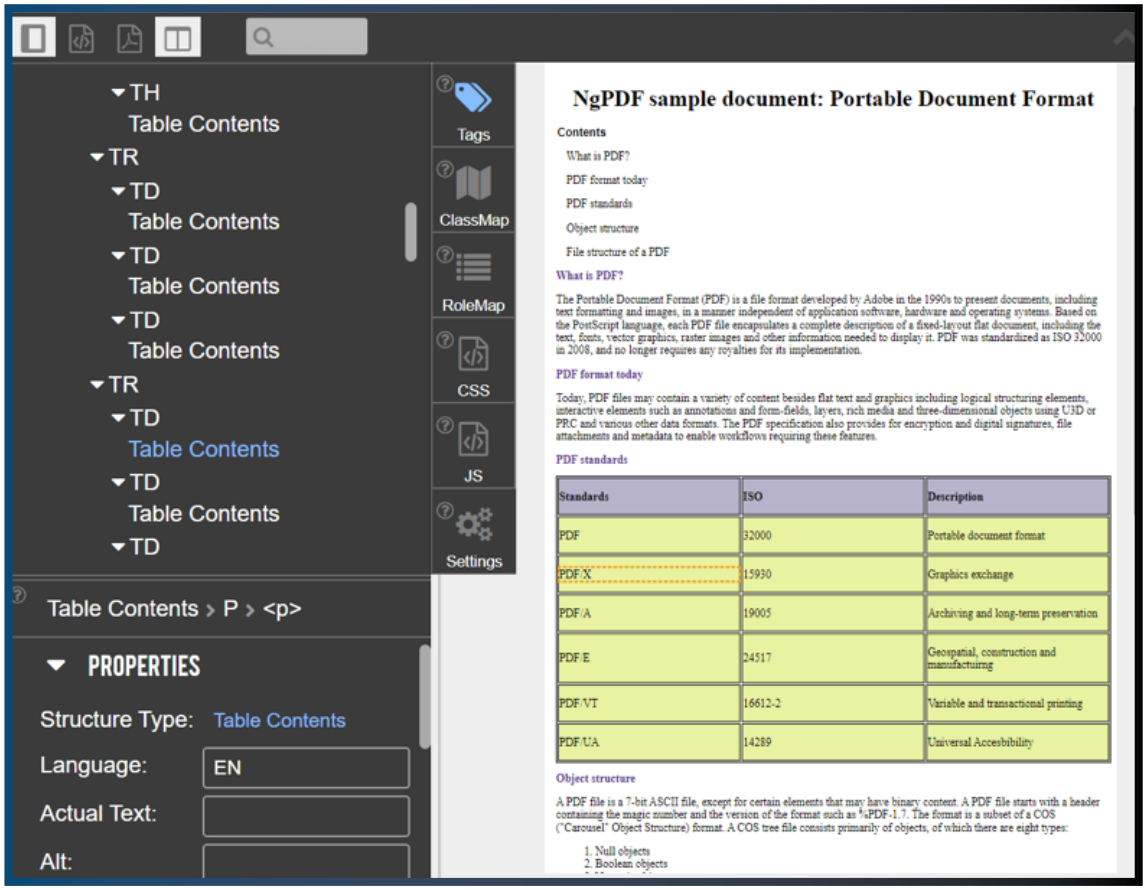
If you drop a random PDF file in your possession onto the ngPDF upload area, there is a high chance you will see the disappointing message "*Tagged structure not found*":



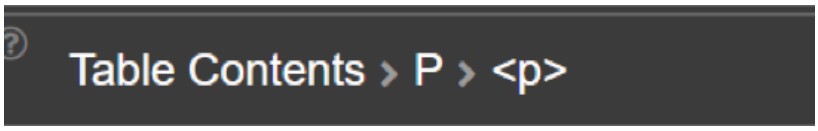

This means that your PDF document is not Tagged, i.e. it does not contain a logical structure hierarchy. This does not mean your document is corrupt. It will still be shown without any problems, but the software and assistive technologies will experience difficulty reading out your document. As the Derivation algorithm uses the same logical structure as

assistive technologies, it also won't work for documents with missing logical structure.

But if your PDF document is Tagged, the ngPDF Editor will provide you with detailed information on the logical structure tree. Its right hand side visualizes the complete structure tree as well as all the properties of its structure elements:




In addition, in the above screenshot we can also see the properties of the selected structure element and the idea how the Derivation algorithm works:

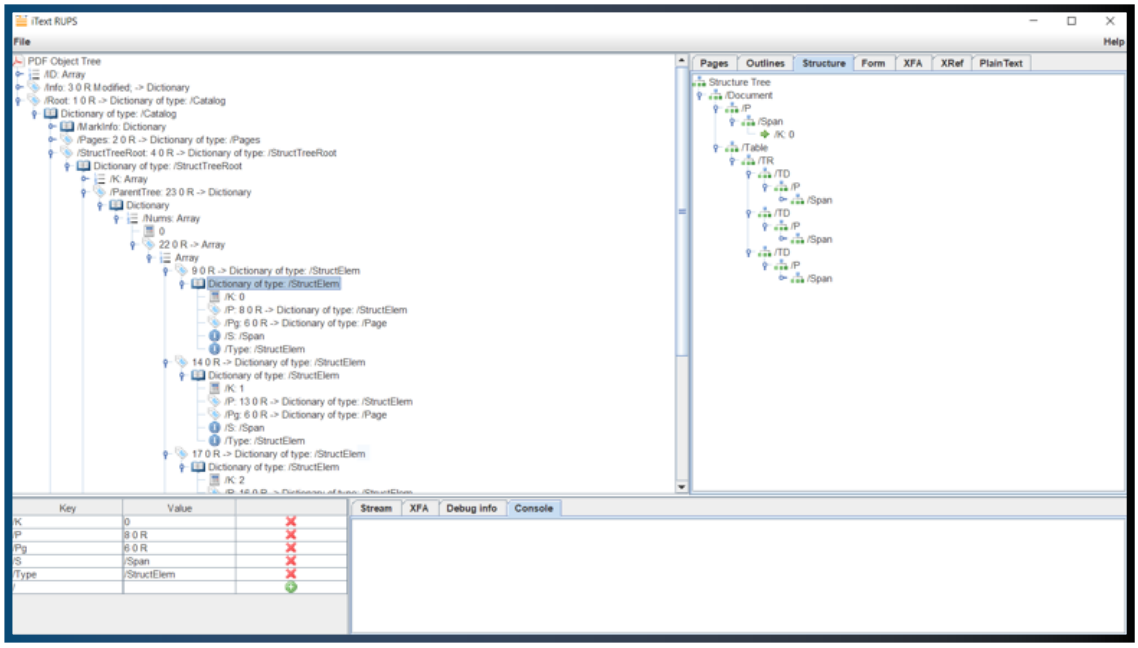



This means that the algorithm first maps the PDF tag **Table Contents** to another PDF tag **P**, which is finally mapped to the HTML tag **<p>**.

## 2.2 Use iText RUPS

iText RUPS is a free low-level PDF inspector available at https://github.com/itext/i7j-rups/releases

It allows users to inspect the complete structure tree of a document as well as low-level data for all its elements. Simply drop your PDF document into the iText RUPS window and choose the “Structure” tab located in the right pane of the application window.



# 3

# How to generate Tagged PDFs

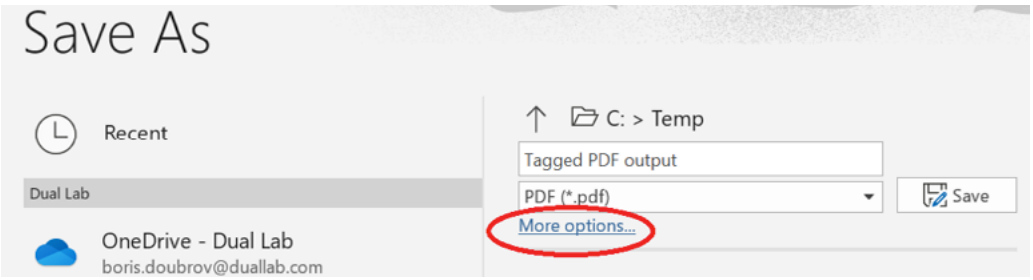
### 3.1 Via authoring software

Common options in terms of authoring software include popular office suites such as Microsoft Office or LibreOffice, or specialized design tools such as Adobe InDesign.

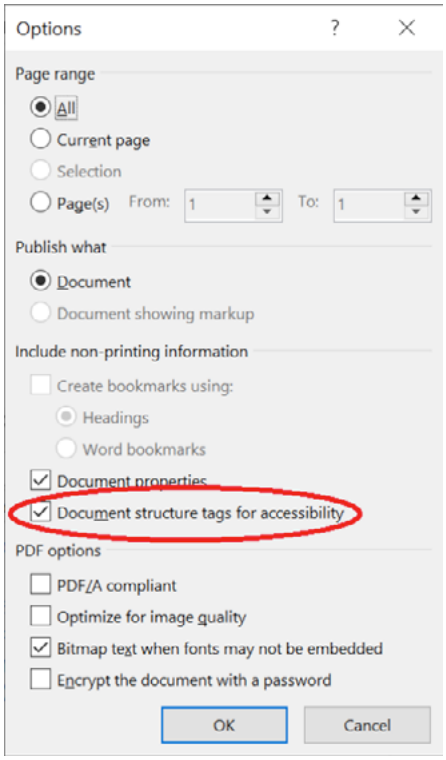
A very natural question one may ask is how to generate Tagged PDFs. Normally this is the responsibility of the authoring software that produces PDF documents to also insert the logical structure into these documents.

The good news is that many commonly used applications for creating PDF files are able to do this. But often this requires extra settings in your “Save As” or “Export to PDF” dialog.

Consider, for example, how this works in Microsoft Word. First, one needs to follow the “More options” link in the Save As dialog of MS Word. **Note that this link will appear only when you choose PDF as an output format!**



Next, the options dialog itself contains a checkbox “Document structure tags for accessibility” which must be checked for Microsoft Word to output Tagged PDFs.




### 3.2 Via the iText API

Using the high level API of iText it is really easy to create a tagged PDF document. The following code sample adds a paragraph and a table with 3 cells to an untagged document.

```
PdfDocument pdfDocument = new PdfDocument(new PdfWriter(outFileName));
Document document = new Document(pdfDocument);

document.add(new Paragraph("Hello world"));

Table table = new Table(3);
table.addCell("Cell 1");
table.addCell("Cell 2");
table.addCell("Cell 3");
document.add(table);

document.close();
```

Turning this code sample into one that creates a PDF file is easily done. It takes a single line of code that needs to be added to the code sample. You need to add the following line before adding any content:

```
pdfDocument.setTagged();
```

This will turn the output PDF into a tagged PDF for no additional effort. This is made possible because iText knows which objects you're adding to the Document and as such knows when to tag a Paragraph as "P".

```
PdfDocument pdfDocument = new PdfDocument(new PdfWriter(outFileName));
pdfDocument.setTagged();
Document document = new Document(pdfDocument);

document.add(new Paragraph("Hello world"));

Table table = new Table(3);
table.addCell("Cell 1");
table.addCell("Cell 2");
table.addCell("Cell 3");
document.add(table);

document.close();
```

This is also possible using iText's low level API, however, using the low level API for this is not advised as you would need extensive knowledge of the PDF specification, and would need to make several highly technical method calls compared to the easy solution provided by the high level API.

# 4

# Challenges

### 4.1 Features still in development or not yet supported

As it often happens with “first versions”, some points of the Derivation Algorithm are still under discussion. Fortunately, this will most likely concern only very specific cases not often seen in real-world documents.

One important PDF feature that is still missing is the HTML presentation of PDF annotations. Annotations are commonly used in PDF to add comments to a document; though in addition to text annotations there are also drawing annotations, multimedia annotations and more. The key difficulty here is that while there is now a standard for Web Annotations, fragmentation in this area and the lack of browser support means the accurate representation of PDF annotations is still an issue to be resolved.

This and other points will be addressed in the future updates of the Derivation Algorithm, which is already a work in progress for a dedicated Technical Working Group of the PDF Association.

ABOUT US

# Get to know iText


Think about a boarding pass for your flight. Or PDF invoices, receipts, forms...most likely they were generated by iText technology!

iText is a global leader in innovative award-winning PDF software. It is used by millions of users - both open source and commercial - around the world to create digital documents for a variety of purposes: invoices, credit card statements, mobile boarding passes, legal archiving and more.

iText works and works well. Our customers choose iText because of our world-class quality of software, and our reliable mature, proven technology in the iText SDK. We are recognized as a global thought leader and innovator in PDF solutions and functionalities. As an open-source PDF library, iText can be embedded into the document solution workflows of various industries and their applications.

Our diverse customer base includes many of the Fortune 500 companies, as well as small companies and government agencies. We strongly believe in the value of open-source software. Our core library, iText 7, is available under the AGPL license. We also offer commercial licensing for customers that do not wish to comply with AGPL and want to keep their source code private.

## VISION

In a world in which speed and efficiency are paramount, we enable companies and people to build the most reliable solutions for document and data exchange, effortlessly.

## MISSION

It's our mission to be the most trusted and comprehensive technology provider which perfectly leverages the power of PDF, by offering open-source and enterprise solutions that streamline the generation and consumption of documents and data.

## CONTACT

marketing@itextpdf.com
**www.itextpdf.com**

## OUR OFFICES


### EUROPE, MIDDLE EAST, AFRICA & CIS

AA Tower
Technologiepark-Zwijnaarde 122
9052 Zwijnaarde
Belgium

sales.isb@itextpdf.com
Tel +32 9 298 02 31
Fax +32 9 270 33 75

### AMERICAS

265 Medford Street,
Suite 500
Somerville, MA 02143
United States

sales.isc@itextpdf.com
Tel +1 617 982 2646
Fax +1 617 982 2647

### ASIA & OCEANIA

Republic Plaza
9 Raffles Place, Level 6, Republic Plaza 1
SINGAPORE 048619
Singapore

sales.isa@itextpdf.com
Tel +65 6932 5062


**itextpdf.com**