



UWP



Search



WINDOWS > GUIDES > SEARCH

Indexed full-text PDF search in UWP

Nutrient supports fast and efficient full-text search in PDF documents through

`PSPDFKit.Search.Library`. This document describes how to get started.

Getting started

To start indexing, create a `Library` and give it a name. You can then add folders that contain PDF files to this named library. The `Library` will index all the PDFs in those folders.

Here's a simple example of how to create or open a library and start indexing PDFs in a directory:

```
1 // Opening a library creates one if it doesn't already exist.
2 var library = await Library.OpenLibraryAsync("MyLibrary");
3
4 // Find a folder containing PDFs.
5 var folderPicker = new Windows.Storage.Pickers.FolderPicker();
6 folderPicker.SuggestedStartLocation = Windows.Storage.Pickers.PickerLocationId
7 folderPicker.FileTypeFilter.Add("*");
8
9 Windows.Storage.StorageFolder folder = await folderPicker.PickSingleFolderAsync();
10 if (folder != null)
11 {
12     // Queue up the PDFs in the folder for indexing.
13     library.EnqueueDocumentsInFolderAsync(folder);
14 }
```



The documents will now be indexed in the background.



ASK AI

Alternatively, you can enqueue a `List` of `IDataProvider` objects with the `EnqueueDocumentsFromProviderAsync` method.

Then, you can choose to start querying documents right away or wait until all documents added to the indexer queue have been completed.

Here's an example of how to wait and then get the list of indexed documents:

```
1 // Wait for indexing to finish.
2 await library.WaitForAllIndexingTasksToFinishAsync();
3
4 // Get the list of indexed documents.
5 var documentUIDs = await library.GetIndexedUidsAsync();
```

Identifying documents

The documents in the list returned by `GetIndexedUidsAsync` are represented by a unique ID (UID). When using `StorageFile` s, this UID is a `string` comprised of a future access token identifying the folder containing the PDF and the file name of the PDF within that folder. For `IDataProvider` s, the indexed UID is a `string` simply containing the `IDataProvider` UID.

Due to the unique restrictions of UWP, when using `StorageFile` s, it's essential that you don't clear the application's `FutureAccessList` if you wish to retain your libraries, as this is the only place for the future access token to be recorded.

Moreover, when using `DataProvider` s, neither the streams nor providers themselves are tracked internally, and they need to be managed by your own application.

You can create a `PSPDFKit.Document.DocumentSource` object for a given document UID using either `DocumentSource.CreateFromStorageFileUidAsync` or `DocumentSource.CreateFromDataProvider`, both of which are static methods.

A `StorageFile` object for the file can be accessed by calling `GetFile` on the created `DocumentSource` object. Note that the method will throw an exception if the document referred to can no longer be located.

Here's an example:

```
1 // Get the list of indexed documents.
2 var documentUIDs = await library.GetIndexedUidsAsync();
3
```

```

4  foreach (var uid in documentUIDs)
5  {
6      try
7      {
8          var documentSource = await DocumentSource.CreateFromUidAsync(uid);
9          StorageFile file = documentSource.GetFile();
10     }
11     catch (Exception e)
12     {
13         // Examine the exception.
14     }
15 }

```

Both the `StorageProvider` and `DataProvider` implementations can be used side by side.

`StorageProvider` UUIDs contain the file name, while `DataProvider` ones are merely numeric, you're able to easily check when needed. Note the need to maintain a list of all relevant providers:

```

1  if (uid.EndsWith(".pdf"))
2  {
3      document = await DocumentSource.CreateFromStorageFileUidAsync(uid);
4  }
5  else
6  {
7      document = DocumentSource.CreateFromDataProvider(_providers.Find(provider :
8  }

```

Index and document status

`Library` allows you to query for the current indexing state.

You can decide to only query the library if all queued documents have been indexed by using

`IsIndexingAsync()`. You may also check the current status of individual documents by using

`GetIndexDocumentStatusAsync()`.

Querying the library

To query the library, use the `SearchAsync` method, supplying it with a `LibraryQuery` object.

Here's an example:

```
1 // Search all documents in the library for the text "Acme."
```



The results of the query are sent to a query result handler, which you must provide to the library.

Here's an example:

```
library.OnSearchComplete += MyOnSearchCompleteMethod;
```



The `OnSearchComplete` event handler receives a reference to the originating library, along with a dictionary mapping a document UID to a `LibraryQueryResult` object. Each result object also contains the UID as a property and a list of the page indexes where matching results were found.

If you wish to show preview snippets, you should set the `GenerateTextPreviews` property in the query object to `true`. Then, preview text snippets will be delivered to you via the `OnSearchPreviewComplete` event handler.

Here's an example:

```
1 library.OnSearchPreviewComplete += MyOnSearchPreviewCompleteMethod;
2
3 var query = new LibraryQuery("Acme")
4 {
5     GenerateTextPreviews = true
6 }
7 var succeeded = await library.SearchAsync(query);
```



The `OnSearchPreviewComplete` event handler receives a reference to the originating library, along with a list of `LibraryPreviewResult` objects — one for each match. Each of these objects contains a UID identifying the document, a page index where the matching text is located, a snippet of text surrounding the match, the range of the matched text within the preview snippet, and the page text. Each object also has an annotation ID indicating whether or not the match was found in an annotation.


Advanced matching options

`Library` offers advanced matching options. You can set these options in a `LibraryQuery` object.

Password-protected documents

When indexing documents, it's possible you might come across a password-protected document.

You can unlock a password-protected document with an event handler, which is fired every time a password is required. The following example shows how this is possible:



```
1 private Library _library;
2
3 internal async void Initialize(PdfView pdfView)
4 {
5     _library = await Library.OpenLibraryAsync("catalog");
6     _library.OnPasswordRequested += Library_OnPasswordRequested;
7 }
8
9 private void Library_OnPasswordRequested(Library sender, PasswordRequest passwordRequest)
10 {
11     if (passwordRequest.Uid.Contains("Password.pdf"))
12     {
13         passwordRequest.Password = "test123";
14         break;
15     }
16
17     passwordRequest.Deferral.Complete();
18 }
```

`PasswordRequest` will always have the UID populated with the path being indexed (note the full path will be assigned with the [future access token](#) and the file name). Check against this string to determine which document requires a password and populate the `Password` member of `PasswordRequest` to unlock the document. Ensure the `Deferral` is completed, as per the last line of `Library_OnPasswordRequested`; otherwise, the index will fail and throw an exception.

Example code

You'll find a complete working code example in the Catalog app provided with the SDK.

Was this helpful?

✓ YES

✗ NO

