



UWP



Save



WINDOWS > GUIDES > ANNOTATIONS > SAVE

Saving annotations to an external source in UWP

Using export and import functionality provided by Nutrient UWP SDK, you can export any changes made to annotations to an external source of your choice, and then import them when you open the document again to restore the changes.

This approach allows you to save on bandwidth and space, as you won't need to export the entire document whenever there are any modifications. Rather, you only send the annotation information whenever necessary, retaining the same base file.

To enable this type of setup, you can use either [Instant JSON](#) or [XFDF](#) as the export format, depending on your needs. Both of these can be converted into strings for easy storage on whichever external source you choose. This guide will cover files. If you want to use stringified data directly, refer to our [saving to a database guide](#).

Exporting and importing XFDF from a file

In this example, you'll save the annotation data to a file with the `xfdf` extension. This file can later be used to import the XFDF data back.

There are a number of different possibilities when creating and writing to files in UWP. Here, you'll create a file in the application's `LocalFolder`:

```
var xfdfFile = await ApplicationData.Current.LocalFolder.CreateFileAsync
```



ASK AI

Then, open a `StorageStreamTransaction` and pass the stream to a `DataWriter`. The `DataWriter` is used by Nutrient's `Document` to write into whichever stream type you chose to use, giving you a great deal of freedom if your needs go against simple files, or if you'd rather create and write into your file in some other way:

```
1 using (var dataWriter = new DataWriter(await xfdfFile.OpenAsync(FileAccess: FileAccessWrite))
2 {
3     await PDFView.Document.ExportXfdfToDataWriterAsync(dataWriter);
4     await dataWriter.StoreAsync();
5 }
```

Note that, in this case, you must call both `StoreAsync` and `CommitAsync` for changes to be saved. This could change, depending on your use case. The important thing is that the stream is committed with the exported XFDF data.

To import, call `Document.ImportXfdfFileAsync`, passing in the relevant `.xfdf` file. In this example, you'd do this like so:

```
1 var importedXfdfFile = await StorageFile.GetFileFromApplicationUriAsync(new Uri("file://..."));
2
3 // or
4
5 var importedXfdfFile = await ApplicationData.Current.LocalFolder.GetFilesAsync().FirstOrDefault(f => f.Name.EndsWith(".xfdf"));
```

And finally, call the following:

```
await PDFView.Document.ImportXfdfFileAsync(importedXfdfFile);
```

Exporting and importing using Instant JSON

For Instant JSON, our SDK outputs a `JsonObject`. The storage of this data into a file is handled by the user however they deem fit. In this example, you'll convert the `JsonObject` into a `string` and add it to a `json` file, similar to how it's done in the section above. You'll use the

`Document.ExportInstantJsonAsync` method:

```

1  var jsonFile = await ApplicationData.Current.LocalFolder.CreateFileAsync('
2
3  using (var dataWriter = new DataWriter(await jsonFile.OpenAsync(FileAccessMode
4  {
5      var instantJsonObject = await PDFView.Document.ExportInstantJsonAsync();
6      dataWriter.WriteString(instantJsonObject.Stringify());

```

Then, you can import it like so:

```

1  var file = await ApplicationData.Current.LocalFolder.GetFilesAsync("annotat
2  await PDFView.Document.ImportInstantJsonAsync(JsonObject.Parse(await FileIO.Re

```

For more information on Instant JSON and its properties and uses, refer to our [feature-specific guide](#).

Was this helpful?

☒ YES

☐ NO

Questions? [Contact us](#)

