Hyland Document Filters 26.4

Implementation Guide

November 2026

© Hyland Software, Inc. All rights reserved.

Table of contents

1. Getting started 11
1.1 Getting started with Document Filters 11
1.2 Getting Started with .NET 15
1.3 Getting Started with C 19
1.4 Getting Started with C++ 27
1.5 Getting Stated with COM 36
1.6 Getting Stated with Java 0
1.7 Getting Stated with Python 0
1.8 About accessibility info extraction 0
1.9 About conversion profiles 0
1.10 About custom streams and extended streams 0
1.11 About File Type Identification 0
1.12 About Fonts 0
1.13 About Multi-Part Archives 0
1.14 About Multithreading 0
1.15 About Optical Character Recognition (OCR) 0
1.16 Content Enrichment 0
1.17 Output Formats 0
2. Tutorials 0
2.1 Tutorials 0
2.2 Opening Files 0
2.2.1 How do I open a document from disk? 0
2.2.2 How do I open a document from memory? 0
2.3 Converting Files 0
2.3.1 How do I convert a document to a PDF file? 0
2.3.2 How do I convert a document to Classic HTML? 0
2.3.3 How do I convert a document to JSON in Hi-Def mode? 0
2.3.4 How do I convert a document to Markdown in Hi-Def mode? 0

© Hyland Software, Inc. All rights reserved.

2.3.5 How do I convert a document to Markdown in Text mode? 0
2.3.6 How do I convert a document to paginated HiDef HTML? 0
2.3.7 How do I convert a document to PNG images? 0
2.3.8 How do I convert a document to Structured XML? 0
2.3.9 How do I convert a PDF to Markdown with Table Detection? 0
2.3.10 How do I convert a scanned image to Markdown with Image Layout Detection? 0
2.3.11 How do I classify and describe a page with machine learning? 0
2.3.12 How do I convert HTML to Hi-Def? 0
2.4 Extracting Files 0
2.4.1 How do I extract metadata from a document? 0
2.4.2 How do I extract sub-documents from documents and archives? 0
2.4.3 How do I extract text and metadata from a document? 0
2.4.4 How do I handle password-protected files? 0
2.5 Other 0
2.5.1 How do I capture log messages? 0
2.5.2 How do I compare documents? 0
2.5.3 How do I create a barcode? 0
2.5.4 How do I integrate with a custom OCR engine? 0
2.5.5 How do I provide AI-generated image descriptions? 0
2.5.6 How do I localize metadata? 0
3. Reference 0
3.1 Overview 0
3.2 C reference 0
3.2.1 C reference 0
3.2.2 Functions 0
3.2.3 Structures 0
3.2.4 Constants 0
3.3 Object reference 0
3.3.1 Overview 0
3.3.2 Annotation 0

© Hyland Software, Inc. All rights reserved.

3.3.3 AnnotationAction 0
3.3.4 AnnotationActionGoTo 0
3.3.5 AnnotationActionGoToRemote 0
3.3.6 AnnotationActionNamed 0
3.3.7 AnnotationActionUri 0
3.3.8 AnnotationAppearanceStream 0
3.3.9 AnnotationAppearanceStreams 0
3.3.10 AnnotationAztec 0
3.3.11 AnnotationBarcode 0
3.3.12 AnnotationBorderStyle 0
3.3.13 AnnotationCode128 0
3.3.14 AnnotationCode39 0
3.3.15 AnnotationDataMatrix 0
3.3.16 AnnotationDefaultAppearance 0
3.3.17 AnnotationEllipse 0
3.3.18 AnnotationFreeText 0
3.3.19 AnnotationGS1_128 0
3.3.20 AnnotationHighlight 0
3.3.21 AnnotationInk 0
3.3.22 AnnotationLine 0
3.3.23 AnnotationLineEndings 0
3.3.24 AnnotationLink 0
3.3.25 AnnotationNamedDestination 0
3.3.26 AnnotationNote 0
3.3.27 AnnotationNoteList 0
3.3.28 AnnotationPDF417 0
3.3.29 AnnotationPoint 0
3.3.30 AnnotationPointList 0
3.3.31 AnnotationPointListList 0
3.3.32 AnnotationPolygon 0

© Hyland Software, Inc. All rights reserved.

3.3.33 AnnotationPolyline 0
3.3.34 AnnotationPopup 0
3.3.35 AnnotationQrCode 0
3.3.36 AnnotationRect 0
3.3.37 AnnotationRectangle 0
3.3.38 AnnotationSquiggly 0
3.3.39 AnnotationStamp 0
3.3.40 AnnotationStickyNote 0
3.3.41 AnnotationStrikeout 0
3.3.42 AnnotationTextMarkup 0
3.3.43 AnnotationUnderline 0
3.3.44 Bookmark 0
3.3.45 Canvas 0
3.3.46 CompareDocumentSettings 0
3.3.47 CompareDocumentSource 0
3.3.48 CompareResultDifference 0
3.3.49 CompareResultDifferenceDetail 0
3.3.50 CompareResults 0
3.3.51 CompareSettings 0
3.3.52 DocumentFilters 0
3.3.53 Extractor 0
3.3.54 FormElement 0
3.3.55 FormElementOption 0
3.3.56 FormKeyValue 0
3.3.57 Hyperlink 0
3.3.58 IGRFormat 0
3.3.59 IGROption 0
3.3.60 Page 0
3.3.61 PageElement 0
3.3.62 RenderPageProperties 0

© Hyland Software, Inc. All rights reserved.

3.3.63 SubFile 0
3.3.64 Word 0
3.4 Structured XML 0
3.4.1 Overview 0
3.4.2 <doc> element 0
3.4.3 <head> element 0
3.4.4 <meta> element 0
3.4.5 <page> element 0
3.4.6 <header>/<footer> elements 0
3.4.7 <section> element 0
3.4.8 <column> element 0
3.4.9 <p> element 0
3.4.10 <line> element 0
3.4.11 <run> element 0
3.4.12 <word> element 0
3.4.13 <float> element 0
3.4.14 <img> element 0
3.4.15 <table> element 0
3.4.16 <tr> element 0
3.4.17 <td> element 0
4. Tech Info 0
4.1 Supported platforms 0
4.2 Platform Requirements 0
4.2.1 Windows 0
4.2.2 Linux 0
4.2.3 Alpine Linux 0
4.2.4 FreeBSD 0
4.2.5 Solaris 0
4.2.6 macOS 0
4.2.7 IBM AIX 0

© Hyland Software, Inc. All rights reserved.

4.2.8 Image Layout Detection and Machine Learning 0
4.3 Platform Differences 0
4.4 Supported File Formats 0
4.4.1 Archive 0
4.4.2 CAD 0
4.4.3 Database 0
4.4.4 Email and Messaging 0
4.4.5 Multimedia 0
4.4.6 Other 0
4.4.7 Presentation 0
4.4.8 Raster Image 0
4.4.9 Spreadsheets 0
4.4.10 Text and Markup 0
4.4.11 Vector Image 0
4.4.12 Word Processor and Office 0
4.5 Document format codes 0
4.6 Processing Options 0
4.7 Environment Variables 0
4.8 Third-Party Libraries 0
4.9 Known Issues 0
5. Releases 0
5.1 Releases 0
5.2 Document Filters 26.4.0 (Nov 2026) 0
5.3 Document Filters 26.3.0 (Aug 2026) 0
5.4 Document Filters 26.2.0 (May 2026) 0
5.5 Document Filters 26.1.0 (Feb 2026) 0
5.6 Document Filters 25.4.0 (Nov 2025) 0
5.7 Document Filters 25.3.0 (Aug 2025) 0
5.8 Document Filters 25.2.0 (May 2025) 0
5.9 Document Filters 25.1.0 (Feb 2025) 0

© Hyland Software, Inc. All rights reserved.

5.10 Document Filters 24.4.0 (Nov 2024) 0
5.11 Document Filters 24.3.0 (Aug 2024) 0
5.12 Document Filters 24.2.0 (May 2024) 0
5.13 Document Filters 24.1.0 (Feb 2024) 0
5.14 Document Filters 23.3.0 (Sep 2023) 0
5.15 Document Filters 23.2.0 (May 2023) 0
5.16 Document Filters 23.1.0 (Feb 2023) 0
5.17 Document Filters 22.4.0 (Nov 2022) 0
5.18 Document Filters 22.3.0 (Aug 2022) 0
5.19 Document Filters 22.2.0 (May 2022) 0
5.20 Document Filters 22.1.0 (Feb 2022) 0
5.21 Document Filters 21.11.0 (Nov 2021) 0
5.22 Document Filters 21.8.0 (Aug 2021) 0
5.23 Document Filters 21.5.0 (May 2021) 0
5.24 Document Filters 21.2.0 (Feb 2021) 0
5.25 Document Filters 11.4.20 (Nov 2020) 0
5.26 Document Filters 11.4.19 (Jul 2020) 0
5.27 Document Filters 11.4.18 (May 2020) 0
5.28 Document Filters 11.4.17 (Feb 2020) 0
5.29 Document Filters 11.4.16 (December 2019) 0
5.30 Document Filters 11.4.15 (September 2019) 0
5.31 Document Filters 11.4.14 (June 2019) 0
5.32 Document Filters 11.4.13 (March 2019) 0
5.33 Document Filters 11.4 Build 3054 (November 2018) 0
5.34 Document Filters 11.4 Build 2990 (September 2018) 0
5.35 Document Filters 11.4 Build 2934 (July 2018) 0
5.36 Document Filters 11.4 Build 2878 (May 2018) 0
5.37 Document Filters 11.4 Build 2822 (March 2018) 0
5.38 Document Filters 11.4 Build 2766 (January 2018) 0
5.39 Document Filters 11.4 Build 2710 (November 2017) 0

© Hyland Software, Inc. All rights reserved.

5.40 Document Filters 11.4 Build 2647 (September 2017) 0
5.41 Document Filters 11.4 Build 2600 (August 2017) 0
5.42 Document Filters 11.4 Build 2581 (July 2017) 0
5.43 Document Filters 11.4 Build 2543 (June 2017) 0
5.44 Document Filters 11.4 Build 2452 (March 2017) 0
5.45 Document Filters 11.4 Build 2480 (April 2017) 0
5.46 Document Filters 11.3 Build 2400 (January 2017) 0
5.47 Document Filters 11.3 Build 2356 (December 2016) 0
5.48 Document Filters 11.3 Build 2322 (November 2016) 0
5.49 Document Filters 11.3 Build 2228 (August 2016) 0
5.50 Document Filters 11.3 Build 2172 (June 2016) 0
5.51 Document Filters 11.3 Build 2116 (April 2016) 0
5.52 Document Filters 11.3 Build 2074 (March 2016) 0
5.53 Document Filters 11.3 Build 2040 (January 2016) 0
5.54 Document Filters 11.3 Build 1998 (December 2015) 0
5.55 Document Filters 11.2 Build 1962 (November 2015) 0
5.56 Document Filters 11.2 Build 1914 (September 2015) 0
5.57 Document Filters 11.2 Build 1884 (September 2015) 0
5.58 Document Filters 11.2 Build 1858 (August 2015) 0
5.59 Document Filters 11.2 Build 1808 (June 2015) 0
5.60 Document Filters 11.2 Build 1766 (May 2015) 0
5.61 Document Filters 11.2 Build 1732 (March 2015) 0
5.62 Document Filters 11.2 Build 1666 (January 2015) 0
5.63 Document Filters 11.2 (November 2014) 0
5.64 Document Filters 11.1 Build 1546 (October 2014) 0
5.65 Document Filters 11.1 Build 1464 (July 2014) 0
5.66 Document Filters 11.1 Build 1394 (April 2014) 0
5.67 Document Filters 11.1 Build 1333 (March 2014) 0
5.68 Document Filters 11.1 Build 1262 (December 2013) 0
5.69 Document Filters 11.1 (October 2013) 0

© Hyland Software, Inc. All rights reserved.

5.70 Document Filters 11.0 (July 2013) 0

© Hyland Software, Inc. All rights reserved.

1. Getting started

1.1 Getting started with Document Filters

Document Filters is a comprehensive SDK designed for software developers looking to integrate robust file identification, data extraction, document transformation, and format conversion features into their applications.

This SDK is available as Dynamic Link Libraries (DLLs) for Windows and Shared Objects (SOs) for UNIX-based systems. With Document Filters, developers can:

- Identify a wide range of file types.
- Extract text and metadata from hundreds of document formats.
- Retrieve sub-documents and attachments from various formats, including MS Office documents, ZIP files, RARs, 7-Zips, ISOs, CABs, PSTs, and OSTs.
- Convert popular document formats into High-Definition outputs that preserve styles, layouts, and images, with support for formats such as TIFF, HTML, PDF, and Structured XML.
- Utilize Canvas and Drawing functions for document markup, permanent annotations, and redaction.

For sample code and header definitions in C++, C#, HTML, Java, and Python, please visit GitHub - Hyland Document Filters.

1.1.1 In This Section

Getting Started with .NET	Hyland Document Filters provides robust document processing capabilities that can be easily integrated into your .NET applications. Follow these instructions to set up your environment.
Getting Started with C	Hyland Document Filters allows you to integrate powerful document processing capabilities into your C applications. Below are instructions on how to set up your application depending on your build system, such as CMake, Visual Studio, or Make.
Getting Started with C++	Hyland Document Filters allows you to integrate powerful document processing capabilities into your C++ applications. Below are instructions on how to set up your application depending on your build system, such as CMake, Visual Studio, or Make.
Getting Started with COM	Hyland Document Filters offers powerful document processing capabilities that can be seamlessly integrated into your Windows applications using COM. However, using COM involves extra setup steps, so first-level bindings (e.g., .NET, Java, or C++) should be chosen when available for easier integration.

© Hyland Software, Inc. All rights reserved.

Getting Started with Java	Hyland Document Filters provides robust document processing capabilities that can be easily integrated into your Java applications. Follow these instructions to set up your environment.
Getting Started with Python	Hyland Document Filters provides robust document processing capabilities that can be easily integrated into your Python applications. Follow these instructions to set up your environment.
About Accessibility Info Extraction	Document Filters supports the extraction of accessibility information from MS Office documents, aiding in the development of accessible products. The extracted "alt text" labels include images, drawings, objects, word art, smart art, charts, icons, shapes, and text boxes across Word (DOC and DOCX), PowerPoint (PPT and PPTX), Excel (XLS and XLSX), and Visio (VSDX) files. The extracted alt text is surfaced in various output formats -- HTML5, Classic HTML, XML, and PDF -- when the appropriate options for creating accessible output are provided.
About Conversion Profiles	The Hyland Document Filters allow you to customize conversion settings through a configuration file named `ISYS11df.ini`. This file should be placed in the executable folder. If present, the settings specified in this file will be applied during processing, unless explicitly overridden by the application in the `Open` or `Canvas` methods.
About Custom Streams and Extended Streams	Hyland Document Filters supports customizable streams, allowing you to read from storage systems not natively recognized by Document Filters. For example, you may want to read files directly from a database or an FTP site. Additionally, you can utilize a specialized type of custom stream known as an Extended Stream. Extended Streams help Document Filters retrieve additional information about your stream when needed, such as handling requests for specific parts of a multi-part archive.
About File Type Identification	File type identification is the process of determining a file's format based on its content rather than its extension. This is crucial for accurate processing, security scanning, and compatibility with different software applications. Hyland Document Filters identifies file types by analyzing a file’s byte stream, starting with the first 2048 bytes and reading additional data if necessary. It applies a combination of signature-based detection, heuristics, and container recognition to determine the format.
About Fonts	To effectively render a document into HD format, Document Filters must have access to the appropriate fonts to accurately represent each character on the page. Fonts not only provide the visual representation of characters but also contain size and measurement information necessary for proper content positioning.
About Multi-Part Archives	

© Hyland Software, Inc. All rights reserved.

	Multi-part archive files, such as certain ZIP and RAR formats, are comprised of two or more files packaged together. Document Filters supports the processing of these multi-part archives, allowing users to easily manage complex file structures.
About Multithreading	Document Filters may be run in a multithreaded application with minimal effort. Following a basic set of rules is essential to avoid issues such as data corruption or race conditions.
About Optical Character Recognition (OCR)	Document Filters provides comprehensive support for Optical Character Recognition (OCR), enabling users to extract text from a variety of document formats. It includes built-in support for the Tesseract OCR engine, which is a widely-used open-source OCR solution. Additionally, Document Filters allows for the use of other versions of Tesseract, giving you the flexibility to choose the version that best fits your needs.

Content Enrichment

Overview	The Content Enrichment section introduces key features that enhance the processing of documents by identifying and extracting valuable structural elements such as tables, headers, and footers. These capabilities allow users to transform raw document data into a more organized and meaningful format, facilitating improved data extraction and analysis. By leveraging these features, you can enrich your documents with critical insights and streamline workflows, making it easier to work with complex content.
Table Detection	Table Detection is a powerful feature of the Hyland Document Filters that enables automatic extraction and processing of tables from various document types. This functionality is particularly valuable when working with file formats that inherently store structured tabular data, such as Microsoft Office documents (Word, Excel, PowerPoint) and other similar productivity applications.
Image Description	Image Description lets Hyland Document Filters invoke a callback into your application whenever it encounters an image while processing a document, giving you the opportunity to extend or replace the image's alt text with a description from an external service — for example an AI vision model.
Image Layout Detection	Image Layout Detection uses a machine learning model to recover the structure of scanned pages and photographs. Optical Character Recognition (OCR) reads the words on an image, but it does not know which words form a heading, a table cell, or a page footer. Image Layout Detection finds those regions first, and OCR then reads the words into them.

Output Formats

© Hyland Software, Inc. All rights reserved.

Overview	Hyland Document Filters offers a variety of output modes, including text, images, PDF, and XML. This section delves into some of these options in more detail.
JSON Output	Hyland Document Filters provides robust options for generating JSON output from various document formats. By configuring the output settings, you can customize the generated JSON to suit your specific needs, from data structures to metadata inclusion.
Markdown Output	Hyland Document Filters provides powerful options for generating Markdown output from various document formats. By configuring the output settings, you can customize the generated Markdown to suit your specific needs, from table styles to metadata inclusion.
PDF Output	Hyland Document Filters offers robust features for generating PDF output from a variety of document formats. By adjusting the output settings, you can tailor the resulting PDF to meet your specific requirements, including layout preferences, security features, and metadata inclusion.

© Hyland Software, Inc. All rights reserved.

1.2 Getting Started with .NET

Hyland Document Filters provides robust document processing capabilities that can be easily integrated into your .NET applications. Follow these instructions to set up your environment.

1.2.1 Installing the Bindings

The Document Filters .NET bindings are available on NuGet under the package name Hyland.DocumentFilters, which provides the easiest and recommended way to integrate Document Filters into your .NET applications.

If needed, you can also access the source code for the bindings in the `bindings/dotnet` directory of the Document Filters GitHub repository. This allows for modifications and manual builds when customization is required.

> **Note**
>
> Earlier versions utilized `Perceptive.DocumentFilters` as a pre-compiled project, which remains included in the package. To upgrade a project to use `Hyland.DocumentFilters`, simply update the project reference and change the namespace accordingly.

> **Note**
>
> DocumentFilters DLLs are designed for a specific architecture. When your application targets AnyCPU, the architecture can be determined at runtime, so both 32-bit and 64-bit versions of the DocFilters DLLs must be present.
>
> If you're using the NuGet package and targeting .NET Core, the selection of the appropriate DLLs is automated. However, if you are targeting .NET Framework, you must ensure that the DLLs are discoverable according to the Default Probing rules for Unmanaged (native) libraries.

Using Published Bindings

To use the official Document Filters .NET bindings, you can add the `Hyland.DocumentFilters` package from NuGet to your project using Visual Studio or the command line.

Via the Command Line:

Run the following command in your terminal:

```
dotnet add package Hyland.DocumentFilters
```

Via Visual Studio:

© Hyland Software, Inc. All rights reserved.

Follow these steps to add the bindings via Visual Studio:

1. Open your project in Visual Studio.
2. In the Solution Explorer, right-click on your project and select **Manage NuGet Packages**.
3. In the **NuGet Package Manager** window, select the Browse tab.
4. In the search box, type `Hyland.DocumentFilters`.
5. Once found, click on the `Hyland.DocumentFilters` package and select the desired version.
6. Click Install to add the package to your project.
7. After installation, the package will be listed under your project's Dependencies in Solution Explorer, and you can start using the Document Filters API.

This ensures that you always have the latest stable version of the bindings, making integration and updates straightforward.

Building the Bindings

If you need to modify the .NET bindings or build them manually, follow these steps:

1. Clone the Document Filters GitHub repository.
2. Navigate to the `bindings/dotnet` directory.
3. Open the solution file in Visual Studio or another preferred IDE.
4. Make any necessary changes to the code.
5. Build the project to generate the custom .NET bindings for your application.

Once built, you can reference the custom bindings in your .NET project manually, or package them into a local NuGet package for reuse.

1.2.2 Initializing and calling Document Filters

C#

```
using Hyland.DocumentFilters;

var api = new Hyland.DocumentFilters.Api();
api.Initialize("YOUR_LICENSE_KEY_HERE", ".");
```

The preceding code loads the `DocumentFilters` package into global scope, then creates a new `Api` singleton and `Initialize`s it with a license key. Replace "YOUR_LICENSE_KEY_HERE" with your actual license key.

The
second argument controls where DocFilters should look for resources, such as configuration files and fonts. The `.` tells it to look in the same directory as the DocFilter's shared libraries.

© Hyland Software, Inc. All rights reserved.

Note: ISYSdf11.dll must be either in same folder as the currently executing Assembly or found by Default Probing rules for Unmanaged (native) libraries.

1.2.3 Extracting Text

C#

```
using Hyland.DocumentFilters;

var api = new Hyland.DocumentFilters.Api();
api.Initialize("YOUR_LICENSE_KEY_HERE", ".");

using var doc = api.GetExtractor("filename.doc");
doc.Open(Hyland.DocumentFilters.OpenType.BodyAndMeta)

while (!doc.EndOfStream)
{
    var text = doc.GetText(4096);
    Console.Out.WriteLine(text);
}
```

The preceding code loads the file `filename.doc` into an extractor `doc`. By using a scoped `using` block, the extractor will be closed when `doc` goes out of scope.

It then opens the extractor with `BodyAndMeta` indicating that we want to extract both the text (body) and the metadata of the document.

Finally, it loops over calling `GetText` until the extractor reports `EndOfStream`.

1.2.4 Converting a Document

C#

```
using Hyland.DocumentFilters;

var api = new Hyland.DocumentFilters.Api();
api.Initialize("YOUR_LICENSE_KEY_HERE", ".");

using var doc = api.GetExtractor("filename.doc");
using var canvas = api.MakeOutputCanvas("output.pdf", Hyland.DocumentFilters.CanvasType.PDF);

doc.Open(Hyland.DocumentFilters.OpenType.FormatImage)

canvas.RenderPages(doc);
```

The preceding code loads the file `filename.doc` into an extractor `doc`. By using a scoped `using` block, the extractor will be closed when `doc` goes out of scope.

It also creates a new `canvas` object of type `CanvasType.PDF` and stores it as `canvas`. The canvas will also be automatically closed when it goes out of scope.

It then opens the extractor with `FormatImage` indicating that we want to convert the file to an image based output. This triggers the file to be paginated.

© Hyland Software, Inc. All rights reserved.

Finally, it calls `RenderPages` to output every page from `doc` into the `canvas`. You could also manually iterate of over the pages and call `RenderPages`.

> **Did you know?**
>
> You can render more than one document to a canvas. If you want to stitch multiple files together, simply load each document into it's own `Extractor`, then call `RenderPage/s` onto a single canvas.

© Hyland Software, Inc. All rights reserved.

1.3 Getting Started with C

Hyland Document Filters allows you to integrate powerful document processing capabilities into your C applications. Below are instructions on how to set up your application depending on your build system, such as CMake, Visual Studio, or Make.

1.3.1 Clone and Include the Document Filters Repository

The Document Filters GitHub repository contains the necessary header files and libraries. If you are not using CMake's FetchContent, clone the repository manually into your project:

```
git clone https://github.com/Hyland/DocumentFilters.git
```

The main header file you need to work with is `DocumentFilters.h`, located in the `bindings/c/include` directory. If you're using CMake's `FetchContent`, manually cloning the repository is not required (details below).

1.3.2 Integrating with CMake

If you're using CMake, you can either manually clone the repository or use the `FetchContent` module to automatically fetch it without cloning. Here's how:

Using FetchContent (No Cloning Required)

CMake's `FetchContent` allows you to fetch the repository directly without manually cloning it. Add the following to your CMakeLists.txt:

CMakeLists.txt

```
include(FetchContent)

FetchContent_Declare(
    HylandDocumentFiltersC
    GIT_REPOSITORY https://github.com/Hyland/DocumentFilters.git
    GIT_TAG <desired-tag-or-branch>
    SOURCE_SUBDIR bindings/c
)

FetchContent_MakeAvailable(HylandDocumentFiltersC)
```

Once included, link the `DocumentFilters::C` library to your target:

CMakeLists.txt

```
target_link_libraries(your_target PRIVATE DocumentFilters::C)
```

© Hyland Software, Inc. All rights reserved.

Manual Cloning with CMake

If you prefer to manually clone the repository, first clone the repo into your project and then link it manually:

bash

```
git clone https://github.com/Hyland/DocumentFilters.git
```

After cloning, add the include directory and the library manually in your `CMakeLists.txt`:

CMakeLists.txt

```
add_subdirectory(path/to/DocumentFilters/bindings/c)
```

1.3.3 Integrating with Visual Studio (vcxproj-based)

To use Hyland Document Filters in a Visual Studio project, follow these steps:

Include the Header Files

- Go to **Project > Properties > Configuration Properties > C/C++ > General > Additional Include Directories**.
- Add the path to the bindings/c/include directory:

```
c:\path\to\DocumentFilters\bindings\c\include
```

Link the ISYS11df Library

- Go to **Project > Properties > Configuration Properties > Linker > General > Additional Library Directories**, and add the directory containing the ISYS11df.lib library:

```
c:\path\to\DocumentFilters\bindings\c\libs\x64
```

- In **Linker > Input > Additional Dependencies**, add:

```
ISYS11df.lib
```

1. Copy Required DLLs (if needed):
2. Ensure that the `ISYS11df.dll` file is accessible at runtime by copying it to your project output directory or ensuring it's available in your system's PATH.

© Hyland Software, Inc. All rights reserved.

1.3.4 Integrating with Makefiles projects

If you're using a manual build system, such as `Make` with a `Makefile`, you will need to manually specify the include paths and link the required libraries.

Include Header Files and Link Libraries in Makefile

Update your `Makefile` to include the necessary paths for headers and link the `ISYS11df` library. Here's an example setup:

Makefile

```
CC = gcc
CFLAGS = -Ipath/to/DocumentFilters/bindings/c/include
LDFLAGS = -Lpath/to/DocumentFiltersBinaries -lISYS11df

all: my_app

my_app: main.o
    $(CC) -o my_app main.o $(LDFLAGS)

main.o: main.c
    $(CC) $(CFLAGS) -c main.c

clean:
    rm -f *.o my_app
```

Compile the Application

After setting up the Makefile, you can build your application by running:

bash

```
make
```

With this approach, you'll need to manually download the Document Filters Release Binaries and specify their location as a link library directory, referred to above as path/to/DocumentFiltersBinaries.

1.3.5 Initializing and calling Document Filters

After setting up the necessary includes and linking the required libraries, the next step is to initialize the Document Filters library. This initialization is essential, as it prepares the library to process documents effectively. Below is a sample code snippet demonstrating how to perform this initialization in your C application.

© Hyland Software, Inc. All rights reserved.

C

```c
#include <stdio.h>
#include "DocumentFilters.h"

int main(int argc, char** argv)
{
    Instance_Status_Block isb;
    Error_Control_Block ecb;
    SHORT inst;

    strncpy_s(isb.Licensee_ID1, "YOUR_LICENSE_KEY_HERE", sizeof(isb.Licensee_ID1) - 1);
    Init_Instance(0, ".", &isb, &inst, &ecb);
    if (ecb.Msg[0])
        return 1;

    return 1;
}
```

In the code above, we initialize the global Document Filters library, which is required only once per application. Replace "YOUR_LICENSE_KEY_HERE" with your actual license key.

The second argument in the `Init_Instance` function specifies where the Document Filters library should search for resources, such as configuration files and fonts. By passing `.` as the argument, you instruct the library to look in the same directory as the shared libraries of Document Filters. This ensures that the library can access all necessary resources for proper functioning.

1.3.6 Extracting Text

Once the Document Filters library is initialized, you can start extracting text from documents. The following code snippet illustrates how to load a document and retrieve its contents using the Document Filters API. This example focuses on extracting text from a Word document (`.doc` file) and converting it to a more accessible format.

© Hyland Software, Inc. All rights reserved.

C

```c
#include <stdio.h>
#include <string.h>
#include "DocumentFilters.h"

const char LICENSE_KEY[] = "YOUR_LICENSE_KEY_HERE";
const char SOURCE_FILE[] = "filename.doc";

int main(int argc, char** argv)
{
    Instance_Status_Block isb = { 0 };
    Error_Control_Block ecb = { 0 };
    SHORT inst = 0;

    strncpy(isb.Licensee_ID1, LICENSE_KEY, sizeof(isb.Licensee_ID1) - 1);
    Init_Instance(0, ".", &isb, &inst, &ecb);
    if (ecb.Msg[0])
    {
        printf("%s\n", ecb.Msg);
        return 1;
    }

    IGR_UCS2 source_file[255] = { 0 };
    IGR_UCS2 canvas_file[255] = { 0 };
    IGR_UCS2 body_text[1024] = { 0 };
    IGR_LONG caps = 0, filetype = 0, handle = 0, body_size = sizeof(body_text) / sizeof(body_text[0]) - 1;
    char ascii_body_text[2048] = { 0 };

    UTF8_to_Widechar_Ex(SOURCE_FILE, sizeof(SOURCE_FILE) - 1, source_file, sizeof(source_file) /
sizeof(source_file[0]));

    if (IGR_Open_File_Ex(source_file, IGR_BODY_AND_META, NULL, &caps, &filetype, &handle, &ecb) != IGR_OK)
        goto bad;

    while (IGR_Get_Text(handle, body_text, &body_size, &ecb) == IGR_OK)
    {
        // convert the buffer from UTF-16 into UTF-8 for printing
        Widechar_to_UTF8_Ex(body_text, body_size, ascii_body_text, sizeof(ascii_body_text)-1);

        // replace DF markers
        for (size_t i = 0; i < sizeof(ascii_body_text) && ascii_body_text[i]; ++i)
        {
            if (ascii_body_text[i] == IGR_CHAR_PARA_BREAK)
                ascii_body_text[i] = '\n';
        }
        printf("%s", ascii_body_text);
        body_size = sizeof(body_text) / sizeof(body_text[0]) - 1;
    }

bad:
    if (ecb.Msg[0])
        printf("%s\n", ecb.Msg);
    if (handle)
        IGR_Close_File(handle, &ecb);
}
```

© Hyland Software, Inc. All rights reserved.

In the code snippet above, we begin by defining the license key and the source file name. The program initializes the Document Filters library and opens the specified document (`filename.doc`) using the `IGR_Open_File_Ex` function. This function provides a handle to the document, which is necessary for subsequent operations.

The Document Filters API uses UCS2/UTF-16 encoding for strings, so the example includes conversion functions to handle the text appropriately. The program enters a loop, calling `IGR_Get_Text` to retrieve text from the document until an error occurs. Each retrieved chunk of text is converted from UTF-16 to UTF-8 for easier printing.

If any error messages are encountered during the process, they are displayed to the user. Finally, the document handle is closed with `IGR_Close_File` to release any resources associated with it, ensuring proper cleanup.

1.3.7 Converting a Document

After successfully initializing the Document Filters library and extracting text, you can proceed to convert documents into different formats, such as PDF. The following code snippet demonstrates how to load a Word document (`.doc` file) and convert it into a PDF using the Document Filters API.

© Hyland Software, Inc. All rights reserved.

C

```c
#include <stdio.h>
#include <string.h>
#include "DocumentFilters.h"

const char LICENSE_KEY[] = "YOUR_LICENSE_KEY_HERE";
const char SOURCE_FILE[] = "filename.doc";
const char CANVAS_FILE[] = "output.pdf";

int main(int argc, char** argv)
{
    Instance_Status_Block isb = { 0 };
    Error_Control_Block ecb = { 0 };
    SHORT inst = 0;

    strncpy(isb.Licensee_ID1, LICENSE_KEY, sizeof(isb.Licensee_ID1) - 1);
    Init_Instance(0, ".", &isb, &inst, &ecb);
    if (ecb.Msg[0])
    {
        printf("%s\n", ecb.Msg);
        return 1;
    }

    IGR_UCS2 source_file[255] = { 0 };
    IGR_UCS2 canvas_file[255] = { 0 };
    IGR_UCS2 body_text[1024] = { 0 };
    IGR_LONG caps = 0, filetype = 0, doc_handle = 0, page_count = 0;
    HCANVAS canvas_handle = 0;
    HPAGE page_handle = 0;

    UTF8_to_Widechar_Ex(SOURCE_FILE, sizeof(SOURCE_FILE) - 1, source_file, sizeof(source_file) / sizeof(source_file[0]));
    UTF8_to_Widechar_Ex(CANVAS_FILE, sizeof(CANVAS_FILE) - 1, canvas_file, sizeof(canvas_file) / sizeof(canvas_file[0]));

    if (IGR_Open_File_Ex(source_file, IGR_FORMAT_IMAGE, NULL, &caps, &filetype, &doc_handle, &ecb) != IGR_OK)
        goto bad;

    if (IGR_Make_Output_Canvas(IGR_DEVICE_PDF, canvas_file, NULL, &canvas_handle, &ecb) != IGR_OK)
        goto bad;

    if (IGR_Get_Page_Count(doc_handle, &page_count, &ecb) != IGR_OK)
        goto bad;

    for (IGR_LONG page_index = 0; page_index < page_count; ++page_index)
    {
        if (IGR_Open_Page(doc_handle, page_index, &page_handle, &ecb) == IGR_OK)
        {
            IGR_Render_Page(page_handle, canvas_handle, &ecb);
            IGR_Close_Page(page_handle, &ecb);
        }
    }

bad:
    if (ecb.Msg[0])
        printf("%s\n", ecb.Msg);
    if (canvas_handle)
        IGR_Close_Canvas(canvas_handle, &ecb);
```

© Hyland Software, Inc. All rights reserved.

```
    if (doc_handle)
        IGR_Close_File(doc_handle, &ecb);
}
```

In this example, the program initializes the Document Filters library with a specified license key and opens the document (`filename.doc`) using the `IGR_Open_File_Ex` function. The file is opened in IGR_FORMAT_IMAGE, indicating that we want to convert it into an image-based output format, which will paginate the document accordingly.

Note: The Document Filters API uses UCS2/UTF-16 encoding for strings, so all input and output strings are converted into UTF-8.

The code then creates a new PDF canvas by calling `IGR_Make_Output_Canvas` with `IGR_DEVICE_PDF` , preparing for the output file.

Next, the program retrieves the total page count of the document with `IGR_Get_Page_Count` and enters a loop to process each page. For every page, it calls `IGR_Open_Page` to obtain a handle to the page, which is then rendered to the PDF canvas using `IGR_Render_Page` .

In case of any errors during the conversion process, appropriate messages are printed. Finally, the program ensures proper cleanup by closing the canvas and the document handle with `IGR_Close_Canvas` and `IGR_Close_File` , respectively. This structured approach ensures that resources are managed efficiently while converting documents.

1.3.8 Troubleshooting

If you encounter the error message:

```
libISYS11df.so: cannot open shared object file: No such file or directory
```

this indicates that the application is unable to find the Document Filters Shared Libraries. The standard `dlopen` rules apply for locating these libraries.

To resolve this issue, you can add the directory containing the libraries to the `LD_LIBRARY_PATH` environment variable. This adjustment will help the application locate the necessary shared libraries and eliminate the error.

© Hyland Software, Inc. All rights reserved.

1.4 Getting Started with C++

Hyland Document Filters allows you to integrate powerful document processing capabilities into your C++ applications. Below are instructions on how to set up your application depending on your build system, such as CMake, Visual Studio, or Make.

1.4.1 Clone and Include the Document Filters Repository

The Document Filters GitHub repository contains the necessary header files and libraries. If you are not using CMake's FetchContent, clone the repository manually into your project:

```
git clone https://github.com/Hyland/DocumentFilters.git
```

The primary header file required for your work is `DocumentFiltersObjects.h`, which can be found in the `bindings/cpp{VERSION}/include` directory. When using CMake's `FetchContent`, there is no need to manually clone the repository (details are provided below).

> **Note**
>
> Using the C++ bindings is not mandatory; your application can opt to interact directly with the C API instead. Follow the instructions on Getting Started with C

1.4.2 Choosing a C++ binding version

The Document Filters library offers bindings for C++17, located in the `bindings/cpp17` directory, as well as deprecated bindings for C++11 found in the `bindings/cpp11` directory. The C++17 bindings deliver comprehensive Object API functionality, encompassing annotations and document comparison, while the C++11 bindings offer limited support. Although the C++11 bindings are labeled as deprecated, they should only be utilized when a modern C++ compiler is unavailable.

Memory Management:

- **C++11 bindings**: Requires manual memory management with raw pointers.
- **C++17 bindings**: Automatic object lifetime management, eliminating manual `delete` calls. Objects are internally reference counted, and can be safely stored on the stack or heap without the caller needing to track their life-time.
- **Migration strategy**: REplace instances of manual memory management, including explicit calls to delete, or wrapping objects in smartptrs. Instead, objects should be placed on the stack, and allow them to be self-managed.

© Hyland Software, Inc. All rights reserved.

String Handling:

- **C++11 bindings**: Provided a mix of ansi string and wide string.
- **C++17 bindings**: Returned strings are normalized to std::wstring, with utility functions for converting between UTF-8 variants. Function parameters can be provided in either utf-8 or wide strings.
- **Migration strategy**: Use the provided string conversion function to convert between string types if your application does not work natively with `std::wstring`.

Object API Coverage:

- **C++11 bindings**: Limited coverage, restricted to text extraction, subfile handling, and rendering. Advanced features such as annotations and document comparison are not supported directly in the C++11 bindings and must be achieved via the C API.
- **C++17 bindings**: Provides full coverage of the Object API, including advanced features like annotations and document comparison, with feature parity across .NET, Python, and Java.
- **Migration strategy**: No required changes unless leveraging new features.

Stream Handling:

- **C++11 bindings**: Streams are handled via raw pointers and manual read/write operations.
- **C++17 bindings**: Streams are supported using modern C++ constructs, including std::istream and std::ostream, allowing for seamless integration with standard C++ I/O streams.
- **Migration strategy**: Replace raw pointer-based stream handling with std::istream and std::ostream-based implementations. This will ensure more efficient and safer stream operations.

Collection Iteration:

- **C++11 bindings**: Collections like lists or arrays are often iterated manually, typically using raw loops or index-based access.
- **C++17 bindings**: The bindings take advantage of range-based loops and standard C++ iterator patterns, improving readability and reducing potential off-by-one errors.
- **Migration strategy**: Migrate to range-based for-loops or use iterators for collections. This will make the code more concise and easier to maintain.

Callbacks:

- **C++11 bindings**: Callbacks are implemented using function pointers, requiring manual management and less flexibility when passing stateful functions.
- **C++17 bindings**: Callbacks can now be implemented using std::function, allowing the use of lambdas and stateful objects to handle callbacks more elegantly.

© Hyland Software, Inc. All rights reserved.

- **Migration strategy**: Replace function pointers with std::function where callbacks are used. This allows greater flexibility, enabling the use of lambdas or member functions with bound state.

Lambda Improvements:

- **C++11 bindings**: Basic lambda support exists, but capturing more complex objects or state (like move-only types) requires more manual management.
- **C++17 bindings**: C++17 lambdas support move-only types and generalized lambda captures, which allow for more flexible handling of objects within closures.
- **Migration strategy**: Refactor complex lambdas, especially those that handle state, to take advantage of C++17's generalized captures. This reduces boilerplate and makes callbacks and functional programming patterns easier to manage.

Filesystem Support:

- **C++11 bindings**: File handling may rely on older APIs or require third-party libraries to handle filesystem operations like directory iteration and path manipulations.
- **C++17 bindings**: C++17 brings native support for filesystem operations via the std::filesystem library, which includes functions for path manipulation, file I/O, and directory traversal.
- **Migration strategy**: Replace any custom or third-party filesystem libraries with std::filesystem. This simplifies file and directory handling with a standardized and efficient API.

1.4.3 Integrating with CMake

If you're using CMake, you can either manually clone the repository or use the `FetchContent` module to automatically fetch it without cloning. Here's how:

Using FetchContent

CMake's `FetchContent` allows you to fetch the repository directly without manually cloning it. Add the following to your CMakeLists.txt:

CMakeLists.txt

```
include(FetchContent)

FetchContent_Declare(
    HylandDocumentFiltersCpp17
    GIT_REPOSITORY https://github.com/Hyland/DocumentFilters.git
    GIT_TAG <desired-tag-or-branch>
    SOURCE_SUBDIR bindings/cpp17
)

FetchContent_MakeAvailable(HylandDocumentFiltersCpp17)
```

© Hyland Software, Inc. All rights reserved.

Once included, link the `DocumentFilters::Cpp17` library to your target:

CMakeLists.txt

```
target_link_libraries(your_target PRIVATE DocumentFilters::Cpp17)
```

Manual Cloning with CMake

If you prefer to manually clone the repository, first clone the repo into your project and then link it manually:

bash

```
git clone https://github.com/Hyland/DocumentFilters.git
```

After cloning, add the include directory and the library manually in your `CMakeLists.txt`:

CMakeLists.txt

```
add_subdirectory(path/to/DocumentFilters/bindings/cpp17)
```

1.4.4 Integrating with Visual Studio projects (vcxproj-based)

To use Hyland Document Filters in a Visual Studio project, follow these steps:

Include the DocumentFiltersCpp17 project in your Solution

- Right-click on your Solution in **Solution Explorer**, then select **Add > Existing Project**.
- Navigate to the `bindings/cpp17` directory and select the `DocumentFiltersCpp17.vcxproj` file.
- In your project, right-click on the project name, then select **Add > Reference*. Ensure that `DocumentFiltersCpp17` is checked.

Include the Header Files

- Go to **Project > Properties > Configuration Properties > C/C++ > General > Additional Include Directories**.
- Add the path to the `bindings/cpp17/include` and `bindings/c/include` directories, as follows:

```
c:\path\to\DocumentFilters\bindings\cpp17\include;c:
\path\to\DocumentFilters\bindings\c\include;
```

© Hyland Software, Inc. All rights reserved.

> **Note**
>
> You need to include both the `Cpp` and `C` include directories in your project.

Link the ISYS11df Library

- No additional steps are required to manually link the `ISYS11df` library, as it is already a dependency of the `DocumentFiltersCpp17` project.
- However, ensure that the `ISYS11df.dll` file is accessible at runtime. You can either copy it to your project output directory or make sure it is available in your system's `PATH`.

1.4.5 Integrating with Makefiles projects

If you're using a manual build system, such as `Make` with a `Makefile`, you will need to manually specify the include paths and link the required libraries.

Include Header Files, build object api and Link Libraries in Makefile

You can modify your `Makefile` to compile the source files and link the required libraries as specified in the CMake script. Below is an example `Makefile` setup:

Makefile

```
CXX = g++
CXXFLAGS = -std=c++17 -Ipath/to/DocumentFilters/bindings/c/include -Ipath/to/DocumentFilters/bindings/cpp17/
include
LDFLAGS = -Lpath/to/DocumentFiltersBinaries -lISYS11df
SOURCES = src/DocumentFiltersObjects.cpp src/DocFiltersAnnotations.cpp src/DocFiltersBookmark.cpp \
          src/DocFiltersCanvas.cpp src/DocFiltersCommon.cpp src/DocFiltersCompareDocumentSettings.cpp \
          src/DocFiltersCompareDocumentSource.cpp src/DocFiltersCompareResultDifference.cpp \
          src/DocFiltersCompareResultDifferenceDetail.cpp src/DocFiltersCompareResults.cpp \
          src/DocFiltersCompareSettings.cpp src/DocFiltersDateTime.cpp src/DocFiltersExtractor.cpp \
          src/DocFiltersFormat.cpp src/DocFiltersFormElement.cpp src/DocFiltersHyperlink.cpp \
          src/DocFiltersOption.cpp src/DocFiltersPage.cpp src/DocFiltersPageElement.cpp \
          src/DocFiltersPagePixels.cpp src/DocFiltersRenderPageProperties.cpp src/DocFiltersStreams.cpp \
          src/DocFiltersStrings.cpp src/DocFiltersSubFile.cpp src/DocFiltersWord.cpp
HEADERS = include/DocumentFiltersObjects.h

all: my_cpp_app

# Rule to build the DocumentFilters library
libDocumentFilters.a: $(SOURCES) $(HEADERS)
    $(CXX) $(CXXFLAGS) -c $(SOURCES)
    ar rcs libDocumentFilters.a *.o

# Link the library to the main application
```

© Hyland Software, Inc. All rights reserved.

```
my_cpp_app: main.o libDocumentFilters.a
    $(CXX) -o my_cpp_app main.o libDocumentFilters.a $(LDFLAGS)

main.o: main.cpp
    $(CXX) $(CXXFLAGS) -c main.cpp

clean:
    rm -f *.o my_cpp_app libDocumentFilters.a
```

Compile the Application

After setting up the Makefile, you can build your application by running:

bash

```
make
```

With this approach, you'll need to manually download the Document Filters Release Binaries and specify their location as a link library directory, referred to above as path/to/DocumentFiltersBinaries.

1.4.6 Initializing and calling Document Filters

After setting up the necessary includes and linking the required libraries, the next step is to initialize the Document Filters library. This initialization is essential, as it prepares the library to process documents effectively. Below is a sample code snippet demonstrating how to perform this initialization in your Cpp application.

C++

```
#include <iostream>
#include <memory>
#include <DocumentFiltersObjects.h>

static const char* LICENSE_KEY = "YOUR_LICENSE_KEY_HERE";

int main() {
    try {
        Hyland::DocFilters::Api api;
        api.Initialize(LICENSE_KEY, ".");
    } catch (const std::exception& error) {
        std::cout << "Initialization error: " << error.what() << std::endl;
        return 1;
    }

    std::cout << "Document Filters initialized successfully!" << std::endl;
    return 0;
}
```

© Hyland Software, Inc. All rights reserved.

Explanation:

- **License Key**: The LICENSE_KEY is provided as part of the initialization process and is required to unlock the full functionality of the Document Filters library. Replace "YOUR_LICENSE_KEY_HERE" with your actual license key.
- **Resource Directory**: The second parameter to Initialize, ".", specifies the directory where Document Filters should look for resources such as configuration files and fonts. In this case, "." means it will search the current working directory where the shared libraries reside. You can adjust this path based on your setup.
- **Error Handling**: The try-catch block captures any exceptions thrown during initialization and outputs the error message. This ensures you can handle initialization failures gracefully.

The DocumentFilters objects are now initialized and ready to be used for various document processing tasks, such as extraction, rendering, or comparison.

1.4.7 Extracting Text

Once the Document Filters library is initialized, you can begin extracting text from documents. The following C++ code snippet demonstrates how to load a document and extract its contents using the Document Filters API. This example focuses on extracting text from a Word document (.doc file).

C++

```cpp
#include <iostream>
#include <memory>
#include <DocumentFiltersObjects.h>

static const char* LICENSE_KEY = "YOUR_LICENSE_KEY_HERE";

int main() {
    try {
        Hyland::DocFilters::Api api;
        api.Initialize(LICENSE_KEY, ".");

        // Load the document for text extraction
        auto doc = api.GetExtractor(L"filename.doc");

        // Open the document for both body text and metadata extraction
        doc.Open(IGR_BODY_AND_META);

        // Loop through the document until EOF and extract text
        while (!doc->getEOF()) {
            auto text = doc.GetText();
            std::wcout << text << std::endl;
        }

    } catch (const std::exception& error) {
        std::cout << "Error: " << error.what() << std::endl;
        return 1;
    }
```

© Hyland Software, Inc. All rights reserved.

```
    return 0;
}
```

Explanation:

- **Document Loading**: The document is loaded using `api.GetExtractor`, which returns an object for extracting content. In this case, `"filename.doc"` is the input Word document. Update the file path accordingly.
- **Opening the Document**: The `doc.Open(IGR_BODY_AND_META)` method is called to open the document for text extraction. The `IGR_BODY_AND_META` flag specifies that both the document body and metadata should be extracted.
- **Text Extraction**: The code enters a loop, repeatedly calling `doc.GetText()` until the `getEOF()` method indicates the end of the document. The extracted text is output using std::wcout to handle wide characters.
- **Automatic Resource Management**: Since the document object is automatically managed, it will be closed when it goes out of scope, ensuring proper cleanup without manual intervention.

This approach efficiently extracts text from documents while managing resources automatically.

1.4.8 Converting a Document

After initializing the Document Filters library, you can convert documents into different formats, such as PDF. The following C++ code snippet demonstrates how to load a Word document (`.doc` file) and convert it into a PDF using the Document Filters API.

C++

```
#include <iostream>
#include <memory>
#include <DocumentFiltersObjects.h>

static const char* LICENSE_KEY = "YOUR_LICENSE_KEY_HERE";

int main() {
    try {
        Hyland::DocFilters::Api api;
        api.Initialize(LICENSE_KEY, ".");

        // Load the document for conversion
        auto doc = api.GetExtractor(L"filename.doc");

        // Create an output canvas for PDF rendering
        auto canvas = api.MakeOutputCanvas(L"result.pdf", IGR_DEVICE_IMAGE_PDF, "");

        // Open the document in image format for rendering
        doc.Open(IGR_FORMAT_IMAGE);

        // Loop through the document pages and render each one to the PDF canvas
        for (auto&& page : doc.pages()) {
            canvas.RenderPage(page);
        }
```

© Hyland Software, Inc. All rights reserved.

```
    } catch (const std::exception& error) {
        std::cout << "Error: " << error.what() << std::endl;
        return 1;
    }

    return 0;
}
```

Explanation:

- **Document Loading**: The document is loaded using `api.GetExtractor` with the path `"filename.doc"`. Replace this with your actual file path.
- **Canvas Creation**: A PDF canvas is created with the `MakeOutputCanvas` function, specifying the output file as `"result.pdf"`. The `IGR_DEVICE_IMAGE_PDF` flag is used to indicate that the output format is PDF.
- **Opening the Document**: The document is opened using `doc.Open(IGR_FORMAT_IMAGE)`, which prepares the document for rendering as an image-based format (used for PDF conversion).
- **Page Rendering**: The code iterates through each page of the document using a range-based loop. For each page, the `canvas.RenderPage(page)` method is called to render it onto the PDF canvas.
- **Automatic Resource Management**: Both the `doc` and `canvas` objects are automatically managed and will be closed when they go out of scope, ensuring proper cleanup.

This approach efficiently converts documents to PDF format while handling resources automatically.

© Hyland Software, Inc. All rights reserved.

1.5 Getting Stated with COM

Hyland Document Filters offers powerful document processing capabilities that can be seamlessly integrated into your Windows applications using COM. However, using COM involves extra setup steps, so first-level bindings (e.g., .NET, Java, or C++) should be chosen when available for easier integration.

1.5.1 Clone and Include the Document Filters Repository

The Document Filters GitHub repository contains the necessary files and libraries.

1.5.2 Additional Setup for COM

The `ISYS11df.dll` must be registered with Windows using the `regsvr32` command. This step is required for COM to function correctly.

1.5.3 Initializing and calling Document Filters

- Register and import the type library information from **ISYS11df.dll**.
- Create a "Hyland.DocumentFilters.11" object.
- Call `DocumentFilters.Initialize`.
- Call `DocumentFilters.GetExtractor` for one or more documents.
- Call `Extractor.GetText` to retrieve a document's text.
- Call `Extractor.Close` when finished with each document.

© Hyland Software, Inc. All rights reserved.