



Web



Office to PDF



On this page



WEB > CONVERSION > OFFICE TO PDF > ONEDRIVE

Convert Office to PDF in Microsoft OneDrive



COPY PAGE



Nutrient Web SDK is a client-side JavaScript library that's fully compatible with Microsoft OneDrive for converting Office documents to PDF directly in the browser, without the need for server-side processing. To convert Office documents such as DOCX, XLSX, and PPTX to PDF, Nutrient Web SDK relies entirely on its own technology built from the ground up, and it doesn't depend on third-party tools such as LibreOffice or Microsoft Office. For more information on the supported Office formats, see the [list of supported file types](#).

TRY FOR FREE

LAUNCH DEMO

Converting Office documents to PDFs after displaying

To convert an Office document to a PDF after displaying it in the Nutrient viewer, follow the steps below.



ASK AI

- 1 Load the source Office document (optional). To load the document without a user interface visible to the user, use the `headless` parameter.
- 2 Make changes to the document (optional). For example, add annotations.
- 3 Convert the source document to a PDF with the `exportPDF` method (optional). Use the `outputFormat` flag to create a PDF/A document. For more information, see converting PDF to PDF/A.
- 4 Save the output document. The `exportPDF` method returns a `Promise` that resolves to an `ArrayBuffer` containing the output PDF document. You can use the resulting `ArrayBuffer` to download or persist the output PDF in storage. For more information on downloading or persisting the exported `ArrayBuffer`, see the guides on saving a document.

The following example loads an Office document and exports it to a PDF:

```
NutrientViewer.load({
  container: "#pspdfkit",
  document: "source.docx",
  licenseKey: "YOUR_LICENSE_KEY"
}).then((instance) => {
  instance.exportPDF();
});
```



The following example loads an Office document, exports it to a PDF with conformance level PDF/A-4f, and downloads it in the client's browser:

```
NutrientViewer.load({
  container: "#pspdfkit",
  document: "source.docx",
  licenseKey: "YOUR_LICENSE_KEY"
})
.then((instance) =>
  instance.exportPDF({
    outputFormat: {
      conformance: NutrientViewer.Conformance.PDFA_4F
    }
  })
)
```



```

.then(function (buffer) {
  const blob = new Blob([buffer], { type: "application/pdf" });
  const objectUrl = window.URL.createObjectURL(blob);
  downloadPdf(objectUrl);
  window.URL.revokeObjectURL(objectUrl);
});
function downloadPdf(blob) {
  const a = document.createElement("a");
  a.href = blob;
  a.style.display = "none";
  a.download = "output.pdf";
  a.setAttribute("download", "output.pdf");
  document.body.appendChild(a);
  a.click();
  document.body.removeChild(a);
}

```

When exporting a document, you have several options. Refer to our guides on [flattening annotations](#) and [incremental saving](#) for more details.

Auto saving can be configured for different scenarios and use cases. You can find more information in our [auto save](#) guide.

Converting Office documents with custom fonts to PDFs after displaying

When you convert an Office document with custom fonts to a PDF, Nutrient Web SDK might not have access to these fonts due to licensing constraints. In this case, Nutrient replaces unavailable fonts with their equivalents — like Arial with Noto — by default. To make sure the output PDF uses the same fonts as the original Office document, provide the path to the custom font files to Nutrient.

To convert an Office document with custom fonts to a PDF after displaying it in the Nutrient viewer, follow the steps below.

- 1 Load the custom fonts. For more information, see the guide on [adding custom fonts](#).
- 2 [Load the source Office document](#) (optional). To load the document without a user interface visible to the user, use the `headless` parameter.
- 3 Make changes to the document (optional). For example, [add annotations](#).

- 4 Convert the source document to a PDF with the `exportPDF` method (optional). Use the `outputFormat` flag to create a PDF/A document. For more information, see [converting PDF to PDF/A](#).
- 5 Save the output document. The `exportPDF` method returns a `Promise` that resolves to an `ArrayBuffer` containing the output PDF document. You can use the resulting `ArrayBuffer` to download or persist the output PDF in storage. For more information on downloading or persisting the exported `ArrayBuffer`, see the [guides on saving a document](#).

The following example loads an Office document and exports it to a PDF:

```
const fetchFont = (fontFileName) =>
  fetch(`https://example.com/${fontFileName}`).then((r) => {
    if (r.status === 200) {
      return r.blob();
    } else {
      throw new Error();
    }
  });

const customFonts = ["arial.ttf", "helvetica.ttf", "tahoma.ttf"].map(
  (font) => new NutrientViewer.Font({ name: font, callback: fetchFont })
);

NutrientViewer.load({
  customFonts,
  container: "#pspdfkit",
  document: "source.docx",
  licenseKey: "YOUR_LICENSE_KEY"
}).then((instance) => {
  instance.exportPDF();
});
```



When exporting a document, you have several options. Refer to our guides on [flattening annotations](#) and [incremental saving](#) for more details.

Auto saving can be configured for different scenarios and use cases. You can find more information in our [auto save](#) guide.

Converting Office documents to PDFs without displaying

To convert an Office document to a PDF without displaying it in the Nutrient viewer, follow the steps below.

- 1 Load and convert the source Office document using the `convertToPDF` method. This method takes the following parameters:
 - ⌘ A `Configuration` object that specifies the path to the source document and the license key.
 - ⌘ A member of the `Conformance` enumeration that specifies the conformance level of the output PDF document (optional). If you provide this parameter, the output is a PDF/A document.
- 2 Save the output document. The `convertToPDF` method returns a `Promise` that resolves to an `ArrayBuffer` containing the output PDF document. You can use the resulting `ArrayBuffer` to download or persist the output PDF in storage. For more information on downloading or persisting the exported `ArrayBuffer`, see the [guides on saving a document](#).

The following example exports the loaded document to a PDF with conformance level PDF/A-4f:

```
NutrientViewer.convertToPDF(  
  {  
    document: "source.docx",  
    licenseKey: "YOUR_LICENSE_KEY"  
  },  
  NutrientViewer.Conformance.PDFA_4F  
);
```



The following example converts an Office document to a PDF document and downloads it in the client's browser:

```
NutrientViewer.convertToPDF({  
  document: "source.docx",  
  licenseKey: "YOUR_LICENSE_KEY"
```



```

}).then(function (buffer) {
  const blob = new Blob([buffer], { type: "application/pdf" });
  const objectUrl = window.URL.createObjectURL(blob);
  downloadPdf(objectUrl);
  window.URL.revokeObjectURL(objectUrl);
});
function downloadPdf(blob) {
  const a = document.createElement("a");
  a.href = blob;
  a.style.display = "none";
  a.download = "output.pdf";
  a.setAttribute("download", "output.pdf");
  document.body.appendChild(a);
  a.click();
  document.body.removeChild(a);
}

```

Converting Office documents with custom fonts to PDFs without displaying

When you convert an Office document with custom fonts to a PDF, Nutrient Web SDK might not have access to these fonts due to licensing constraints. In this case, Nutrient replaces unavailable fonts with their equivalents — like Arial with Noto — by default. To make sure the output PDF uses the same fonts as the original Office document, provide the path to the custom font files to Nutrient.

To convert an Office document with custom fonts to a PDF without displaying it in the Nutrient viewer, follow the steps below.

- 1 Load the custom fonts. For more information, see [adding custom fonts](#).
- 2 Load and convert the source Office document using the `convertToPDF` method.

This method takes the following parameters:

- ⌘ A `Configuration` object that specifies the path to the source document, the license key, and the custom fonts used in the document.
- ⌘ A member of the `Conformance` enumeration that specifies the conformance level of the output PDF document (optional). If you provide this parameter, the output is a PDF/A document.

- 3 Save the output document. The `convertToPDF` method returns a `Promise` that resolves to an `ArrayBuffer` containing the output PDF document. You can use the resulting `ArrayBuffer` to download or persist the output PDF in storage. For more information on downloading or persisting the exported `ArrayBuffer`, see the [guides on saving a document](#).

The following example exports the loaded document to a PDF with conformance level PDF/A-2a:

```
const fetchFont = (fontFileName) =>
  fetch(`https://example.com/${fontFileName}`).then((r) => {
    if (r.status === 200) {
      return r.blob();
    } else {
      throw new Error();
    }
  });

const customFonts = ["arial.ttf", "helvetica.ttf", "tahoma.ttf"].map(
  (font) => new NutrientViewer.Font({ name: font, callback: fetchFont })
);

NutrientViewer.convertToPDF(
  {
    customFonts,
    document: "source.docx",
    licenseKey: "YOUR_LICENSE_KEY"
  },
  NutrientViewer.Conformance.PDFA_2A
);
```



Configuring conversion from Excel documents to PDF

You can configure the conversion of Excel documents by specifying limits for the maximum content width and height per sheet converted. They can be used to manage memory usage during spreadsheet conversions.

To configure these limits when converting Excel documents to PDF, set the `spreadsheetMaximumContentHeightPerSheet` and

`spreadsheetMaximumContentWidthPerSheet` properties.

- ✧ `spreadsheetMaximumContentHeightPerSheet` sets the maximum height (in millimeters) for spreadsheet content. The default value is `0` (unlimited).
- ✧ `spreadsheetMaximumContentWidthPerSheet` sets the maximum width (in millimeters) for spreadsheet content. The default value is `0` (unlimited).

Example using `NutrientViewer.convertToPDF()`

```
NutrientViewer.convertToPDF({
  document: "source.xlsx",
  licenseKey: "YOUR_LICENSE_KEY",
}, null, {
  spreadsheetMaximumContentHeightPerSheet: 2000,
  spreadsheetMaximumContentWidthPerSheet: 200
});
```



Example using `NutrientViewer.load()`

```
NutrientViewer.load({
  document: "source.xlsx",
  licenseKey: "YOUR_LICENSE_KEY",
  officeConversionSettings: {
    spreadsheetMaximumContentHeightPerSheet: 2000,
    spreadsheetMaximumContentWidthPerSheet: 200
  }
});
```



Configuring conversion from Word documents to PDF

You can configure the conversion of Word documents by specifying how the history of revisions should be handled.

- ✧ `documentMarkupMode` sets the markup mode for the document:

- ✧ `noMarkup` — Render the document as if all the changes were accepted. Comments are *not* converted to comment annotations. This is the default.
- ✧ `original` — Render the document as if all the changes were rejected (as if no changes/redlines were made to the document). Comments are *not* converted to comment annotations.
- ✧ `simpleMarkup` — Render the document as if all the changes were accepted. Comments are converted to comment annotations.
- ✧ `allMarkup` — Render the document with all markups. Redlines (suggestions) show as redlines (strikethrough with a red line, red font for changes). Comments are converted to comment annotations.

The default value is `noMarkup` .

Example using `NutrientViewer.convertToPDF()`

```
NutrientViewer.convertToPDF({  
  document: "source.docx",  
  licenseKey: "YOUR_LICENSE_KEY",  
}, null, {  
  documentMarkupMode: 'simpleMarkup',  
});
```



Example using `NutrientViewer.load()`

```
NutrientViewer.load({  
  document: "source.docx",  
  licenseKey: "YOUR_LICENSE_KEY",  
  officeConversionSettings: {  
    documentMarkupMode: 'simpleMarkup',  
  }  
});
```





When exporting a document, you have several options. Refer to our guides on [flattening annotations](#) and [incremental saving](#) for more details.

Auto saving can be configured for different scenarios and use cases. You can find more information in our [auto save](#) guide.

Was this helpful?

[About](#) [Contact](#) [Legal](#)

 YES

 NO
