



UWP



Save a document



WINDOWS > GUIDES > SAVE A DOCUMENT

Save PDFs to a custom data provider in UWP

Nutrient UWP SDK includes support for saving `DocumentSource`s to data providers.

A data provider defines a common interface for Nutrient to read and write PDF documents from custom sources. This is especially helpful if you want to support your own encryption or compression scheme.

Existing data providers

Nutrient UWP SDK ships with two predefined providers:

`StorageFileDataProvider``RandomAccessStreamDataProvider`

You can also write your own custom data providers by implementing the `IDataProvider` interface. The [Catalog project](#) contains another two examples of custom providers:

`ExampleDataProvider``AesDataProvider`

Custom data provider

Creating your own custom data provider is a straightforward process. You'll have to create a class that implements the `IDataProvider` interface and all its methods. These methods should help you structure your data in a way that's compatible with Nutrient UWP SDK.



ASK AI

Writing

Writing is done separately from reading, and it uses data sinks. These are classes specifically for writing data. They represent a destination to write to and allow you to export documents directly to streams or other desired mediums.

Nutrient UWP SDK ships with two default data sinks, which support `DataWriter`s and `IRandomAccessStream`s. Similar to data providers, you can also implement your own data sinks through `IDataSink`.

The `WriteDataAsync` method allows you to control this process:

```
1 public IRandomAccessStream Stream { get; set; }
2
3 private async Task<bool> WriteDataAsync(IBuffer data)
4 {
5     using (var outputStream = Stream.GetOutputStreamAt(Stream.Size))
6     {
7         await outputStream.WriteAsync(data);
8     }
9
10    return true;
11 }
```

Depending on the medium being used and your settings, the `DataSinkOption` class tells your data sink whether to start at the beginning of the stream or to append to it. For example, when incremental saves or annotation flattening are enabled, `IDataSink` must be empty, as the entire document is written out and appending is not supported.

Finally, documents can be exported to your `IDataSink` with the `Document.ExportToDataSinkAsync` method:

```
await PDFView.Document.ExportToDataSinkAsync(new ExampleDataSink(option) { St
```

Our [Catalog](#) application includes two examples of `IDataSink` implementations.

Was this helpful?

✓ YES

✗ NO

Questions? [Contact us](#)

