January 8, 2026

RELEASE NOTES

This is a special iText PDF

As per usual, we have decided to create a PDF file that showcases a lot of what is possible with iText by Apryse.

This is no ordinary PDF, as it is PDF/UA-2 and PDF/A-4F compliant, created with the help of pdfHTML.

You can read more about it below.

Table of Contents

Release iText Core 9.5.0

> As always, we've made the Core release notes into a showcase PDF document. Not only does it conform to the latest PDF/UA-2 standard for accessible PDF documents, it's also digitally signed and has source code and resources required to recreate the document yourself embedded. Have fun!

Release date: Jan 8th 2026

In this first iText Core release of 2026 we're focusing on upcoming additions to the PDF specification; specifically adding support for Brotli-compressed PDF streams, and post-quantum-safe algorithms for digital signing.

The validation of PAdES signature profiles has been improved to detect Document Security Store changes between revisions.

In addition, there are various improvements and fixes across the core modules. See below for more details.

Support for Brotli Compressed PDF Streams

The Flate/Deflate compression method was first introduced in version 1.2 of the PDF specification back in 1996. Based on zlib, it has served well though is rather long in the tooth these days. With that in mind, the PDF Association plans to add support for the modern Brotli compression standard to the PDF specification in the near future. As befits our position in the industry, we're introducing Brotli support to iText now rather than later.

In this release we've added proof-of-concept support for reading and writing PDFs that use Brotli compression, including a new filter handler and corresponding high-level API hooks.

For now at least, you should consider this feature experimental, since current support for Brotli-compressed PDFs in other software is scarce. Even after it becomes part of the official specification it will take some time for the majority of PDF viewers and browsers to support such documents. As always with new standards and technology, however, it's important for iText along with other major PDF vendors to adopt and popularize them as soon as possible.

For those brave souls who wish to proceed, the new brotli-compressor module is only available from our artifactory, for now. See the README for more details on its usage and configuration.

Post-Quantum-Safe Digital Signature Algorithm Support

Once again, this is one for the future; albeit a rather more uncertain one. Quantum computers are beginning to step outside of the purely theoretical world, and with them comes the dangers of Q-Day. That is, the day when a quantum computer could conceivably instantly render existing means of encryption obsolete.

Fortunately, some extremely clever boffins have been aware of the dangers for some time, and a variety of encryption algorithms has been developed to counter quantum-based attacks. Thanks to the venerable Bouncy Castle cryptographic modules used by iText, we have implemented POC support for the Post-Quantum algorithms supported by Bouncy Castle.

Again, official support in the PDF specification will be coming soon and so it's time for us to implement support in iText. However, it should be noted that these are not yet supported in FIPS mode, as the official PQC-safe algorithms for FIPS are still to be defined. Once PQC-safe FIPS mode is possible, you can be sure iText will be at the forefront of PDF implementations.

PAdES Signature Profiles

We've made further headway into making digital signatures easier with preset PAdES signature profiles. The PAdES profile validation is improved and can detect when the Document Security Store (DSS) changes between revisions. It will emit a TimestampsAfterDSSEvent when appropriate, improving timestamp validation accuracy.

Also in this release you'll find new samples for the Ukraine and Moldova trusted lists which are available from the European Commission's eIDAS Dashboard site.

Pull Requests

For this release we want to thank dajoropo for their contribution to improve error handling when an attempt to create a PDF from a TIFF image fails. Now, iText Core will include the original exception to assist in diagnosing why the error occurred.

Bug Fixes and Miscellaneous

We have introduced a common, cross-platform JSON AST and converters to serialize/deserialize between Java/C# objects and a unified JSON representation, improving maintainability and GraalVM/AOT support.

Support for East Asian (Japanese) line-breaking rules in the layout module was added to avoid orphan punctuation at the beginning of lines, improving typography for Japanese text.

We also investigated and resolved an issue with form filling and flattening introduced after version 9.0.0, after a customer reported a regression in performance.

A bug in calculating the maximum number of XRef elements was fixed, avoiding potential overflow issues in large documents.

We fixed an issue preventing OCG layers from being added, modified, or removed in append mode was fixed, including when no prior OCG layers existed.

We also fixed the handling of unencrypted metadata in encrypted documents by aligning decryption logic with encryption dictionary flags (AES-256 vs AES-GCM behavior), and clarified how metadata is treated on creation.

A bug when validating a PDF signature was resolved, where iText did not use OCSP/CRL responses that were added to the document's DSS in a non-timestamped revision.

For Java, the robustness of structure tree handling after document merges was improved by detecting invalid /StructParent indices and logging appropriate warnings instead of throwing NullPointerExceptions.

Other Stuff

> If you use iText for digital signing, you may be interested in the Digital Signatures Hub which contains a ton of useful resources and examples.

Don't forget that in addition to the resources on our Knowledge Base, on our GitHub you can find a ton of useful up-to-date samples in the following repos:

Java

https://github.com/itext/itext-publications-examples-java
https://github.com/itext/itext-publications-book-java
https://github.com/itext/itext-publications-signing-examples-java
https://github.com/itext/itext-publications-signatures-java
https://github.com/itext/itext-publications-highlevel-java
https://github.com/itext/itext-publications-jumpstart-java

.NET

https://github.com/itext/itext-publications-samples-dotnet

> If you want to create ZUGFeRD/Factur-X-e-invoices with iText Core, we have both Java and .NET code samples available targeting the current ZUGFeRD/Factur-X specification. They demonstrate how to embed the XML invoice data and add the metadata required for conformance. Read this article to learn more about ZUGFeRD/Factur-X, and using these code samples to create EN 16931-compatible e-invoices.

Bear in mind that our `master` branch contains samples for the current stable release, while the default `develop` branch is for the bleeding edge commits towards the next release.

Also, don't forget to check out the release-related examples below, as well as the updated Core add-ons in the iText Suite we've released this time:

iText Suite 9.5 Releases

Release pdfCalligraph 5.0.5
Release pdfHTML 6.3.1
Release pdfOCR 4.1.2
Release pdfOptimizer 4.1.2
Release pdfSweep 5.0.5
Release pdfXFA 5.0.5

Downloads

	GitHub	Maven	NuGet	Artifactory
iText Core – 9.5.0 (Java)	link	link	N/A	link
iText Core – 9.5.0 (.NET)	link	N/A	link	link

Changelog

New features

- DEVSIX-9471 - Implement POC with Bouncy Castle for Post-Quantum algorithms
- DEVSIX-9578 – Brotli compression POC (reading/writing Brotli-compressed PDFs).
- DEVSIX-9587 – Brotli compression/decompression test coverage.
- DEVSIX-9594 – Update embedded Brotli decoder (Google upstream).
- DEVSIX-9608 – GraalVM support for Brotli compressor.
- DEVSIX-9531 - Samples for Ukraine and Moldova trusted lists
- DEVSIX-8978 – Japanese line-breaking rules.

Improvements

- DEVSIX-9567 – Correct handling of unencrypted metadata in encrypted PDFs.
- DEVSIX-9599 – Unified output-stream “finish” API and encryption handling.
- DEVSIX-9405 – Preserve original exceptions for TIFF image errors.
- DEVSIX-9641 – Prevent endless recursion in digital signature validation.
- DEVSIX-9616 – Extended PAdES integration test suite.
- DEVSIX-9610 – Improved DSS change detection and timestamp validation (PAdES).
- DEVSIX-9602 – Tests for modern/post-quantum signing algorithms.
- DEVSIX-9600 – Compare PAdES validation with eSigDSS.
- DEVSIX-9636 – Forms filling performance regression resolved.
- DEVSIX-9036 – Update VeraPDF model (PDF/UA validation).

Bug Fixes

- DEVSIX-9623 - CalculateMaxElementsInXref casting from long to int
- DEVSIX-9597 - OCG layers can't be added in append mode

- DEVSIX-9586 – Infinite layout loop in list + keep-together scenario.
- DEVSIX-9637 – NullReferenceException with nested tables and keep-together.
- DEVSIX-6486 – Stricter validation of transformation matrix length.
- DEVSIX-9623 – Fix casting when computing max XRef elements.
- DEVSIX-9179 – More robust structure trees after merges (avoid NPEs).
- DEVSIX-9525 - Use OCSP/CRL responses from the latest revision of the document

Deprecation of Older iText Versions

We're taking this opportunity to announce End of Life dates for deprecated iText versions. EOL for iText 7.1 was April 2025, and iText 7.2 was October 2025. As for iText 8.0, this will reach EOL in October 2026.

> EOL for these versions means they will transition to maintenance mode, and only receive security patches. However, just as with iText 5 we will continue to provide support for our customers. Even though iText 5 has been in maintenance mode since 2016, we regularly release new versions to address CVEs and other security-related bugfixes. See CVEs or the iText 5-specific CVEs page for more information.

Contributors

We'd like to shout out the following contributors for this release:

- Core team
  - Eugene Bochilo
  - Dmitry Chubrick
  - Angelina Pavlovets
  - Alexandr Pliushchou
  - Vitali Prudnikovich
  - Dmitry Radchuk
  - Andrei Stryhelski
  - Nanou Persoons
  - Glenn Volckaert
  - Alexandr Fedorov
  - Guust Ysebie
  - Yulian Gaponenko
  - Raf Hens
- Product / Marketing
  - André Lemos
  - Ian Morris
- Infrastructure / Devops
  - Marco Andries
  - Yauheni Borbut
- Research
  - Michaël Demey
  - Vlad Lipskiy
- Input from other teams
  - Rainer Plöckl
  - Alison Anderson
- Input from outside
  - Michael Klink
  - Matthias Valvekens

eBooks

Blockchain for PDF Documents
iText: Jump-Start Tutorial for .NET
iText: Jump-Start Tutorial for Java
iText: Converting HTML to PDF with pdfHTML
iText: Building Blocks
Best iText 7 Questions on StackOverflow

Installation Instructions

Installing iText for Java
Installing iText for .NET
Installing iText Community for Java developers
Installing iText Community for .NET developers

Examples (latest ones)

Refactored PDF Conformance Architecture
PAdES Signing High Level API
iText Core: Signature Appearance Improvements
iText 8.0.2 Delivers PDF/A-4 Support
High-level Annotation Flattening

FAQ (latest ones)

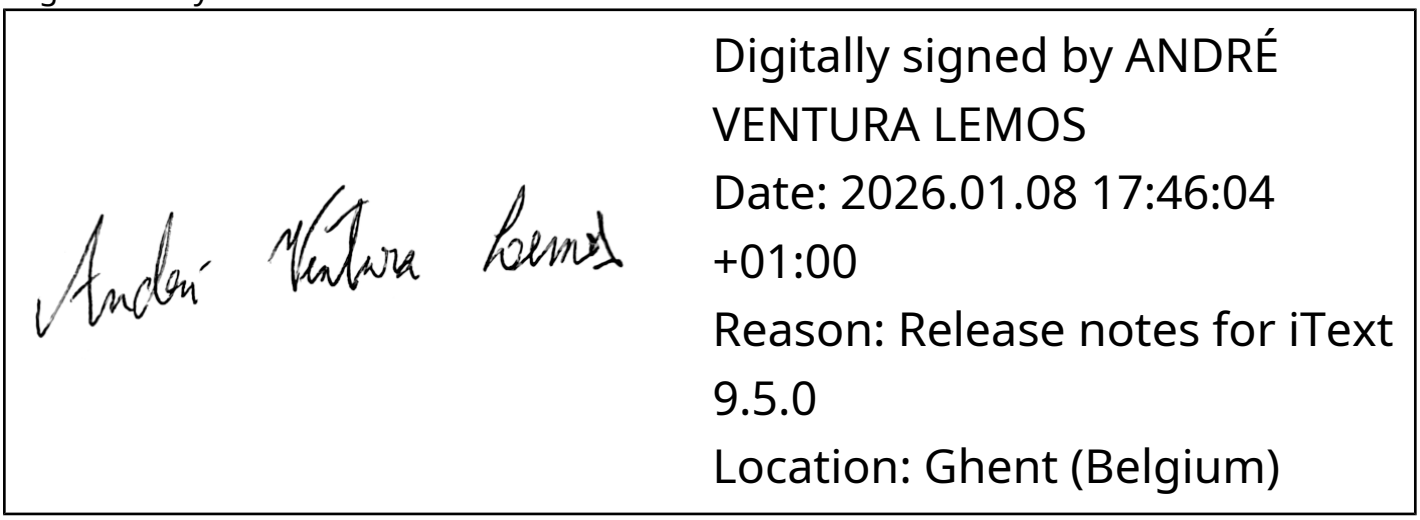
How to decrypt a PDF document with the owner password?
How to encrypt PDF using a certificate?
How to enable LTV for a timestamp signature?
Why isn't the Rupee symbol showing?
How do I install iText Core?

Features of this PDF

- Used pdfHTML to generate the PDF content (the HTML content is attached)
- It is both PDF/A-4f and PDF/UA-2 compliant (making it also a WTPDF)
- There is a MAC protected (AES 256 Encryted, SHA 256 Protected. Password is 'itext') version of this PDF attached (had to be separate, as PDF/A-4 does not allow it)
- Digitally signed using a Portuguese Identity Card (scroll below to check the code on how to validate it!)
- The main logo is generated using iText's custom SVG rendering engine
- Dynamically generated table of contents and bookmarks
- *Creatively* used Layers to toggle between Java and .NET code below
- Automatic Pagination by using our Events engine
- Fonts were *subsetted* for a smaller file size

To run the project (using .NET 8), just check the instructions on the attached file README.md.

Signed off by:



Generated by: Content: Ian Morris Source code: Guust Ysebie

Addendum: Verify the signature using iText (Java)

```java
package com.itextpdf.samples.sandbox.signatures.validation;

import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfReader;
import com.itextpdf.signatures.validation.SignatureValidator;
import com.itextpdf.signatures.validation.ValidatorChainBuilder;
import com.itextpdf.signatures.validation.lotl.LotlCountryCodeConstants;
import com.itextpdf.signatures.validation.lotl.LotlFetchingProperties;
import com.itextpdf.signatures.validation.lotl.LotlService;
import com.itextpdf.signatures.validation.lotl.QualifiedValidator;
import com.itextpdf.signatures.validation.lotl.RemoveOnFailingCountryData;
import com.itextpdf.signatures.validation.report.ValidationReport;

import java.io.IOException;
import java.util.Map;

public class  LotlSimpleSignatureValidation {

        public static final String SRC =  "./src/main/resources/pdfs"
            +  "/super_official_document_signed.pdf";

        public static void main(String[] args)  throws IOException {
            ValidatorChainBuilder builder =  new ValidatorChainBuilder();
            builder.trustEuropeanLotl(true);
            LotlFetchingProperties fetchingProperties =  new LotlFetchingProperties(
                                    new RemoveOnFailingCountryData());

             // Add other country codes if needed
            fetchingProperties.setCountryNames(LotlCountryCodeConstants.PORTUGAL);
            LotlService.initializeGlobalCache(fetchingProperties);

             // To perform Qualification validation, provide QualifiedValidator instance.
             // You can use this same instance to obtain the results after validation.
            QualifiedValidator qualifiedValidator =  new QualifiedValidator();
            builder.withQualifiedValidator(qualifiedValidator);

             try (PdfDocument document =  new PdfDocument(new PdfReader(SRC))) {
                SignatureValidator validator = builder.buildSignatureValidator(document);
                ValidationReport report = validator.validateSignatures();
                 // Separately, now you can obtain Qualification results
                Map<String, QualifiedValidator.QualificationValidationData> result =
                        qualifiedValidator.obtainAllSignaturesValidationResults();
            }  catch (IOException e) {
                 throw new RuntimeException(e);
            }
    }
}
```