



Web



To SharePoint Online



On this page



WEB > SAVE A DOCUMENT > TO SHAREPOINT ONLINE

Save PDFs to SharePoint Online

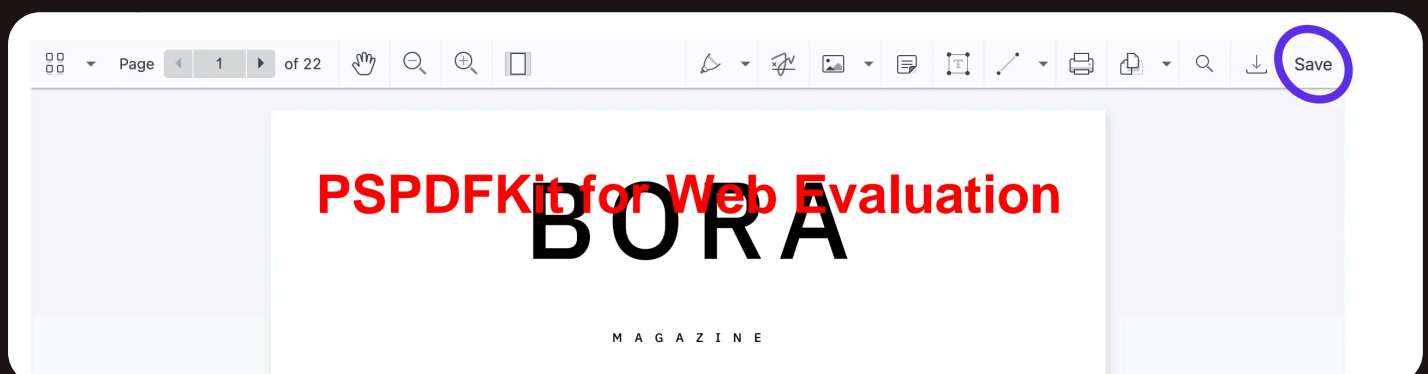


COPY PAGE



With Nutrient SharePoint SDK, PDF files opened using the SharePoint web part or file handler can be directly saved back to SharePoint.

Saving files back to SharePoint can be done programmatically or manually, which is initiated by the user when clicking the Save button in the user interface. The saving behavior can be customized as needed.



It's also possible for users to export a file by downloading it to their local storage. If desired, this export capability can be disabled (see our guide on how to prevent downloading or printing).



ASK AI

When opened in a web part

For saving file edits back to SharePoint Online from a web part, you can use the [@pnp/pnpjs](#) package, which allows you to easily interact with the SharePoint REST API.

You can find an example implementation in our [GitHub project](#).

Here's a snippet showing how to do it:

```
import { sp } from "@pnp/sp";
import "@pnp/sp/webs";
import "@pnp/sp/files";
import "@pnp/sp/folders";

// ...

const saveItem = {
  type: "custom",
  title: "Save",
  async onPress() {
    const fileContent = await instance.exportPDF();
    const file = sp.web.getFileByUrl(documentURL);

    await file.setContent(fileContent);
  }
};
```



`documentURL` is the same URL used to open the document in the SharePoint web part.

When opened with a file handler

Once you've configured the file handler endpoint ([see how to get started with the file handler](#)), it'll be able to parse the `items` property from the [activation parameters](#) array sent by SharePoint Online containing Graph URLs for the specified files. The Graph URL will be used to save changes back to SharePoint Online.

You can find a sample implementation of this saving functionality in our [GitHub project](#).

You can perform the saving entirely from the client side, although it can also be done from the server side using the same calls to the Microsoft Graph API. If saving from the server side, you can store the changes back to SharePoint Online with languages other than JavaScript.

The following snippet saves the document to SharePoint Online using client-side code (no server-side component is needed):



```
const saveItem = {
  type: "custom",
  title: "Save",
  onPress() {
    async function exportAndSave() {
      const content = await instance.exportPDF();
      return uploadContentToSharePoint(content, token, graphUrl);
    }
  }
};

async function uploadContentToSharePoint(content, token, graphUrl) {
  const itemInfoResponse = await fetch(graphUrl, {
    headers: {
      authorization: `Bearer ${token}`
    }
  });

  if (!itemInfoResponse.ok) {
    throw new Error("Error getting item details before save.");
  }

  const itemInfo = await itemInfoResponse.json();

  // Construct a Graph URL to `PUT` changes.
  const itemUrl = `https://graph.microsoft.com/v1.0/drives/${itemInfo.parentReference.id}/items/${itemInfo.id}`;

  const totalSize = content.byteLength;
  // The Graph API considers files larger than 5 MB "large."
  // We need to use the upload session API for these cases:
  // https://docs.microsoft.com/en-us/graph/api/driveitem-createuploadsession.
  const isLargeFile = totalSize >= 5 * 1024 * 1024;

  if (isLargeFile) {
    await uploadLargeFileToSharePoint(content, token, itemUrl);
  }
}
```

```

    } else {
      await uploadSmallFileToSharePoint(content, token, itemUrl);
    }
  }

  async function uploadSmallFileToSharePoint(content, token, itemUrl) {
    const contentUrl = `${itemUrl}/content`;

    // Update the file via the Graph API.
    const updateResult = await fetch(contentUrl, {
      body: content,
      headers: {
        authorization: `Bearer ${token}`
      },
      method: "PUT"
    });

    if (!updateResult.ok) {
      const err = await updateResult.text();
      throw new Error(err);
    }
  }

  async function uploadLargeFileToSharePoint(content, token, itemUrl) {
    const uploadSessionUrl = `${itemUrl}/createUploadSession`;

    const uploadSessionResponse = await fetch(uploadSessionUrl, {
      method: "POST",
      headers: {
        authorization: `Bearer ${token}`,
        "Content-Type": "application/json"
      },
      body: JSON.stringify({
        item: {
          "@microsoft.graph.conflictBehavior": "replace"
        }
      })
    });

    if (!uploadSessionResponse.ok) {
      throw new Error(uploadSessionResponse.statusText);
    }

    const { uploadUrl } = await uploadSessionResponse.json();

    // Splitting the file into chunks of 10 MB.

```

```
const CHUNK_SIZE = 10 * 1024 * 1024;
const totalSize = content.byteLength;
let currentOffset = 0;

while (currentOffset < totalSize) {
  const chunkSize = Math.min(CHUNK_SIZE, totalSize - currentOffset);
  const chunk = content.slice(
    currentOffset,
    currentOffset + chunkSize
  );

  const uploadResponse = await fetch(uploadUrl, {
    method: "PUT",
    headers: {
      authorization: `Bearer ${token}`,
      "Content-Length": chunk.byteLength.toString(),
      "Content-Range": `bytes ${currentOffset}-${
        currentOffset + chunkSize - 1
      }/${totalSize}`
    },
    body: chunk
  });

  if (!uploadResponse.ok) {
    throw new Error(uploadResponse.statusText);
  }

  currentOffset += chunkSize;
}
}
```

Was this helpful?

✓ YES

✗ NO