


Java To a custom data provider [HOME](#)  [GUIDES](#)  [JAVA](#)  [SAVE A DOCUMENT](#)  [TO CUSTOM DATA PROVIDER](#)

# Save PDFs to custom data providers in Java



COPY PAGE



To save a `PdfDocument` to a custom data provider, use the `PdfDocument.saveAs()` method.

## Custom data provider

For full control over the data writing, extend the `WritableDataProvider` class to provide a custom data sink for saving documents. Here's an example of an in-memory data provider that uses a byte array for reading and writing data:

```
// A custom data provider that implements `WritableDataProvider`
// so data can be written to a custom data sink, `pdfData`.
class InMemoryDataProvider implements WritableDataProvider {
    private final List<Byte> pdfData = new ArrayList<>();

    @Override
    public long getSize() {
        return pdfData.size();
    }
}
```



ASK AI

```

@Nullable
@Override
public String getTitle() {
    return "Fake-Title";
}

@Override
public boolean canWrite() {
    // Allowing writing.
    return true;
}

@Override
public boolean startWrite(final WriteMode writeMode) {
    // Remove the current data if rewriting from scratch.
    // If `writeMode` is `WriteMode.APPEND` instead, keep the original content
    // so that write operations are appended to the end of the array. This is
    // incremental saving operations.
    if (writeMode == WriteMode.REWRITE) {
        pdfData.clear();
    }

    return true;
}

@Override
public boolean write(final byte[] bytes) {
    for (final byte value : bytes) {
        pdfData.add(value);
    }
    return true;
}

@Override
public boolean finishWrite() {
    // Nothing to do.
    return true;
}

@Override
public @Nullable byte[] read(final long size, final long offset) {
    final List<Byte> subset = new ArrayList<>(pdfData.subList((int) offset, (int) offset + (int) size));
    final byte[] bytes = new byte[(int) size];
    int i = 0;
    for (final Byte b : subset) {
        bytes[i] = b;
    }
}

```

```

        i++;
    }
    return bytes;
}

@Override
public boolean supportsAppending() {
    // You can append onto the string if you want to.
    return true;
}

@Override
public void release() {
    pdfData.clear();
}
}

```

You can save to the custom provider using `PdfDocument.saveAs()`:

```
document.saveAs(new CustomDataProvider(), new DocumentSaveOptions.Builder().build
```

## Document save options

Use `DocumentSaveOptions` for more control over a document's saving behavior. The following options are supported:

- ✧ `applyRedactionAnnotations` — Applies any redaction annotations present. A full save will be performed, as it isn't possible to apply redactions incrementally.
- ✧ `flattenAnnotations` — Flattens any annotations present. The `incremental` flag will be ignored when flattening.
- ✧ `forceSave` — Forces a save even if there were no modifications.
- ✧ `incremental` — Instead of rewriting a PDF, changes will be appended to it. This typically results in a significantly faster save if the document is large, as only the changes are added. It isn't possible to save incrementally when the `flattenAnnotations` or `applyRedactionsAnnotations` options are turned on.

## Example

`DocumentSaveOptions` can be created using the `save options` `Builder`. Here's an example showing how to flatten annotations when saving:

```
document.save(new DocumentSaveOptions.Builder().flattenAnnotations(true).build())
```

---

## Was this helpful?

✓ YES

✗ NO



Copyright 2025 Nutrient. All rights reserved.