



Web



To SharePoint on-premises



On this page



WEB > SAVE A DOCUMENT > TO SHAREPOINT ON PREMISES

Save PDFs to SharePoint on-premises

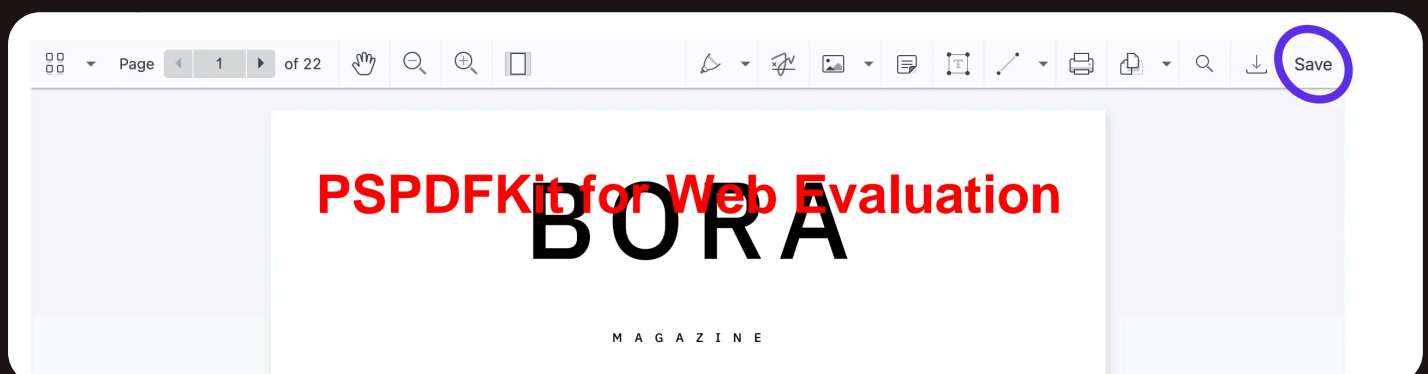


COPY PAGE



With Nutrient SharePoint SDK, files that are opened using the SharePoint file handler can be directly saved back to your SharePoint On-Premises environment.

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

Adding a save button

The following example demonstrates how to add a Save button to the user interface, which will directly save the file back to the SharePoint environment.

It assumes the current user has sufficient rights to the edited PDF file. It doesn't include error handling, which we recommend you implement both on the client side and on the server side to cover connection errors, validating user rights, checking whether files need to be checked out before saving, other internal errors, etc. We recommend you customize this example with your specific error handling requirements.

We also recommend you familiarize yourself with information on [user rights management](#), [metadata](#), and [checking out files](#) with SharePoint On-Premises.

To add the Save button, update the script inside the main content placeholder as follows:

```
<script type="text/javascript">
  // !!! Enter your license key here !!!
  let licenseKey = "paste your license here";

  // Load PDF editor.
  NutrientViewer.load({
    container: "#pspdfkit",
    document: "<%= fileURL %>",
    licenseKey: licenseKey
  })
  .then(function (instance) {
    // Add Save button.
    instance.setToolbarItems((items) => {
      items.push({
        type: "custom",
        id: "save-button",
        title: "Save",
        onPress: () => {
          savePDF(instance);
        }
      });
      return items;
    });
    console.log("PSPDFKit loaded", instance);
  })
  .catch(function (error) {
```



```

        console.error(error.message);
    });

// This method exports the file from the editor and saves it into SharePoint.
async function savePDF(instance) {
    // Export modified document.
    const arrayBuffer = await instance.exportPDF();
    // Create blob for upload.
    const blob = new Blob([arrayBuffer], { type: "application/pdf" });
    // Retrieve existing form data.
    const formData = new FormData(
        document.getElementById("<%%= Form.ClientID %>")
    );
    // Add file URL and modified file content.
    formData.append("fileURL", "<%%= fileURL %>");
    formData.append("fileContent", blob);
    // Post back the form data.
    await fetch(window.location, {
        method: "POST",
        body: formData
    })
    .then((response) => {
        // Handle response.
        console.log(response);
        if (response.ok) {
            alert("File saved.");
        } else {
            alert(
                "Sorry, something went wrong." +
                "\nStatus: " +
                response.status +
                "\nMessage: " +
                response.statusText +
                "\n\nSee console log for more details."
            );
        }
    })
    .catch((error) => {
        // Handle errors.
        alert(
            "Sorry, something went wrong.\n\nSee console log for more details."
        );
        console.log(error);
    });
}
</script>

```

Then, modify the `ViewPDF.aspx.cs` code behind the file to include the `fileURL` public property and some new code in the `Page_load` event:



```
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;
using System;

namespace SavePDFSharePoint.Layouts.SavePDFSharePoint
{
    public partial class ViewPDF : LayoutsPageBase
    {
        /// <summary>
        /// Gets the file URL extracted from the URL key.
        /// </summary>
        public string fileURL { get; private set; }

        /// <summary>
        /// Page load event.
        /// </summary>
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                // Extract the file URL for the SharePoint file.
                fileURL = Request.QueryString["fileURL"];
            }
            else
            {
                // Update the file specified by the `fileURL` field with the data received.
                Web.Files.Add(
                    Request["fileURL"],
                    Request.Files["fileContent"].InputStream,
                    true);
            }
        }
    }
}
```

Assuming no error occurs and the current user has sufficient rights to the edited PDF file, the user will save edits directly back to the on-premises environment.

For an out-of-the-box PDF viewing and editing solution that includes user rights management, checking out files, and error handling, check out [Document Editor](#). It's powered by Nutrient technology, and it's designed as a drop-in solution that requires minimal configuration.

Was this helpful?

☒ YES

☐ NO

Copyright 2025 Nutrient. All rights reserved.