

[DOCS](#)[CONTACT SALES](#)[UWP](#)[Get started](#)[GETTING STARTED](#)[UWP](#)

Getting started on UWP

[EXISTING PROJECT](#)

⋮

Nutrient UWP SDK (formerly PSPDFKit for Windows) is being sunset, in line with Microsoft's decision to gradually phase out UWP by limiting updates to bug fixes and security patches. All Nutrient UWP licenses will continue to work and be supported until 1 December 2026, or until your license expires, whichever occurs later. Interested in transitioning to another Nutrient product? [Contact us](#) and we'll be happy to help.

Requirements

Windows 10 or Windows 11

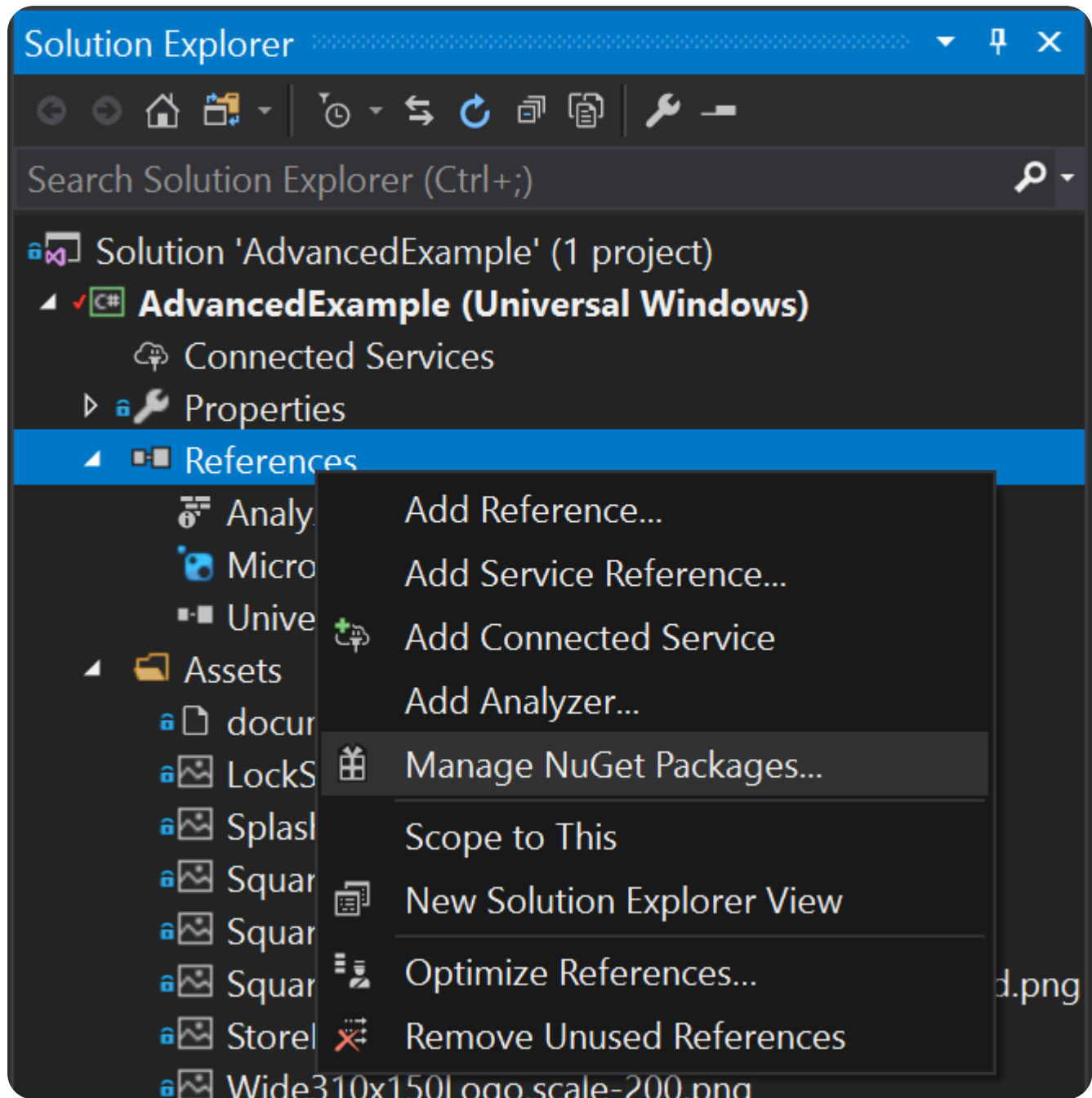
Visual Studio 2019 or above

A UWP-based application

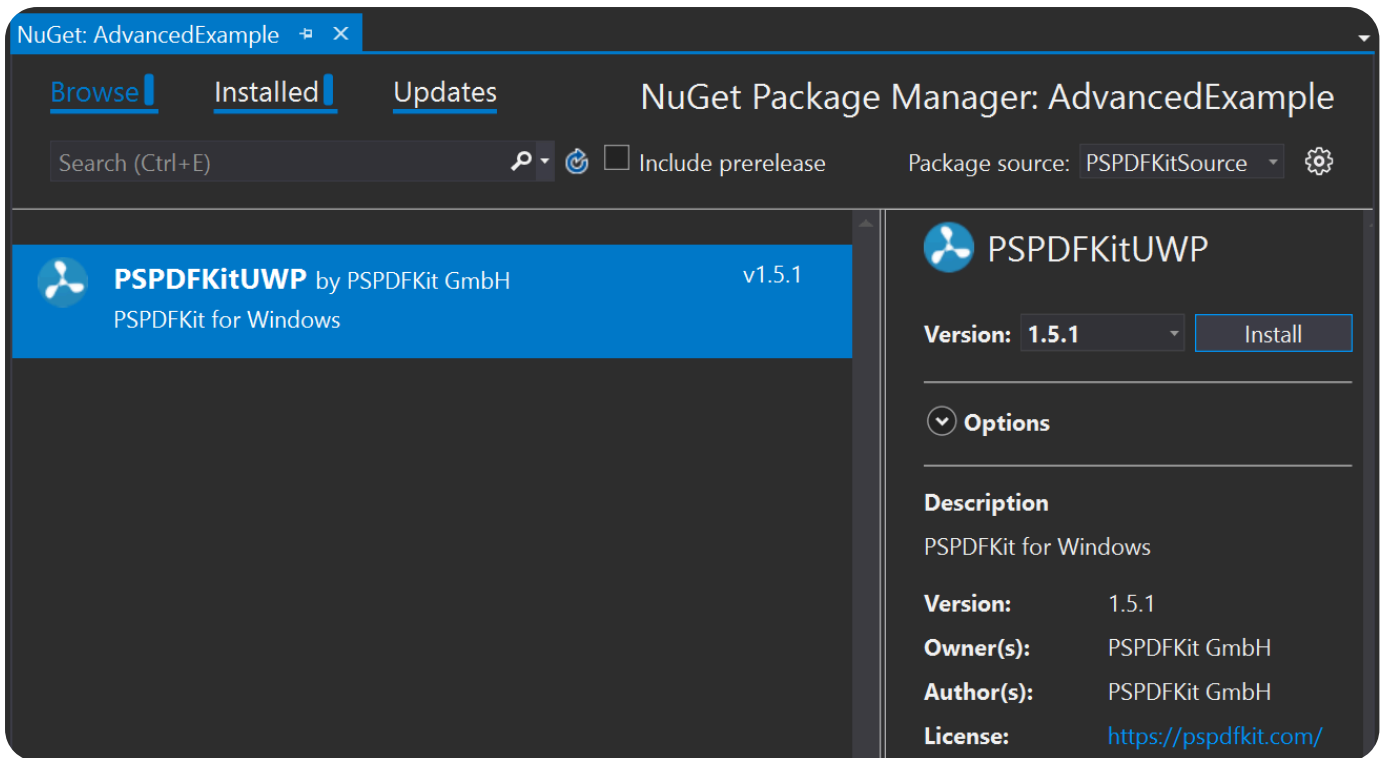
1 Adding Nutrient to your project

[ASK AI](#)

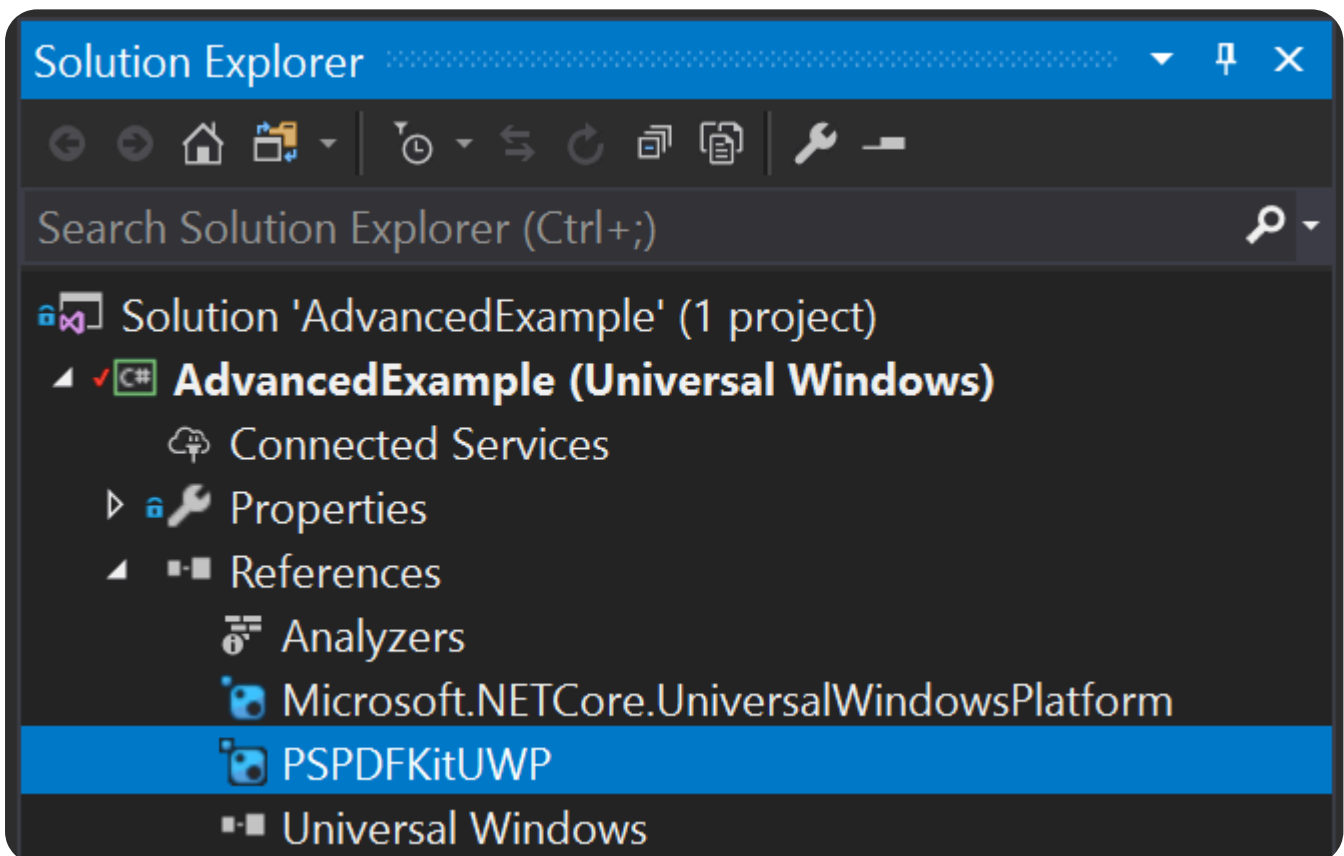
- 1 Open your app's solution, and in the Solution Explorer, right-click **References** and click the menu item **Manage NuGet Packages....** This will open the NuGet Package Manager for your solution.



- 2 Search for `pspdfkituwp`, and you'll find the package on [nuget.org](https://www.nuget.org).
- 3 On the right side, in the panel describing the package, click on the Install button to install the package.



- 4 Once that's complete, you'll see a reference to the package in the Solution Explorer under **References**.



2 Displaying a PDF

- 1 Open `MainPage.xaml` and paste the following XML as a child of the `<Grid>` element:

```

1  <Grid.RowDefinitions>
2    <RowDefinition Height="*" />
3    <RowDefinition Height="52" />
4  </Grid.RowDefinitions>
5  <ui:PdfView Grid.Row="0" Name="PdfView" />
6  <Button IsEnabled="False" Content="Open PDF" HorizontalAlignment="Left" Ma

```

- 2 You'll see a green line under `ui:PdfView` . If you hover over it, you'll be given the option for potential fixes. Apply the suggested fix, which will add the following namespace to `<Page>` :

```

1  <Page
2    xmlns:ui="using:PSPDFKit.UI"
3  />

```

- 3 If you've purchased a license and assigned it to a bundle ID, you can add it here:

```
<ui:PdfView Grid.Row="0" Name="PdfView" License="..." />
```

If not, omitting the `License` property will initialize the `PdfView` in trial mode.

- 4 Open the `MainPage.xaml.cs` file, add a delegate handler to `PdfView.InitializationCompletedHandler` in the constructor, and add the button's click handler function:

```

1  using PSPDFKit.Document;
2  using PSPDFKit.Pdf;
3  using PSPDFKit.UI;
4
5  public MainPage()
6  {
7      InitializeComponent();
8
9      // This handler is invoked once the `PdfView` has been initialized,
10     PdfView.InitializationCompletedHandler += delegate(PdfView sender, D
11     {
12         // Now that the `PdfView` is ready, enable the button for opening
13         Button_OpenPDF.IsEnabled = true;
14     };

```

```
15 }
16
17 private async void Button_OpenPDF_Click(object sender, RoutedEventArgs
18 {
19     var picker = new Windows.Storage.Pickers.FileOpenPicker();
20     picker.ViewMode = Windows.Storage.Pickers.PickerViewMode.Thumbnail;
21     picker.SuggestedStartLocation = Windows.Storage.Pickers.PickerLocati
22     picker.FileTypeFilter.Add(".pdf");
23
24     var file = await picker.PickSingleFileAsync();
25     if (file != null)
26     {
27         var document = DocumentSource.CreateFromStorageFile(file);
28         await PdfView.Controller.ShowDocumentAsync(document);
29     }
30 }
```

Note that you must wait for initialization of the `PdfView` to be complete before using `PdfView.Controller`.

- 5 In the Build toolbar, choose Debug and `x86` or `x64` — whichever you prefer.
- 6 Then, in the menu, select Build > Build Solution.
- 7 Start the application, and you should be able to open a PDF by clicking the Open button.

Next steps

React Native for Windows UWP support

Nutrient UWP SDK comes with optional support for React Native. This makes it easy to add PDF support to your React Native Windows app. See our open source GitHub repository for details on how to integrate.

Support for .NET, WPF (Windows Presentation Foundation), and Xamarin

There's ongoing work by Microsoft to allow integrating UWP controls in WPF apps. We also have a Xamarin library for UWP, and you can check out our [Xamarin guide](#) for more information.

Support for Windows 7/Windows 8.1

For companies that still need to support legacy Windows, we offer CEF as a replacement. This has a slightly higher performance impact (around 20–30 percent slower), but otherwise, it's functionally equivalent; it just needs more direct interaction with JavaScript.

Mainstream support for Windows 7 ended on 13 January 2015, and mainstream support for Windows 8.1 ended on 9 January 2018. Microsoft offered paid extended support until January 2020/2023, respectively.

Was this helpful?

YES

NO

Questions? [Contact us](#)