



UWP



Open a document



WINDOWS &gt; GUIDES &gt; OPEN A DOCUMENT

# Open PDFs from a custom data provider in UWP

In addition to loading documents directly using `StorageFile`, `IBuffer`, or a `Uri`, Nutrient UWP SDK includes support for creating `DocumentSource`s from 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`

These providers can be used to create a `DocumentSource`, which is then used by the `Controller` to display the document:

```
1 _fileStream = await file.OpenAsync(FileAccessMode.ReadWrite);  
2
```



ASK AI

```
3 var dataProvider = new ExampleDataProvider(_fileStream);
4 var documentSource = DocumentSource.CreateFromDataProvider(dataProvider);
5
6 await PDFView.Controller.ShowDocumentAsync(documentSource);
```

## Custom data provider

To create your own custom data provider, 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.

### Reading

For reading, the `ReadAsync` method is used internally when needed. The content of this method will vary depending on how documents are obtained in your own application. Within the `ReadAsync` method, your chosen data structures should be adapted to match the `IBuffer` return expected by Nutrient UWP SDK, like so:

```
1 private readonly IRandomAccessStream _data;
2
3 private async Task<IBuffer> ReadAsync(uint size, uint offset)
4 {
5     if (offset >= _data.Size)
6     {
7         throw new Exception("Attempt to read beyond the end of a random access
8     }
9
10    var copy = new Buffer(size);
11    _data.Seek(offset);
12    return await _data.ReadAsync(copy, size, InputStreamOptions.ReadAhead);
13 }
```

Note that your method must ensure the size and offset used for the reading operation are valid according to your data.

For extra examples, check out our [Catalog](#) application, which includes two sample `IDataSink` implementations. For writing to a data sink, refer to our [save a document to a custom data provider](#) guide.

---

Was this helpful?

✓ YES

✗ NO

Questions? [Contact us](#)

