

PDFiumTrak Programmer's Reference

Contents

1. Introduction 4

1.1. What is PDFium? 4

1.2. What PDFiumTrak Does 4

1.3. Clarion Version and Prerequisites 6

1.4. PDFiumTrak Files 7

2. PDFiumTrak Setup 7

2.1. Overview 7

2.2. PDFiumTrak File Placement 8

2.3. Project Solution Explorer 9

2.4. Global MAP Embed 10

2.5. PDFiumTrakViewer.Clw 11

3. PDFiumTrak Function Reference 14

3.1. Connection and PDF Checks 14

3.1.1. PT_TestConnection 15

3.1.2. PT_CheckPDF 15

3.1.3. PT_GetPageCount 16

3.2. Build a PDF from Images 17

3.2.1. PT_BeginImagePDF 17

3.2.2. PT_AddImageFile 18

3.2.3. PT_AddJPGPage 19

3.2.4. PT_FinishImagePDF 20

3.2.5. PT_AbortImagePDF 21

3.3. Copy, Split, Merge, and Rotate PDFs 22

3.3.1. PT_SavePDFPages 22

3.3.2. PT_SplitPDF 24

3.3.3. PT_MergePDFs 26

3.3.4. PT_RotatePDFPage 28
3.3.5. PT_RotatePDFAllPages 30
3.4. Export PDF Pages to JPEG 33
3.4.1. PT_ExtractJPG 33
3.4.2. PT_ExtractTest 34
3.4.3. Choosing JPEG Size and Quality 36
3.5. Metadata, Text, and Bookmarks 37
3.5.1. PT_GetPDFMetadata 37
3.5.2. PT_SearchPDFPage 40
3.5.3. PT_FindPDFText 42
3.5.4. PT_GetPDFPageText 43
3.5.5. PT_GetPDFBookmarks 44
3.6. View and Print PDFs 45
3.6.1. PT_ViewPDF 46
3.6.2. PT_PrintPDF 49
3.7. PDFiumTrak Return Codes 50
4. PDFiumTrak Testbed Application 55
4.1. Overview 55
4.2. Tab 1 56
4.3. Tab 2 57
4.4. Tab 3 58
4.5. Tab 4 59
5. PDFiumTrak Support 60
5.1. Troubleshooting 60
5.2. Contact Us 61
5.3. Revision History 61

1. Introduction

PDFiumTrak is a Win32 PDF toolkit for 32-bit Clarion applications.

1.1. What is PDFium?

PDFium is an open-source C++ PDF rendering engine originally developed by Foxit Software (https://www.foxit.com/) and open-sourced by Google in 2014. It serves as the native PDF engine powering some highly visible software applications, to include:

- Google uses PDFium inside Chrome (the most widely used browser in the world).
- Microsoft uses PDFium inside Edge (the default web browser in Windows 10 and 11).
- Amazon uses PDFium inside Amazon Echo and Fire TV products.
- Dropbox uses PDFium inside its client tools to preview files.

Key Reasons Developers Choose PDFium

- High Rendering Accuracy: Because it is the core engine behind Chromium, the pages it parses and rasterizes match what users see natively in the browser, handling complex layouts, fonts, and "scanned-ish" PDFs very reliably.
- Performance and Scale: Exercised daily by billions of browser users, it handles high-speed page rendering, bitmap rasterization, text extraction, and form-filling under heavy production loads.
- Permissive License: Released under a permissive BSD license, making it free to integrate into commercial, open-source, or enterprise software products without restrictive copyleft requirements.
- Active Enterprise Support: Backed and contributed to by major corporate leaders (Google, Microsoft, Dropbox, and specialized SDK vendors like Nutrient), ensuring regular security patch cycles and vulnerability fixes.

PDFium is released under the permissive New BSD License (3-Clause BSD). You are free to use, modify, distribute, and integrate it into both personal and commercial projects without paying licensing fees, per the vendor.

You can explore the source code and documentation directly on the PDFium Source Repository.

1.2. What PDFiumTrak Does

PDFiumTrak is a "wrapper" for PDFium which permits your Clarion application work with PDF files. You can display a customer invoice in a Clarion window, combine a folder of scanned images into one PDF, or save just the pages you need from a longer document (amongst other common tasks):

CONNECTION AND PDF CHECKS

- PT_TestConnection - Returns 12345 to confirm that Clarion can call the DLL.
- PT_CheckPDF - Tries to open the PDF and load page 1. Returns 0 if both succeed or 1 if the file should be skipped. This check has no timeout.
- PT_GetPageCount - Returns the number of pages in a PDF.

EXPORT PDF PAGES TO JPEG

- PT_ExtractJPG - Renders one PDF page to a JPEG at the requested width and quality. Pass the page number, width, and quality as LONG variables by reference.
- PT_ExtractTest - Performs the same JPEG export using one pipe-delimited command string.

COPY, SPLIT, MERGE, AND ROTATE PDFS

- PT_SavePDFPages - Copies selected pages into a new PDF. Accepts lists and ranges such as 1,3-5 and preserves the requested order.
- PT_SplitPDF - Writes each source page to its own numbered PDF in a new folder. Returns the number of files created.
- PT_MergePDFs - Combines multiple PDFs in the supplied file order. Reports the input position if a source file fails.
- PT_RotatePDFPage - Rotates one selected page and saves a new copy of the entire PDF. The other pages keep their orientation.
- PT_RotatePDFAllPages - Rotates every page by the requested angle and saves a new copy of the PDF. Use 180 degrees to correct an upside-down scan.

METADATA, TEXT, AND BOOKMARKS

- PT_GetPDFMetadata - Reads one metadata field, such as Title, Author, Keywords, or CreationDate, into a Clarion CSTRING.
- PT_SearchPDFPage - Counts case-insensitive text matches on one specified page.
- PT_FindPDFText - Searches forward from a starting page. Returns the first matching page and reports how many matches are on it.
- PT_GetPDFPageText - Extracts the existing text from one page into a Clarion CSTRING. Returns the number of UTF-8 bytes written.
- PT_GetPDFBookmarks - Lists bookmark titles, nesting levels, and destination pages for bookmarks that point into the PDF.

BUILD A PDF FROM IMAGES

- PT_BeginImagePDF - Starts a new image-to-PDF job and returns its job ID through a LONG variable.
- PT_AddImageFile - Adds a BMP, JPG, PNG, or TIFF to the job. Each frame of a multipage TIFF becomes a PDF page.
- PT_AddJPGPage - Adds a JPEG directly as one page in the job.
- PT_FinishImagePDF - Saves and verifies the completed PDF, then releases the job.
- PT_AbortImagePDF - Cancels the job and cleans up its temporary output.

VIEW AND PRINT PDFS

- PT_ViewPDF - Opens a resizable Clarion viewer with page navigation, zoom, text search, a custom title, and optional printing.

    - Bookmarks - Appears when the PDF has bookmarks pointing to its pages. Open the list and select a bookmark to jump to its page. The BOOKMARK button is hidden if the PDF doesn't contain embedded bookmarks
        - Rotate 180 - Flips the current page or all pages, saves a new PDF, and opens the saved copy at the same page.
        - Optionally hide the PRINT button within the PDF viewer window; optionally change the PDF viewer window title
        - The Clarion source code file (*.CLW) is included with PDFiumTrak purchase - you can modify the viewer's appearance and behavior to meet your application needs
- PT_PrintPDF - Opens the Windows print dialog for printer selection, page ranges, and copies.

Your application calls the PDFiumTrak DLL for these operations. If you want to display PDFs inside your application, add the included full-featured PDFiumTrak Clarion viewer module.

To get started, check Clarion Version and Prerequisites, then follow PDFiumTrak Setup. The testbed application and sample files let you try the functions before adding PDFiumTrak to your own application.

1.3. Clarion Version and Prerequisites

PDFiumTrak is a 32-bit PDF toolkit for Windows applications developed with Clarion. The supplied testbed application was built and tested using Clarion 12, build 14000.

Clarion version
Your application must target 32-bit Windows (Win32). Compatibility with other Clarion versions should be verified by compiling and testing your application.

The supplied PDFiumTrak.lib import library was generated using Clarion 12, build 14000. If your Clarion version cannot link this library, use LibMaker from your Clarion installation to generate a new import library from the supplied PDFiumTrak.dll. Then rebuild and test your application.

Development requirements
To call the PDFiumTrak functions from your application, add PDFiumTrak.lib to your project and add the required function declarations to your Global MAP. See the PDFiumTrak Setup topics for instructions.

To use the included PDF viewer, also add PDFiumTrakViewer.clw to your project as a compiled source module. Update its MEMBER declaration to reference your application's main .CLW file, as described in the PDFiumTrakViewer.CLW help topic.

Runtime requirements
Deploy the following files alongside your application's executable:
- PDFiumTrak.dll — The 32-bit PDFiumTrak function library.
- pdfium.dll — The matching 32-bit PDFium engine supplied with this release.

The supplied build also requires the Microsoft Visual C++ runtime for x86. Include the supplied redistributable runtime files, msvcp140.dll and vcruntime140.dll, alongside your executable, or install a compatible Microsoft Visual C++ x86 Redistributable on the destination computer.

Include the Clarion runtime files required by your application.

PDFiumTrak.lib is used when building your application. PDFiumTrakViewer.clw is compiled into your application when viewer support is included.

Before deployment
Test your application on a computer that has neither Clarion nor Visual Studio installed. This confirms that your distribution includes the runtime files needed to execute it.

1.4. PDFiumTrak Files

PDFiumTrak consists of the following files:

Build files

- PDFiumTrak.lib — The import library you add to your Clarion project so it can call the DLL functions.
- PDFiumTrakViewer.clw — The Clarion source module for the included viewer. Add it to your project if you want to open PDFs in a window within your application.

Runtime files

- PDFiumTrak.dll — The 32-bit DLL that provides the toolkit's PDF functions.
- pdfium.dll — The PDFium engine used by the toolkit. This release was tested with build 159.0.8138.0. Deploy the supplied copy alongside PDFiumTrak.dll and your application's executable.

Test files

- PDFiumTrak Test Fixtures.zip — Original sample PDFs and BMP, JPG, PNG, and TIFF images for the testbed. The included README describes the expected results so you can verify your setup.

2. PDFiumTrak Setup

Install the runtime files, link the import library, and add the declarations before trying the examples.

2.1. Overview

Use the supplied 32-bit DLLs and Clarion import library to add PDFiumTrak to your application. Start with a 32-bit Clarion project.

1. Place the runtime files. Copy PDFiumTrak.dll and pdfium.dll beside your application's executable. See PDFiumTrak File Placement.

2. Add the import library. Add PDFiumTrak.lib to your Clarion project. If you want the included PDF viewer, also add PDFiumTrakViewer.clw as a source module. See PDFiumTrakViewer.clw for instructions how to modify, and then include, this source file in your application.
3. Add the declarations. Copy the declarations for the functions you plan to call into the Global MAP Embed.
4. Verify the connection. Build and run your application, then call PT_TestConnection(). A return value of 12345 confirms the call reached the PDFiumTrak DLL. Next, call PT_GetPageCount() with the supplied Good.pdf. It should return 5.

2.2. PDFiumTrak File Placement

The PDFiumTrak library consists of these files:

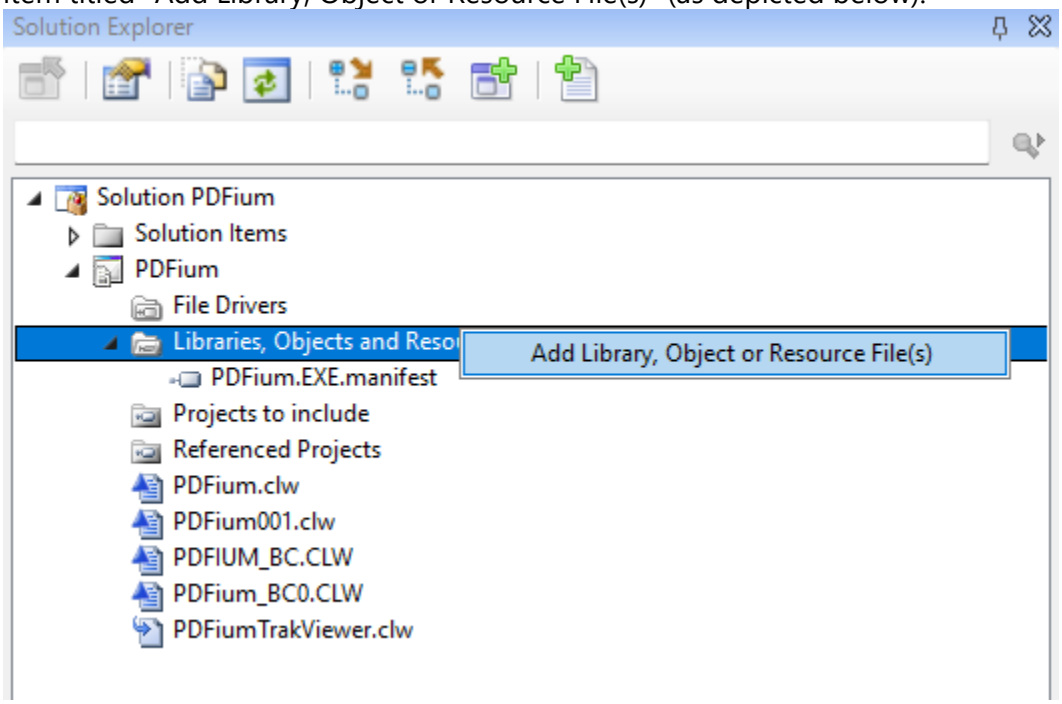
PDFiumTrak Files	Suggested Installation Location
PDFiumTrak.dll	<Your Clarion12 Folder Location>\Accessory\Bin
Pdfium.dll	<Your Clarion12 Folder Location>\Accessory\Bin
PDFiumTrak.Lib	<Your Clarion12 Folder Location>\Accessory\Lib
PDFiumTrakViewer.clw	<Your Clarion12 Folder Location>\Accessory\Libsrc\Win

Your compiled Clarion application requires the two .DLL files (Pdfium.dll and PDFiumTrak.dll) to execute - it is recommended they reside in your .Exe folder. See Project Solution Explorer for how to add the build files to your project.

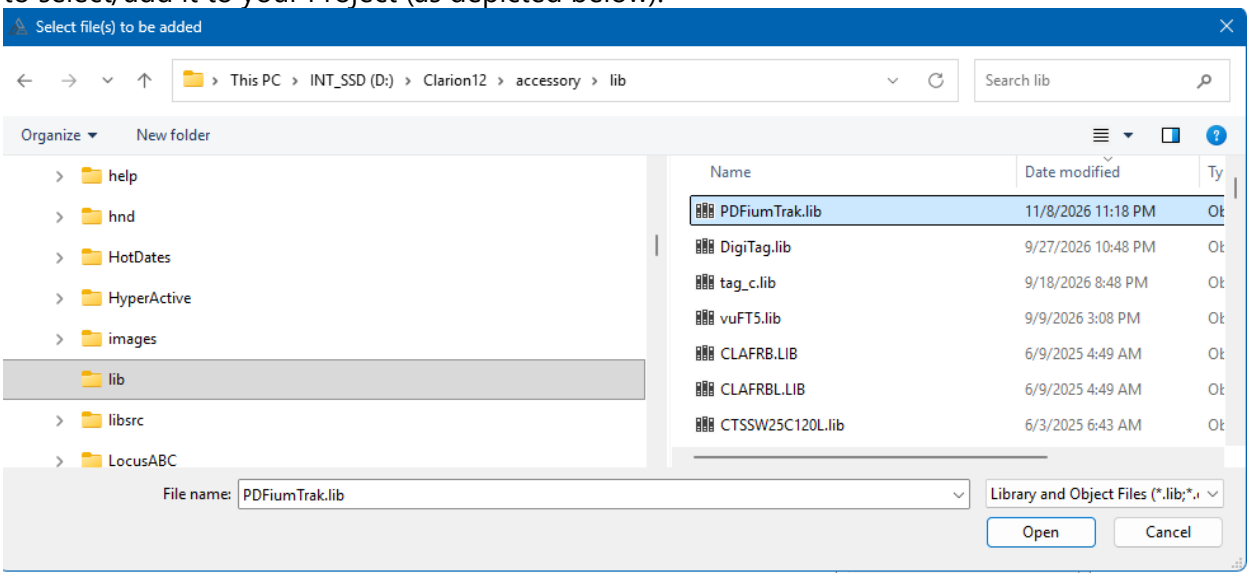
2.3. Project Solution Explorer

Add the PDFiumTrak.Lib file to your application (mandatory):

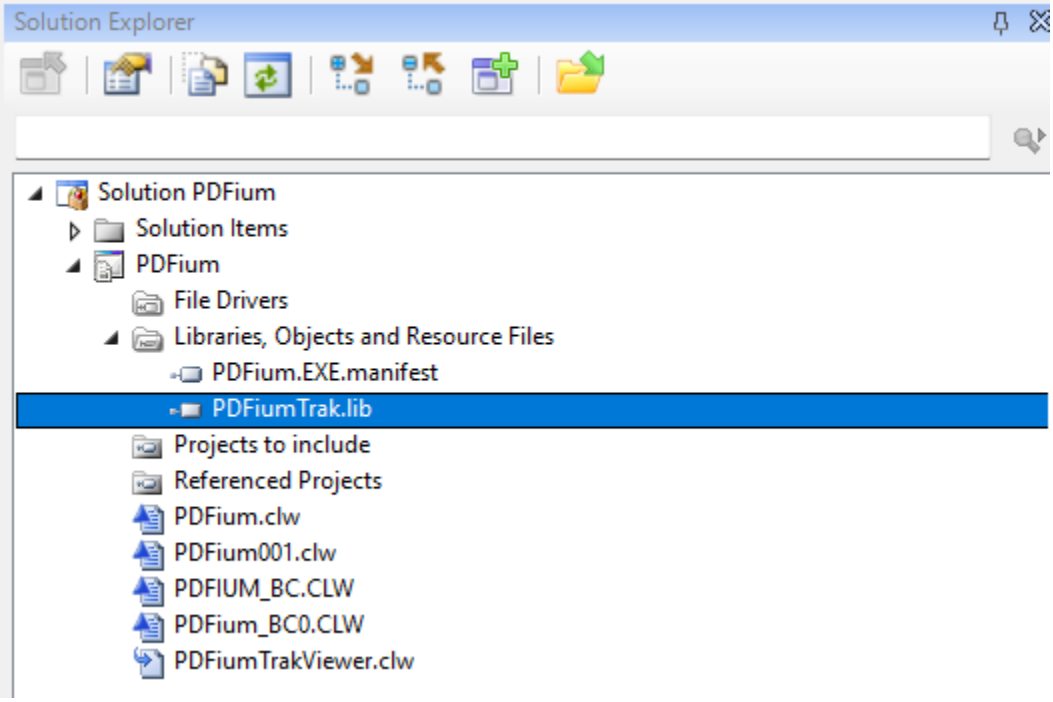
1. Open your application in the Clarion IDE
2. Open the Project Explorer; right click on Libraries, Objects and Resources, and then click the popup menu item titled "Add Library, Object or Resource File(s)" (as depicted below):



3. Navigate to your <Clarion12 install location>\accessory\lib folder, then double left click on PDFiumTrak.lib to select/add it to your Project (as depicted below):



4. The Solution Explorer will now show the PDFiumTrak.lib file added to the application (as depicted below):



5. *Save the project*

The next step (and final) configuration step is adding the code block into the Global MAP embed

2.4. Global MAP Embed

Open your application's Global Map

1. Click the Global Embeds tab
2. Locate the embed titled "Inside the Global Map"; click the INSERT button and select SOURCE
3. Copy/paste the following block of code into the new SOURCE, flush aligned with card column 1, line number 1:

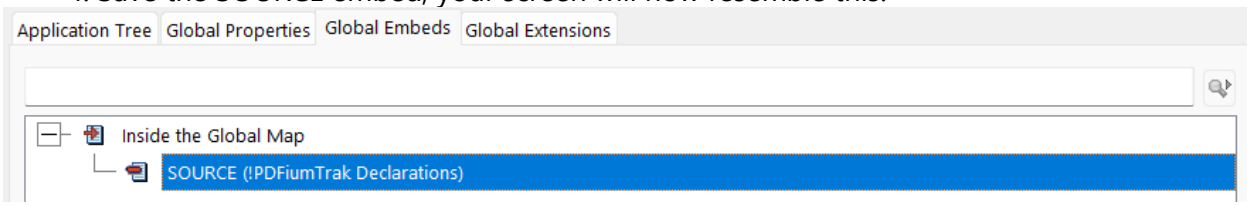
```
!PDFiumTrak Declarations
INCLUDE('CWUTIL.INC'),ONCE
MODULE('PDFiumTrak.lib')
  PT_TestConnection(),LONG,RAW,DLL(dll_mode),NAME('_PT_TestConnection@0')
  PT_GetPageCount(*CSTRING),LONG,RAW,PASCAL,NAME('_PT_GetPageCount@4')
  PT_ExtractJPG(*CSTRING,*LONG,*CSTRING,*LONG,*LONG),LONG,RAW,PASCAL,NAME('_PT_ExtractJPG@20')
  PT_ExtractTest(*CSTRING),LONG,RAW,PASCAL,NAME('_PT_ExtractTest@4')
  PT_CheckPDF(*CSTRING),LONG,RAW,PASCAL,NAME('_PT_CheckPDF@4')
  PT_BeginImagePDF(*CSTRING,*LONG),LONG,RAW,PASCAL,NAME('_PT_BeginImagePDF@8')
  PT_AddJPGPage(LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_AddJPGPage@8')
```

```
  PT_FinishImagePDF(LONG),LONG,RAW,PASCAL,NAME('_PT_FinishImagePDF@4')
  PT_AbortImagePDF(LONG),LONG,RAW,PASCAL,NAME('_PT_AbortImagePDF@4')
  PT_AddImageFile(LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_AddImageFile@8')
  PT_PrintPDF(*CSTRING,ULONG),LONG,RAW,PASCAL,NAME('_PT_PrintPDF@8')
  PT_SearchPDFPage(*CSTRING,LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_SearchPDFPage@12')
  PT_SavePDFPages(*CSTRING,*CSTRING,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_SavePDFPages@12')
  PT_GetPDFMetadata(*CSTRING,*CSTRING,*CSTRING,LONG),LONG,RAW,PASCAL,NAME('_PT_GetPDFMetadata@16')
  PT_MergePDFs(*CSTRING,*CSTRING,*LONG),LONG,RAW,PASCAL,NAME('_PT_MergePDFs@12')
  PT_FindPDFText(*CSTRING,LONG,*CSTRING,*LONG),LONG,RAW,PASCAL,NAME('_PT_FindPDFText@16')
  PT_SplitPDF(*CSTRING,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_SplitPDF@8')
  PT_GetPDFPageText(*CSTRING,LONG,*CSTRING,LONG),LONG,RAW,PASCAL,NAME('_PT_GetPDFPageText@16')
  PT_GetPDFBookmarks(*CSTRING,*CSTRING,LONG,LONG),LONG,RAW,PASCAL,NAME('_PT_GetPDFBookmarks@16')
  PT_RotatePDFPage(*CSTRING,LONG,LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_RotatePDFPage@16')
  PT_RotatePDFAllPages(*CSTRING,LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_RotatePDFAllPages@12')
END
```

If you want to use the PDFiumTrak PDF viewer function, add these four lines of code underneath the previously pasted block of code:

```
! We are using the PDFiumTrak PDF viewer in this application, so this MODULE must be included too:
MODULE('PDFiumTrakViewer.clw')
   PT_ViewPDF(*CSTRING,<STRING>,BYTE=1)
END
```

4. Save the SOURCE embed; your screen will now resemble this:



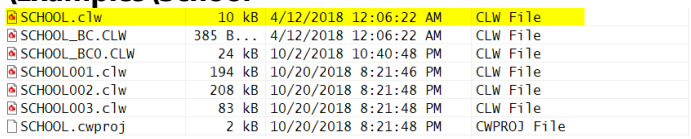
2.5. PDFiumTrakViewer.Clw

Add the PDFiumTrak PDF Viewer function to your application (optional):

1. Copy the PDFiumTrakViewer.Clw file from the Clarion12 libsrc folder (e.g. <Clarion12 install folder>\accessory\libsrc\win) into your application's folder where it's .APP file resides (e.g. D:\Clarionapps\PDFium)
2. Open the PDFiumTrakViewer.Clw file in Notepad (or your preferred text editor). Change line #1 of that file which currently contains **MEMBER('PDFium.clw')** and modify that line to the same as your current

project's primary .CLW name. For example, using two classic SoftVelocity example applications (School and Solodex), here are the primary .CLW files within their respective folders:

\Examples\School



SCHOOL.clw	10 kB	4/12/2018 12:06:22 AM	CLW File
SCHOOL_BC.CLW	385 B...	4/12/2018 12:06:22 AM	CLW File
SCHOOL_BC0.CLW	24 kB	10/2/2018 10:40:48 PM	CLW File
SCHOOL001.clw	194 kB	10/20/2018 8:21:46 PM	CLW File
SCHOOL002.clw	208 kB	10/20/2018 8:21:48 PM	CLW File
SCHOOL003.clw	83 kB	10/20/2018 8:21:48 PM	CLW File
SCHOOL.cwproj	2 kB	10/20/2018 8:21:48 PM	CWPROJ File

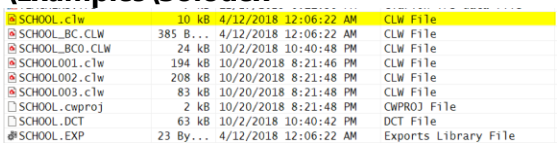
Change the first line of the PDFiumTrakViewer.Clw file from

```
1   MEMBER('PDFium.clw')
2
3  ! Add this file to the Clarion project as a source module.
4  ! In the program's Global MAP embed, add the following:
5  !
6  !   MODULE('PDFiumTrakViewer.clw')
7  !     PT_ViewPDF(*CSTRING,<STRING>,BYTE=1)
8  !   END
```

to instead match the projects master .CLW name:

```
1   MEMBER('School.clw')
2
3  ! Add this file to the Clarion project as a source module.
4  ! In the program's Global MAP embed, add the following:
5  !
6  !   MODULE('PDFiumTrakViewer.clw')
7  !     PT_ViewPDF(*CSTRING,<STRING>,BYTE=1)
8  !   END
```

\Examples\Solodex



SCHOOL.clw	10 kB	4/12/2018 12:06:22 AM	CLW File
SCHOOL_BC.CLW	385 B...	4/12/2018 12:06:22 AM	CLW File
SCHOOL_BC0.CLW	24 kB	10/2/2018 10:40:48 PM	CLW File
SCHOOL001.clw	194 kB	10/20/2018 8:21:46 PM	CLW File
SCHOOL002.clw	208 kB	10/20/2018 8:21:48 PM	CLW File
SCHOOL003.clw	83 kB	10/20/2018 8:21:48 PM	CLW File
SCHOOL.cwproj	2 kB	10/20/2018 8:21:48 PM	CWPROJ File
SCHOOL.DCT	63 kB	10/2/2018 10:40:42 PM	DCT File
SCHOOL.EXP	23 By...	4/12/2018 12:06:22 AM	Exports Library File

Change the first line of the PDFiumTrakViewer.Clw file from

```
1   MEMBER('PDFium.clw')
2
3  ! Add this file to the Clarion project as a source module.
4  ! In the program's Global MAP embed, add the following:
5  !
6  !   MODULE('PDFiumTrakViewer.clw')
7  !     PT_ViewPDF(*CSTRING,<STRING>,BYTE=1)
8  !   END
```

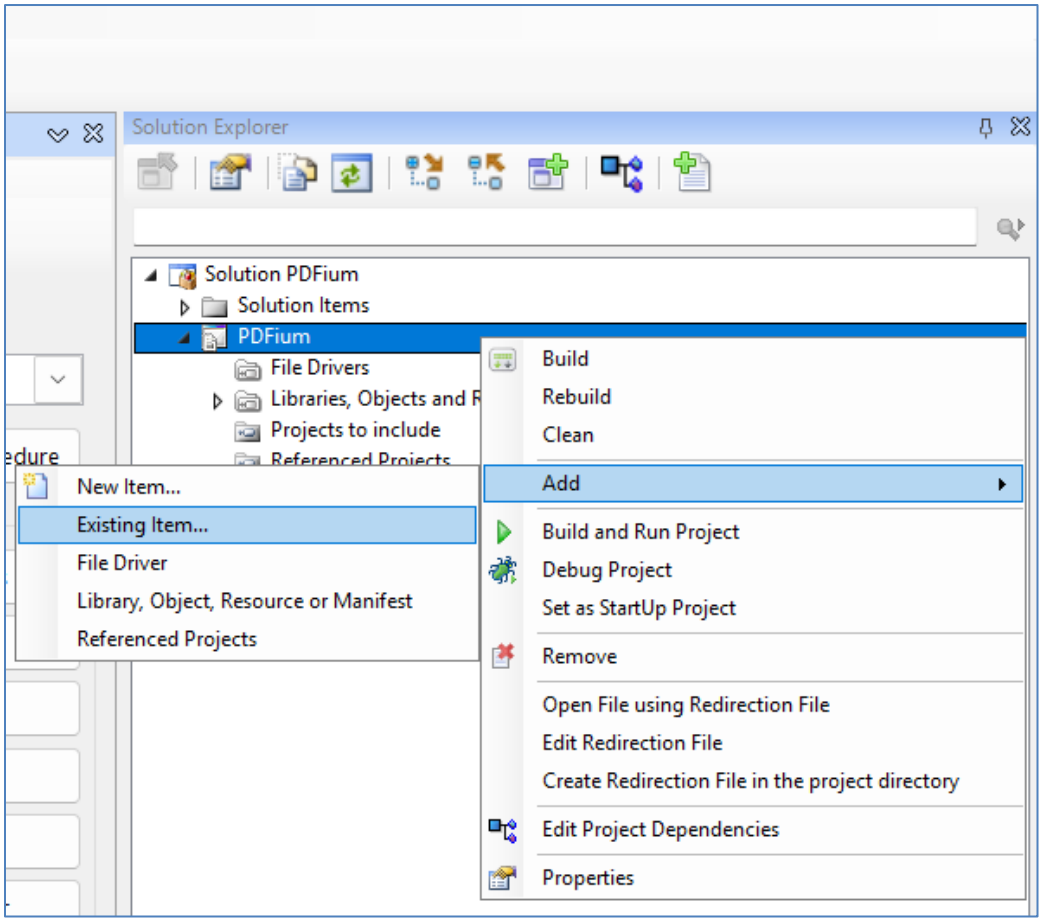
to instead match the projects master .CLW name:

```
1   MEMBER('Solodex.clw')
2
3  ! Add this file to the Clarion project as a source module.
4  ! In the program's Global MAP embed, add the following:
5  !
6  !   MODULE('PDFiumTrakViewer.clw')
7  !     PT_ViewPDF(*CSTRING,<STRING>,BYTE=1)
8  !   END
```

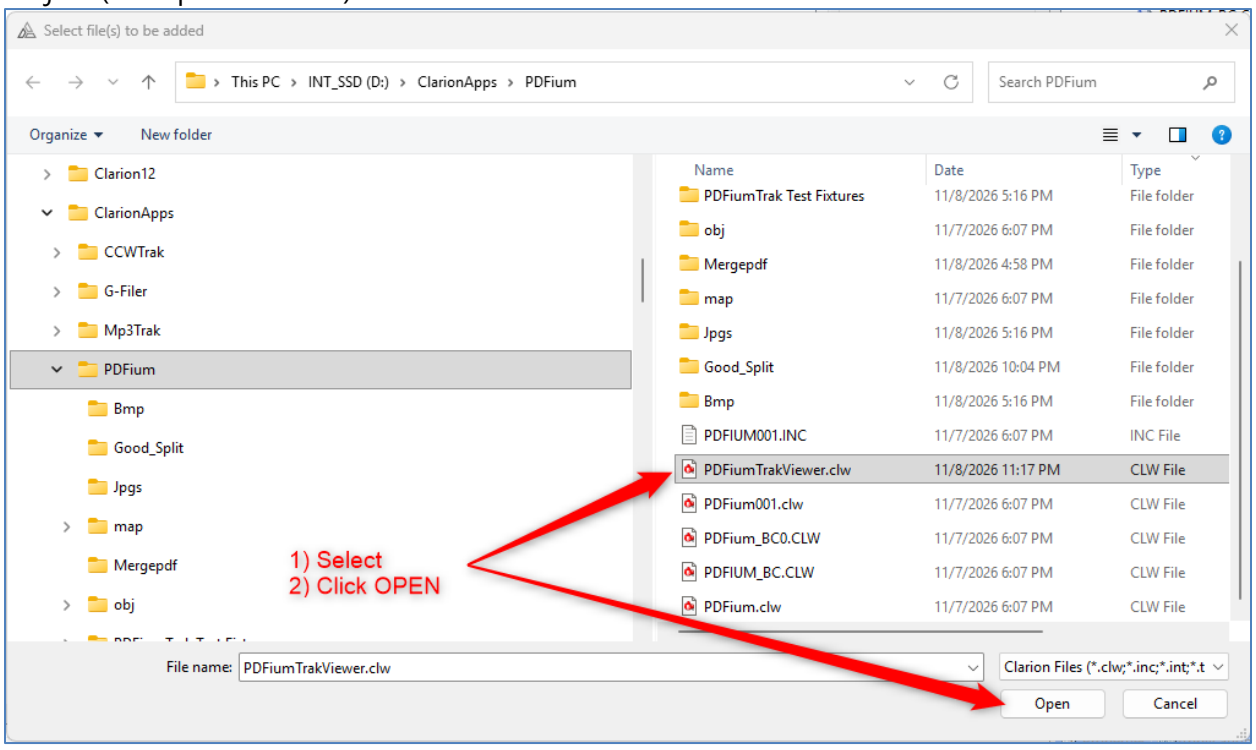
and then save the PDFiumTrakViewer.Clwfile

and then save the PDFiumTrakViewer.Clw file

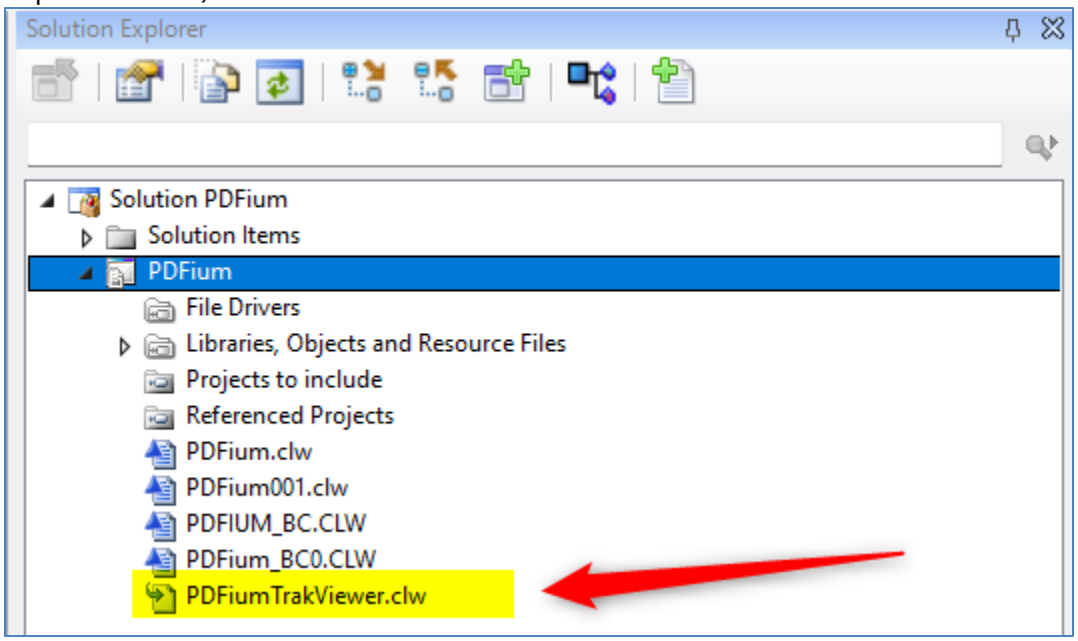
3. Open the Project Explorer; right click on your application's name in the Solution Explorer tree (in this case, the app is named "PDFium"). Select "Add" from the popup menu, and then select "Existing Item" (as depicted below):



4. Navigate to application folder, then double left click on PDFiumTrakViewer.clw to select/add it to your Project (as depicted below):



5. The Solution Explorer will now show the PDFium PDF viewer source file added to the application (as depicted below):



6. Save the project

3. PDFiumTrak Function Reference

This section provides detailed information for each of the PDFiumTrak functions, organized by category:

- Connection and PDF Checks
- Build a PDF from Images
- Copy, Split, Merge and Rotate PDFs
- Export PDF Pages to JPEG
- Metadata, Text, and Bookmark Retrieval
- View and Print PDFs

3.1. Connection and PDF Checks

Select a function below for its declaration, return values, and Clarion example.
PT_TestConnection
PT_CheckPDF
PT_GetPageCount

3.1.1. PT_TestConnection

Tests whether Clarion can call the PDFiumTrak DLL.

Global MAP declaration
```
PT_TestConnection(),LONG,RAW,DLL(dll_mode),NAME('_PT_TestConnection@0')
```

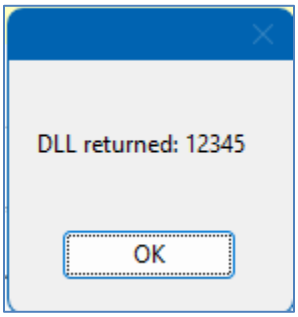
Parameter
This function takes no arguments.

Returns
12345 - The expected DLL export responded.

Local Data
```
Loc:Result LONG
```

Button Accepted embed
```
Loc:Result = PT_TestConnection()
MESSAGE('PT_TestConnection returned ' & Loc:Result)
```

Expected result
The popup reports 12345.



3.1.2. PT_CheckPDF

Tries to open a PDF and load page 1. This in-process check has no timeout and only checks the first page.

Global MAP declaration
```
PT_CheckPDF(*CSTRING),LONG,RAW,PASCAL,NAME('_PT_CheckPDF@4')
```

Parameter
PdfC - Full PDF path in a UTF-8 CSTRING.

Returns
0 - Opened and loaded page 1.
1 - Skip this file. The argument, document, or first page failed the check.

Local Data

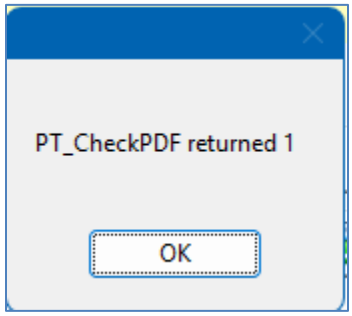
```
Loc:PdfC CSTRING(512)
Loc:Result LONG
```

Button Accepted embed

```
Loc:PdfC = 'D:\ClarionApps\PDFium\Bad.pdf'
Loc:Result = PT_CheckPDF(Loc:PdfC)
MESSAGE('PT_CheckPDF returned ' & Loc:Result)
```

Expected result

The deliberately incomplete Bad.pdf fixture returns 1.



3.1.3. PT_GetPageCount

Returns the number of pages in a PDF. Use it to validate a page number or set the upper bound of a page loop.

Global MAP declaration

Add this line inside the existing MODULE('PDFiumTrak.lib') block.

```
PT_GetPageCount(*CSTRING),LONG,RAW,PASCAL,NAME('_PT_GetPageCount@4')
```

Parameter

PdfPathC - A CSTRING containing the full PDF filename. Supply UTF-8 for paths with non-ASCII characters.

Returns

0 or greater - Number of pages.
-1 - Invalid PDF path argument.
-2 - PDFium could not open the PDF, or the operation failed.

Local Data

```
Loc:CountPdfC CSTRING(512)
Loc:PageCount LONG
```

Button Accepted embed

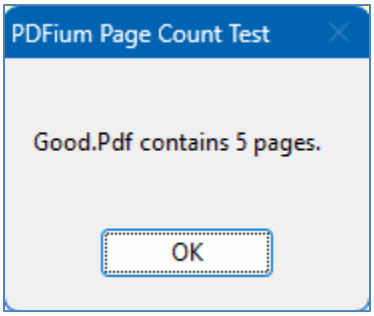
Use Good.pdf from the PDFiumTrak Test Fixtures. Update the path if your testbed is installed elsewhere.

```
Loc:CountPdfC = 'D:\ClarionApps\PDFium\Good.pdf'
Loc:PageCount = PT_GetPageCount(Loc:CountPdfC)

IF Loc:PageCount >= 0
 MESSAGE('Pages in Good.pdf: ' & Loc:PageCount, 'PDF Page Count')
ELSE
 MESSAGE('PT_GetPageCount returned ' & Loc:PageCount, 'PDF Page Count')
END
```

Expected result

The PDF Page Count message shows Pages in Good.pdf: 5.



3.2. Build a PDF from Images

Select a function below for its declaration, return values, and Clarion example.

PT_BeginImagePDF
PT_AddImageFile
PT_AddJPGPage
PT_FinishImagePDF
PT_AbortImagePDF

3.2.1. PT_BeginImagePDF

Starts an image-to-PDF job and opens temporary output.

Global MAP declaration

```
PT_BeginImagePDF(*CSTRING,*LONG),LONG,RAW,PASCAL,NAME('_PT_BeginImagePDF@8')
```

Parameters

OutputC - Path to a new PDF.
Job - LONG passed by reference to receive the job ID.

Returns
0 - Success.
-1 - Invalid argument.
-4 - Could not create temporary output.
-6 - Destination already exists.
-7 - Could not allocate a job ID.

Local Data
```
Loc:OutputC CSTRING(512)
Loc:ImageC CSTRING(512)
Loc:Job LONG
Loc:Result LONG
Loc:Cleanup LONG
```

Button Accepted embed
```
Loc:OutputC = 'D:\ClarionApps\PDFium\BeginImagePDF_Demo.pdf'
Loc:ImageC = 'D:\ClarionApps\PDFium\Jpgs\01.jpg'
Loc:Result = PT_BeginImagePDF(Loc:OutputC, Loc:Job)
IF Loc:Result = 0
 Loc:Result = PT_AddJPGPage(Loc:Job, Loc:ImageC)
 IF Loc:Result = 0
 Loc:Result = PT_FinishImagePDF(Loc:Job)
 END
 IF Loc:Result <> 0
 Loc:Cleanup = PT_AbortImagePDF(Loc:Job)
 IF Loc:Cleanup <> 0
 MESSAGE('Temporary output cleanup failed.')
 END
 END
END
MESSAGE('Image PDF workflow returned ' & Loc:Result)
```

Expected result
The result is 0. The workflow creates a one-page PDF.

3.2.2. PT_AddImageFile

Adds a BMP, JPG, PNG, or TIFF. Each supported TIFF frame becomes one page.

Global MAP declaration
```
PT_AddImageFile(LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_AddImageFile@8')
```

Parameters
Job - Active job ID, passed by value.
ImageC - Full input image path.

Returns
0 - Success.
-1 - Invalid argument.
-2 - Invalid or failed job.

-3 - Image could not be read or is unsupported.
-4 - Image or output processing failed.
-7 - Size or resource limit reached.

Local Data

```
Loc:OutputC CSTRING(512)
Loc:ImageC CSTRING(512)
Loc:Job LONG
Loc:Result LONG
Loc:Cleanup LONG
```

Button Accepted embed

```
Loc:OutputC = 'D:\ClarionApps\PDFium\AddImageFile_Demo.pdf'
Loc:ImageC = 'D:\ClarionApps\PDFium\Png\01.png'
Loc:Result = PT_BeginImagePDF(Loc:OutputC, Loc:Job)
IF Loc:Result = 0
 Loc:Result = PT_AddImageFile(Loc:Job, Loc:ImageC)
 IF Loc:Result = 0
 Loc:Result = PT_FinishImagePDF(Loc:Job)
 END
 IF Loc:Result <> 0
 Loc:Cleanup = PT_AbortImagePDF(Loc:Job)
 IF Loc:Cleanup <> 0
 MESSAGE('Temporary output cleanup failed.')
 END
 END
END
MESSAGE('Image PDF workflow returned ' & Loc:Result)
```

Expected result

The result is 0. The workflow creates a one-page PDF.

3.2.3. PT_AddJPGPage

Embeds a supported JPEG directly as a single page.

Global MAP declaration

```
PT_AddJPGPage(LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_AddJPGPage@8')
```

Parameters

Job - Active job ID, passed by value.
ImageC - Full JPEG path.

Returns

0 - Success.
-1 - Invalid argument.
-2 - Invalid or failed job.
-3 - JPEG could not be read or is unsupported.
-4 - Output processing failed.

-7 - Size or resource limit reached.

Local Data

```
Loc:OutputC CSTRING(512)
Loc:ImageC CSTRING(512)
Loc:Job LONG
Loc:Result LONG
Loc:Cleanup LONG
```

Button Accepted embed

```
Loc:OutputC = 'D:\ClarionApps\PDFium\AddJPGPage_Demo.pdf'
Loc:ImageC = 'D:\ClarionApps\PDFium\Jpgs\01.jpg'
Loc:Result = PT_BeginImagePDF(Loc:OutputC, Loc:Job)
IF Loc:Result = 0
 Loc:Result = PT_AddJPGPage(Loc:Job, Loc:ImageC)
 IF Loc:Result = 0
 Loc:Result = PT_FinishImagePDF(Loc:Job)
 END
 IF Loc:Result <> 0
 Loc:Cleanup = PT_AbortImagePDF(Loc:Job)
 IF Loc:Cleanup <> 0
 MESSAGE('Temporary output cleanup failed.')
 END
 END
END
MESSAGE('Image PDF workflow returned ' & Loc:Result)
```

Expected result

The result is 0. The workflow creates a one-page PDF.

3.2.4. PT_FinishImagePDF

Verifies and publishes the PDF, then releases the job on success.

Global MAP declaration

```
PT_FinishImagePDF(LONG),LONG,RAW,PASCAL,NAME('_PT_FinishImagePDF@4')
```

Parameter

Job - Active job ID, passed by value.

Returns

0 - Success.
-1 - The job has no pages.
-2 - Invalid or failed job.
-4 - Could not finish writing the PDF.
-5 - Output verification failed.
-6 - Destination already exists.

Local Data

```
Loc:OutputC CSTRING(512)
Loc:ImageC CSTRING(512)
Loc:Job LONG
```

```
Loc:Result LONG
Loc:Cleanup LONG
```

Button Accepted embed

```
Loc:OutputC = 'D:\ClarionApps\PDFium\FinishImagePDF_Demo.pdf'
Loc:ImageC = 'D:\ClarionApps\PDFium\Jpgs\01.jpg'
Loc:Result = PT_BeginImagePDF(Loc:OutputC, Loc:Job)
IF Loc:Result = 0
 Loc:Result = PT_AddJPGPage(Loc:Job, Loc:ImageC)
 IF Loc:Result = 0
 Loc:Result = PT_FinishImagePDF(Loc:Job)
 END
 IF Loc:Result <> 0
 Loc:Cleanup = PT_AbortImagePDF(Loc:Job)
 IF Loc:Cleanup <> 0
 MESSAGE('Temporary output cleanup failed.')
 END
 END
END
MESSAGE('Image PDF workflow returned ' & Loc:Result)
```

Expected result

The result is 0. The workflow creates a one-page PDF.

3.2.5. PT_AbortImagePDF

Cancels a job and removes its temporary output.

Global MAP declaration

```
PT_AbortImagePDF(LONG),LONG,RAW,PASCAL,NAME('_PT_AbortImagePDF@4')
```

Parameter

Job - Job ID returned by PT_BeginImagePDF, passed by value.

Returns

0 - Cleaned up or the job was already absent.
-4 - Cleanup failed.

Local Data

```
Loc:OutputC CSTRING(512)
Loc:ImageC CSTRING(512)
Loc:Job LONG
Loc:Result LONG
Loc:Cleanup LONG
```

Button Accepted embed

```
Loc:OutputC = 'D:\ClarionApps\PDFium\AbortImagePDF_Demo.pdf'
Loc:ImageC = 'D:\ClarionApps\PDFium\Jpgs\01.jpg'
Loc:Result = PT_BeginImagePDF(Loc:OutputC, Loc:Job)
IF Loc:Result = 0
 Loc:Result = PT_AddJPGPage(Loc:Job, Loc:ImageC)
 IF Loc:Result = 0
 Loc:Result = PT_AbortImagePDF(Loc:Job)
 END
 IF Loc:Result <> 0
 Loc:Cleanup = PT_AbortImagePDF(Loc:Job)
 IF Loc:Cleanup <> 0
 MESSAGE('Temporary output cleanup failed.')
 END
 END
END
MESSAGE('Image PDF workflow returned ' & Loc:Result)
```

Expected result
The result is 0. The job is canceled and no PDF is published.

3.3. Copy, Split, Merge, and Rotate PDFs

Select a function below for its declaration, return values, and Clarion example.
Copy selected pages
Split into one-page files
Merge files
Rotate one page
Rotate all pages

3.3.1. PT_SavePDFPages

Copies the specified pages into a new PDF in the order supplied. Repeated page numbers are allowed.

Global MAP declaration
```
PT_SavePDFPages(*CSTRING,*CSTRING,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_SavePDFPages@12')
```

Parameters
PdfC - Full PDF path in a UTF-8 CSTRING.
PagesC - Comma-separated page numbers and inclusive ranges. Page numbers start at 1.
OutputC - Full path for the new file.

Returns
0 - Success.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Page is outside the source PDF.
-4 - Could not create the new PDF or import a page.

-5 - Output could not be written, verified, or published.
-6 - Destination already exists.

Local Data

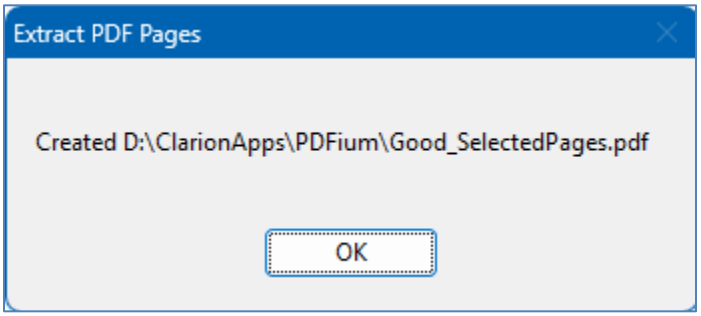
```
Loc:PdfC CSTRING(512)
Loc:PagesC CSTRING(128)
Loc:OutputC CSTRING(512)
Loc:Result LONG
```

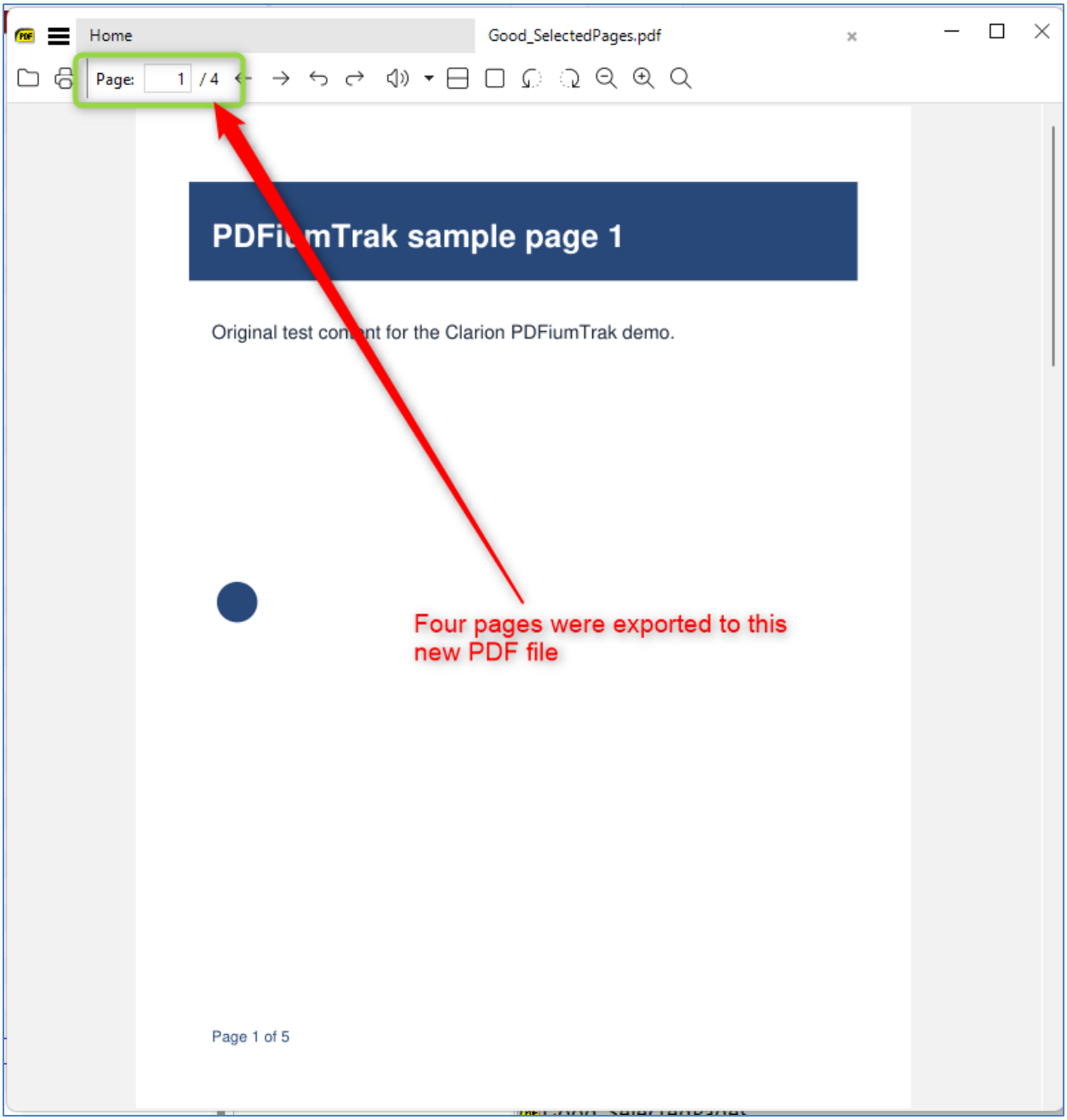
Button Accepted embed

```
Loc:PdfC = 'D:\ClarionApps\PDFium\Good.pdf'
Loc:PagesC = '1,3-5'
Loc:OutputC = 'D:\ClarionApps\PDFium\Good_SelectedPages.pdf'
Loc:Result = PT_SavePDFPages(Loc:PdfC, Loc:PagesC, Loc:OutputC)
MESSAGE('PT_SavePDFPages returned ' & Loc:Result)
```

Expected result

The saved PDF has four pages (original PDF file had five pages). Original page 3 is output page 2.





When running this example again, choose a new output filename or rename the previously generated file. If the destination file already exists, PT_SavePDFPages returns -6.

3.3.2. PT_SplitPDF

Writes every page to a separate numbered PDF and publishes the output folder after verification.

Global MAP declaration

```
PT_SplitPDF(*CSTRING,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_SplitPDF@8')
```

Parameters
PdfC - Full PDF path in a UTF-8 CSTRING.
FolderC - A new output folder under an existing parent folder.

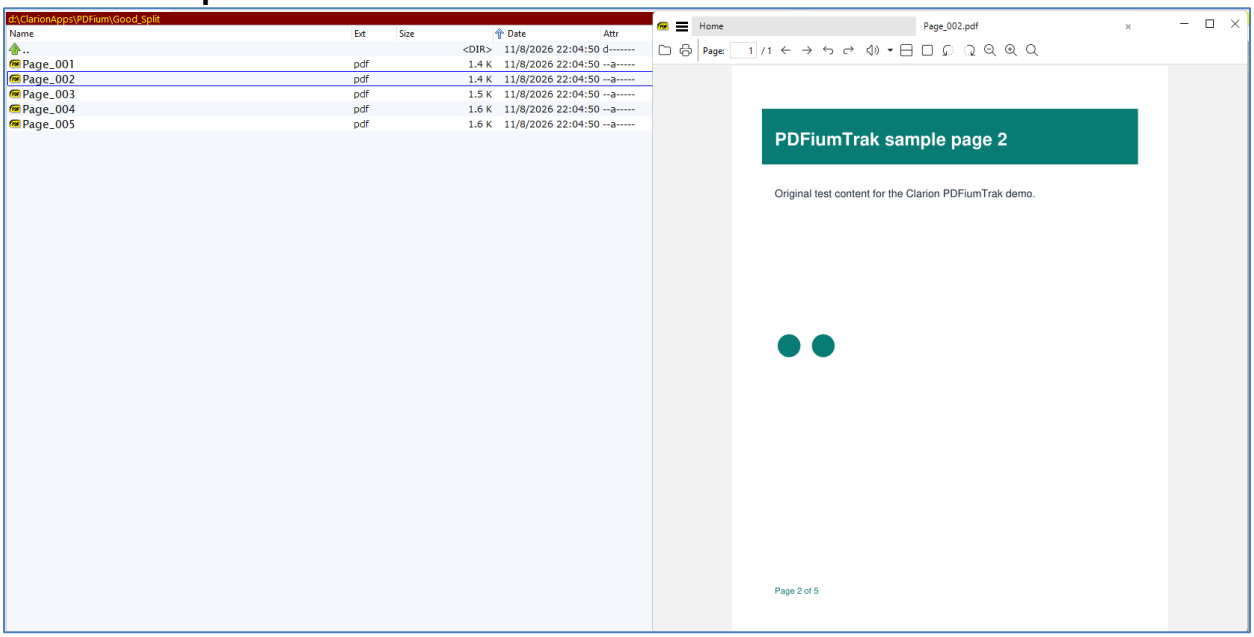
Returns
> 0 - Number of files created.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Source has no pages.
-4 - Could not create the new PDF or import a page.
-5 - Output could not be written, verified, or published.
-6 - Destination already exists.

Local Data
```
Loc:PdfC CSTRING(512)
Loc:FolderC CSTRING(512)
Loc:Result LONG
```

Button Accepted embed
```
Loc:PdfC = 'D:\ClarionApps\PDFium\Good.pdf'
Loc:FolderC = 'D:\ClarionApps\PDFium\Good_Split'
Loc:Result = PT_SplitPDF(Loc:PdfC, Loc:FolderC)
MESSAGE('PT_SplitPDF returned ' & Loc:Result)
```

Expected result
The result is 5. Good_Split contains Page_001.pdf through Page_005.pdf.

Runtime example



Page_002.pdf reopens as a one-page PDF showing source page 2.

3.3.3. PT_MergePDFs

Appends every page from each input PDF in the supplied file order.

Global MAP declaration

```
PT_MergePDFs(*CSTRING,*CSTRING,*LONG),LONG,RAW,PASCAL,NAME('_PT_MergePDFs@12')
```

Parameters

SourcesC - Pipe-delimited list of full input paths. Output order follows input order.
OutputC - Full path for the new file.
Failed - LONG passed by reference. Receives the 1-based failing input position or zero.

Returns

0 - Success.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - An input contains no pages.
-4 - Could not create the new PDF or import a page.
-5 - Output could not be written, verified, or published.
-6 - Destination already exists.

Local Data

```
Loc:SourcesC CSTRING(2048)
Loc:OutputC CSTRING(512)
Loc:Failed LONG
Loc:Result LONG

MergePDFQ QUEUE PRE(MergePDF)
  Name String(256)
  ShortName String(13)
  Date LONG
  Time LONG
  Size LONG
  Attrib BYTE
END

Loc:MergeSources CSTRING(32768)
Loc:MergeFolder CSTRING(512)
Loc:CurrentPath String(255)
Loc:MergeRow LONG
Loc:MergeFile String(1024)
Loc:MergeOutput CSTRING(512)
Loc:MergeResult LONG
Loc:MergeFailed LONG
```

Button Accepted embed

DO MergePDFDirectory

*---------- The ROUTINE below is a separate ROUTINE procedure within this window

```
MergePDFDirectory ROUTINE
```

```
! Retrieve the current path
Loc:CurrentPath = path()

! Build the queue
FREE(MergePDFQ)
Loc:MergeSources = ''
Loc:MergeFolder = CLIP(Loc:CurrentPath) & '\Mergepdf'
Loc:MergeOutput = CLIP(Loc:CurrentPath) & '\Merged_From_Folder.pdf'

DIRECTORY(MergePDFQ, CLIP(Loc:MergeFolder) & '\*.pdf', ff_:NORMAL)
IF RECORDS(MergePDFQ) = 0
    MESSAGE('No PDFs found in ' & CLIP(Loc:MergeFolder), 'Merge PDFs')
    EXIT
END

SORT(MergePDFQ, +MergePDF:Name)

LOOP Loc:MergeRow = 1 TO RECORDS(MergePDFQ)
    GET(MergePDFQ, Loc:MergeRow)
    IF ERRORCODE()
        MESSAGE('Could not read directory entry #' & Loc:MergeRow, 'Merge PDFs')
        FREE(MergePDFQ)
        EXIT
    END

    Loc:MergeFile = CLIP(Loc:MergeFolder) & '\' & CLIP(MergePDF:Name)
    IF Loc:MergeRow > 1
        Loc:MergeFile = '|' & CLIP(Loc:MergeFile)
    END

    IF LEN(CLIP(Loc:MergeSources)) + LEN(CLIP(Loc:MergeFile)) >= |
       SIZE(Loc:MergeSources)
        MESSAGE('The PDF list is too large for the merge buffer.', 'Merge PDFs')
        FREE(MergePDFQ)
        EXIT
    END

    Loc:MergeSources = CLIP(Loc:MergeSources) & CLIP(Loc:MergeFile)
END

Loc:MergeResult = PT_MergePDFs(Loc:MergeSources, Loc:MergeOutput, Loc:MergeFailed)

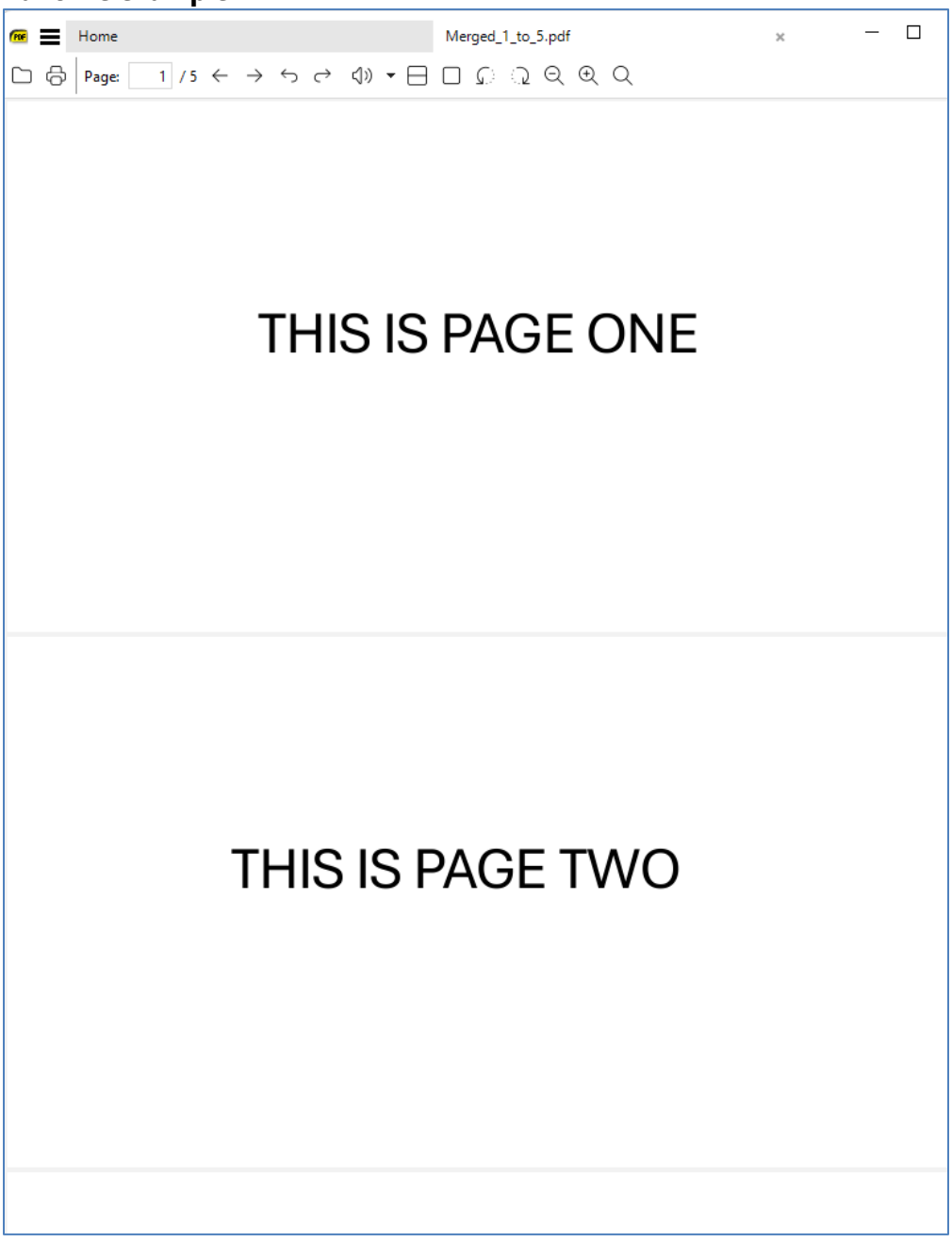
CASE Loc:MergeResult
OF 0
    MESSAGE('Created ' & CLIP(Loc:MergeOutput), 'Merge PDFs')
OF -6
    MESSAGE('Output already exists. Rename it before running again.', 'Merge PDFs')
ELSE
    IF Loc:MergeFailed > 0
        GET(MergePDFQ, Loc:MergeFailed)
        IF ERRORCODE()
            MESSAGE('Input #' & Loc:MergeFailed & ' failed. DLL returned ' & |
                    Loc:MergeResult, 'Merge PDFs')
        ELSE
            MESSAGE('Could not merge ' & CLIP(MergePDF:Name) & |
                    '. DLL returned ' & Loc:MergeResult, 'Merge PDFs')
        END
    ELSE
```

```
            MESSAGE('PT_MergePDFs returned ' & Loc:MergeResult, 'Merge PDFs')
        END
    END
    FREE(MergePDFQ)
```

Expected result

The result and failing position are 0. The new PDF contains five pages in order.

Runtime example



The merged PDF opens with page one followed by page two.

3.3.4. PT_RotatePDFPage

Rotates one selected page and saves a new copy of the entire PDF.

Global MAP declaration

```
PT_RotatePDFPage(*CSTRING,LONG,LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_RotatePDFPage@16')
```

Parameters

PdfC - Full PDF path in a UTF-8 CSTRING.
Page - One-based page number, passed by value.
Degrees - Relative angle. Use 90, 180, or 270 clockwise, or the corresponding negative value.
OutputC - Full path for the new file.

Returns

0 - Success.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Source has no pages or selected page is out of range.
-4 - Page could not be loaded or rotated.
-5 - Output could not be written, verified, or published.
-6 - Destination already exists.

Local Data

```
Loc:PdfC CSTRING(512)
Loc:OutputC CSTRING(512)
Loc:Result LONG
```

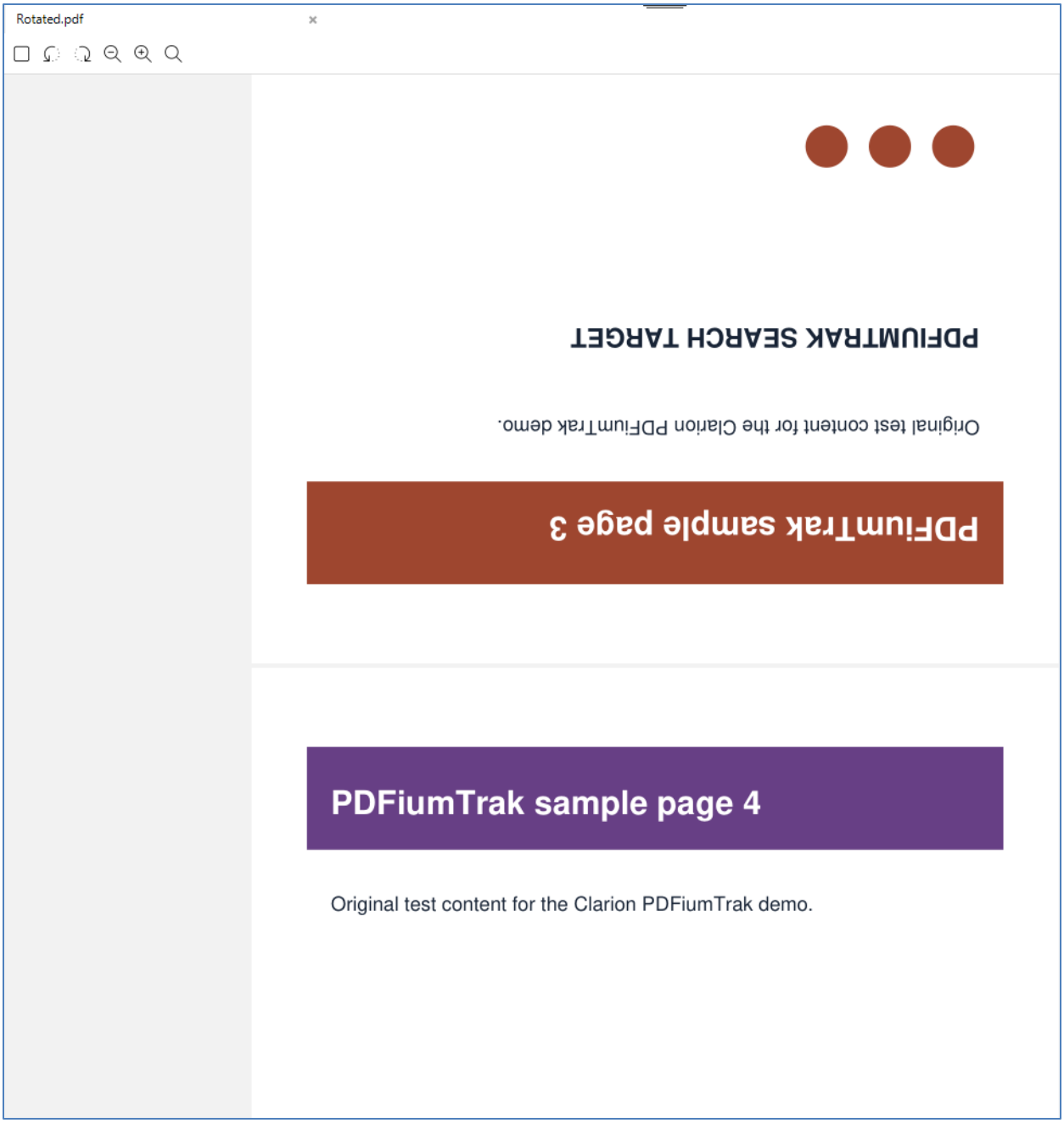
Button Accepted embed

```
Loc:PdfC = 'D:\ClarionApps\PDFium\Good.pdf'
Loc:OutputC = 'D:\ClarionApps\PDFium\Good_Page3_Rotated.pdf'
Loc:Result = PT_RotatePDFPage(Loc:PdfC, 3, 180, Loc:OutputC)
MESSAGE('PT_RotatePDFPage returned ' & Loc:Result)
```

Expected result

The result is 0. Page 3 is upside down and the other pages stay upright.

Runtime example



The saved rotation is visible when the PDF is reopened.

3.3.5. PT_RotatePDFAllPages

Rotates every page and saves a new copy of the entire PDF.

Global MAP declaration

```
PT_RotatePDFAllPages(*CSTRING,LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_RotatePDFAllPages@12')
```

Parameters

PdfC - Full PDF path in a UTF-8 CSTRING.

Degrees - Relative angle. Use 90, 180, or 270 clockwise, or the corresponding negative value.
OutputC - Full path for the new file.

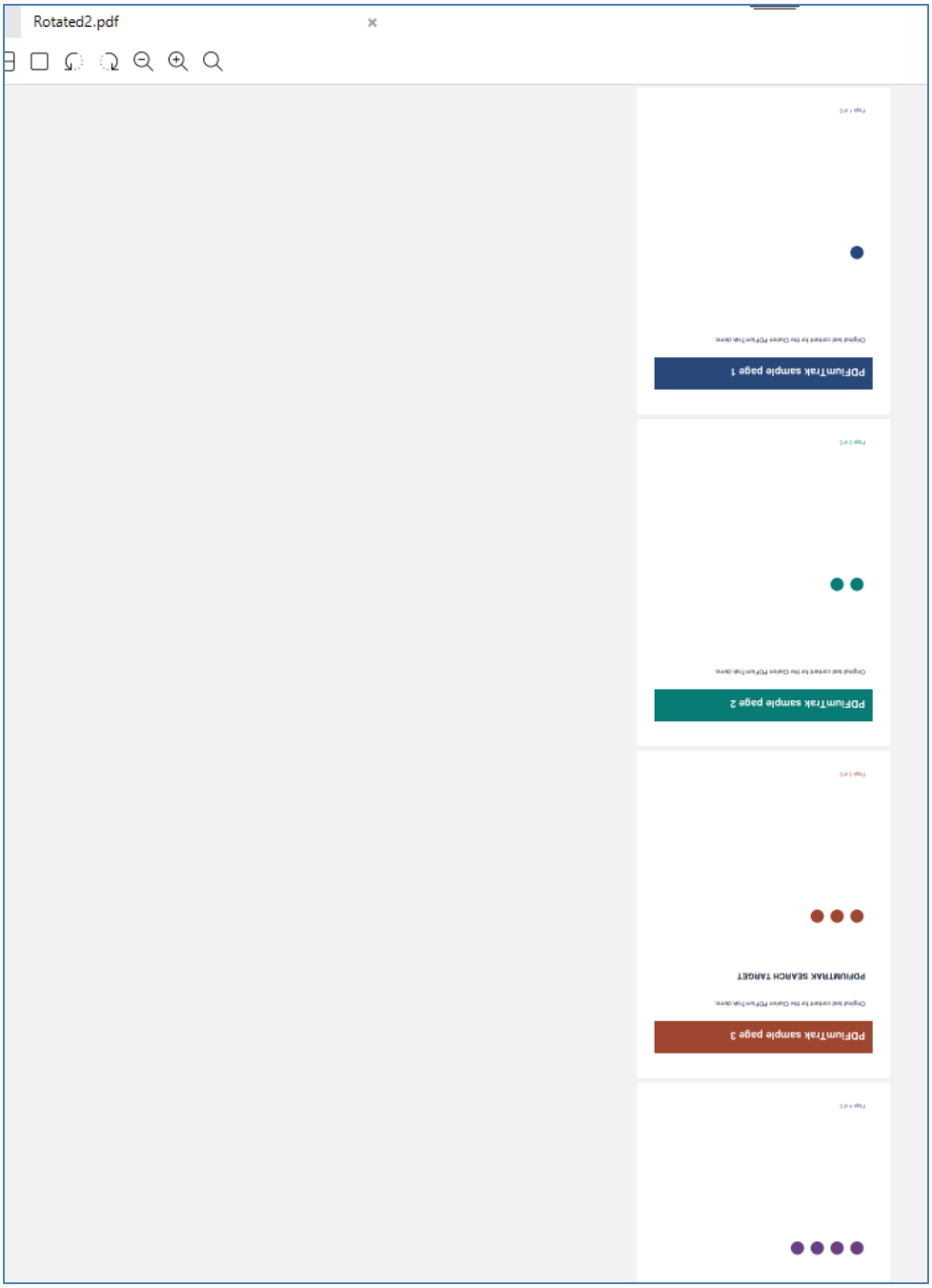
Returns
0 - Success.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Source has no pages or selected page is out of range.
-4 - Page could not be loaded or rotated.
-5 - Output could not be written, verified, or published.
-6 - Destination already exists.

Local Data
```
Loc:PdfC CSTRING(512)
Loc:OutputC CSTRING(512)
Loc:Result LONG
```

Button Accepted embed
```
Loc:PdfC = 'D:\ClarionApps\PDFium\Good.pdf'
Loc:OutputC = 'D:\ClarionApps\PDFium\Good_All_Rotated.pdf'
Loc:Result = PT_RotatePDFAllPages(Loc:PdfC, 180, Loc:OutputC)
MESSAGE('PT_RotatePDFAllPages returned ' & Loc:Result)
```

Expected result
The result is 0. All five pages are upside down in the saved copy.

Runtime example



The saved rotation is visible when the PDF is reopened.

3.4. Export PDF Pages to JPEG

Select a function below for its declaration, return values, and Clarion example.
PT_ExtractJPG
PT_ExtractTest

3.4.1. PT_ExtractJPG

Renders one PDF page to a JPEG at a chosen width and quality.

Global MAP declaration
```
PT_ExtractJPG(*CSTRING,*LONG,*CSTRING,*LONG,*LONG),LONG,RAW,PASCAL,NAME('_PT_ExtractJPG@20')
```

Parameters
PdfC - Full PDF path in a UTF-8 CSTRING.
Page, Width, Quality - LONG variables passed by reference. Page numbers start at 1.
OutputC - Full path for the new file.
Width is measured in pixels. Quality ranges from 1 to 100.

Returns
0 - Success.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Page is outside the source PDF.
-4 - Page could not be loaded.
-5 - Could not render the page.
-6 - JPEG encoding or output failed.
-7 - COM or WIC initialization failed.

Local Data
```
Loc:PdfC CSTRING(512)
Loc:OutputC CSTRING(512)
Loc:Page LONG
Loc:Width LONG
Loc:Quality LONG
Loc:Result LONG
Loc:PT_Retval LONG
Loc:PageNumber LONG
Loc:PathGoodPDF CSTRING(255)
Loc:CurrentPath STRING(255)
```

Button Accepted embed
```
! Generate PDF for the selected page, using the page number within the filename
!Loc:PathGoodPDF is loaded within the window opening embed

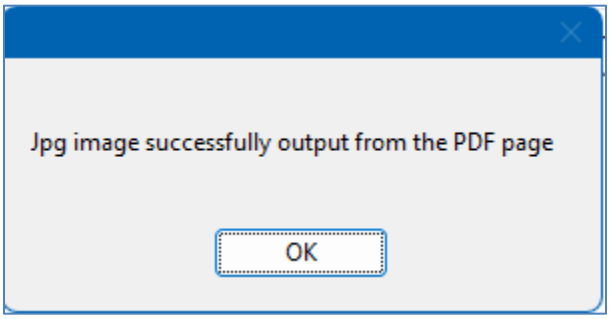
Loc:Currentpath = 'D:\Clarionapps\PDFium'
Loc:PageNumber = 1
```

```
 Loc:PT_Retval = PT_CheckPDF(Loc:PathGoodPDF)

! If the PDF is good, proceed
 IF Loc:PT_Retval = 0
   Loc:OutputC = clip(Loc:CurrentPath) & '\Good_Page' &  Loc:PageNumber & '.jpg'
   Loc:Width = 600
   Loc:Quality = 85
   Loc:Result = PT_ExtractJPG(Loc:PathGoodPDF, Loc:PageNumber, Loc:OutputC, Loc:Width, Loc:Quality)

   ! PT_ExtractJPG will overwrite an existing file of the same name
   IF Loc:Result = 0
    message('Jpg image successfully output from the PDF page')
   ELSE
    message('Error generating the .Jpg from the selected PDF page')
   END
 END

 ! Clear the spinbox control
 Loc:PageNumber = 0
 display()
```



3.4.2. PT_ExtractTest

Renders one PDF page to JPEG using a single pipe-delimited command.

Global MAP declaration

```
PT_ExtractTest(*CSTRING),LONG,RAW,PASCAL,NAME('_PT_ExtractTest@4')
```

Parameter

CommandC - PDF path, page number, JPEG path, width, and quality in that order. Separate the fields with pipes.

Returns

0 - Success.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Page is outside the source PDF.
-4 - Page could not be loaded.
-5 - Could not render the page.
-6 - JPEG encoding or output failed.
-7 - COM or WIC initialization failed.

Local Data

```
Loc:CommandC CSTRING(2048)
Loc:Result LONG
Loc:PT_Retval LONG
Loc:PDFPathC CSTRING(512)
Loc:ExtractResult LONG
Loc:PathGoodPDF CSTRING(255)
```

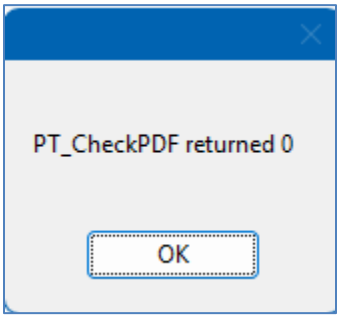
Button Accepted embed

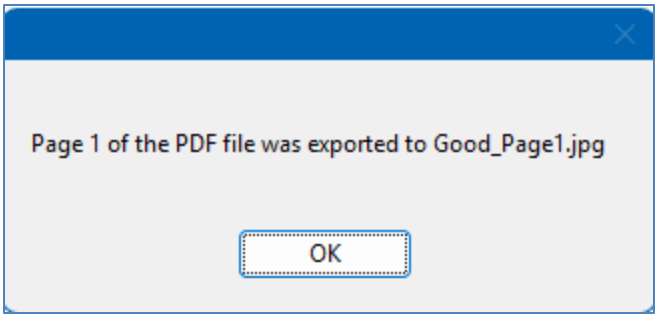
```
!Loc:PathGoodPDF is loaded within the window opening embed
Loc:PT_Retval = PT_CheckPDF(Loc:PathGoodPDF)
MESSAGE('PT_CheckPDF returned ' & Loc:PT_Retval)

! If PDF was good, proceed
IF Loc:PT_Retval = 0
   Loc:PDFPathC = CLIP(Loc:PathGoodPDF) & '|1|D:\ClarionApps\PDFium\Good_Page1.jpg|600|85'
   Loc:ExtractResult = PT_ExtractTest(Loc:PDFPathC)
   if Loc:ExtractResult = 0
      message('Page 1 of the PDF file was exported to Good_Page1.jpg')
   end
END
```

Expected result

PT_CheckPDF() reports 0 (the PDF file isn't corrupt) and PT_ExtractTest() successfully exported the desired page number to a .Jpg image file





3.4.3. Choosing JPEG Size and Quality

PT_ExtractJPG and PT_ExtractTest use two settings to control the generated JPEG:

- Width specifies the output width in pixels. The height is calculated from the PDF page's proportions.
- Quality specifies JPEG compression quality, from 1 to 100. Higher values generally preserve more detail and produce larger files.

Increasing quality preserves detail within the selected dimensions. Increasing width provides more pixels for displaying small text and fine detail.

Suggested starting settings

Intended use	Width in pixels	Quality
Small page thumbnail	150–300	60–75
Document preview in a list or panel	400–600	75–85
General onscreen page viewing	1,000–1,600	85–90
Enlarged viewing of text or diagrams	1,800–2,500	90–95
Detailed export or printing	Calculate from page size and desired DPI	90–95

These settings are starting points. Review the exported image using representative documents, particularly pages containing small text, photographs, or detailed diagrams.

Dimensions and page proportions

For a portrait US Letter page measuring 8.5 × 11 inches, approximate output dimensions are as follows.

Requested width	Calculated height	Effective resolution
600 pixels	776 pixels	71 DPI
1,275 pixels	1,650 pixels	150 DPI
1,700 pixels	2,200 pixels	200 DPI
2,550 pixels	3,300 pixels	300 DPI

Other page sizes and orientations produce different heights. A landscape page exported at the same width has fewer pixels vertically.

To calculate a width for printing, multiply the page's width in inches by the desired dots per inch (DPI). For example, an 8.5-inch-wide page at 300 DPI requires a width of 2,550 pixels.

These calculations describe the effective resolution when printed at that physical size. The function's width argument controls pixel dimensions.

Choosing JPEG quality
Quality 85 is a practical starting point for general previews. Try 90–95 when compression artifacts affect text or fine detail. Quality 100 usually produces larger files, and JPEG compression can still alter the image.

Actual file size depends on page content, dimensions, and quality. A page containing a photograph may produce a much larger JPEG than a mostly white page using identical settings.

Memory considerations
Larger dimensions increase rendering memory requirements. Doubling the width also approximately doubles the height, producing four times as many pixels.

A 2,550 × 3,300 image requires approximately 32 MiB for a four-byte-per-pixel render buffer. PDF processing and JPEG encoding require additional memory. This matters in a 32-bit Clarion application.

Example settings
The existing examples use a width of 600 pixels and quality 85. This produces a compact preview. For more readable small text, try a width of 1,275 or 1,700 pixels with quality 90.

Both functions use the same width and quality settings. PT_ExtractJPG receives them through LONG variables passed by reference. PT_ExtractTest receives them as the final two fields of its pipe-delimited command string.

3.5. Metadata, Text, and Bookmarks

Select a function below for its declaration, return values, and Clarion example.
PT_GetPDFMetadata
PT_SearchPDFPage
PT_FindPDFText
PT_GetPDFPageText
PT_GetPDFBookmarks

3.5.1. PT_GetPDFMetadata

Reads one standard document-information field as UTF-8.

Global MAP declaration

```
PT_GetPDFMetadata(*CSTRING,*CSTRING,*CSTRING,LONG),LONG,RAW,PASCAL,NAME('_PT_GetPDFMetadata@16')
```

Parameters

PdfC - Full PDF path in a UTF-8 CSTRING.
TagC - Title, Author, Subject, Keywords, Creator, Producer, CreationDate, or ModDate.
ValueC and capacity - Caller buffer and its byte size including the terminating NUL.

Returns

0 - Success.
1 - Field is missing or empty.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Buffer too small.
-4 - Metadata read or conversion failed.

Local Data

```
Loc:PdfC CSTRING(512)
Loc:TagC CSTRING(32)
Loc:ValueC CSTRING(2048)
Loc:PathGoodPDF CSTRING(255)
Loc:Result LONG
Loc:MetaTag1 CSTRING(32)
Loc:MetaTag2 CSTRING(32)
Loc:MetaTag3 CSTRING(32)
Loc:MetaTag4 CSTRING(32)
Loc:MetaTag5 CSTRING(32)
Loc:MetaTag6 CSTRING(32)
Loc:MetaTag7 CSTRING(32)
Loc:MetaTag8 CSTRING(32)
Loc:MetaValue CSTRING(2048)
Loc:MetaIndex LONG
Loc:MetaDataMessage STRING(2028)
Loc:MetaTag  CSTRING(32)
Loc:MetaResult LONG
Loc:PDFMetaDataMessage STRING(2048)
```

Button Accepted embed

```
! Retrieve meta data from Good.Pdf
!Loc:PathGoodPDF is loaded within the window opening embed
Loc:MetaTag1 = 'Title'
Loc:MetaTag2 = 'Author'
Loc:MetaTag3 = 'Subject'
Loc:MetaTag4 = 'Keywords'
Loc:MetaTag5 = 'Creator'
Loc:MetaTag6 = 'Producer'
Loc:MetaTag7 = 'CreationDate'
Loc:MetaTag8 = 'ModDate'
Loc:PDFMetaDataMessage = ''
```

```
LOOP Loc:MetaIndex = 1 TO 8
  CASE Loc:MetaIndex
  OF 1
     Loc:MetaTag = Loc:MetaTag1
  OF 2
      Loc:MetaTag = Loc:MetaTag2
  OF 3
     Loc:MetaTag = Loc:MetaTag3
  OF 4
     Loc:MetaTag = Loc:MetaTag4
  OF 5
     Loc:MetaTag = Loc:MetaTag5
  OF 6
     Loc:MetaTag = Loc:MetaTag6
  OF 7
     Loc:MetaTag = Loc:MetaTag7
  OF 8
     Loc:MetaTag = Loc:MetaTag8
  END

  Loc:MetaResult = PT_GetPDFMetadata(Loc:PathGoodPDF, Loc:MetaTag,Loc:MetaValue, SIZE(Loc:MetaValue))

  CASE Loc:MetaResult
  OF 0
     IF LEN(CLIP(Loc:MetaValue)) > 220
      Loc:MetaValue = '(value too long for this popup)'
     END
  OF 1
      Loc:MetaValue = '(not set)'
  ELSE
       Loc:MetaValue = '(DLL returned ' & Loc:MetaResult & ')'
  END

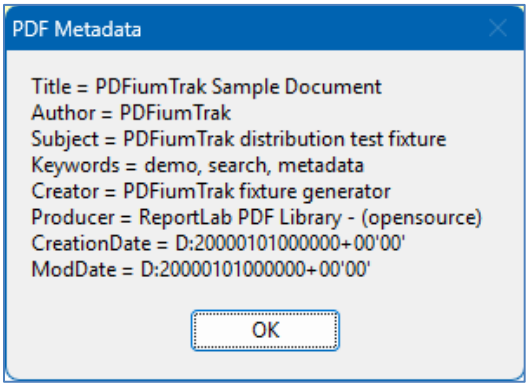
  IF Loc:MetaIndex > 1
     ! This code prettifies the message box string
      Loc:PDFMetaDataMessage = CLIP(Loc:PDFMetaDataMessage) & '|'
  END

  ! Build the message box string
  Loc:PDFMetaDataMessage = CLIP(Loc:PDFMetaDataMessage) &  CLIP(Loc:MetaTag) & ' = ' & CLIP(Loc:MetaValue)
```

```
END

! Display all metadata within the PDF
MESSAGE(Loc:PDFMetaDataMessage, 'PDF Metadata')
```

Expected result
A popup message which lists all of the available PDF metadata properties



3.5.2. PT_SearchPDFPage

Counts case-insensitive substring matches on a specified page.

Global MAP declaration
```
PT_SearchPDFPage(*CSTRING,LONG,*CSTRING),LONG,RAW,PASCAL,NAME('_PT_SearchPDFPage@12')
```

Parameters
PdfC - Full PDF path in a UTF-8 CSTRING.
Page - One-based page number, passed by value.
QueryC - Search text in a UTF-8 CSTRING.

Returns
> 0 - Match count.
0 - No matches.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Page is outside the source PDF.
-4 - Page could not be loaded.
-6 - Text loading or searching failed.
-5 - No extractable text.

Local Data
```
Loc:PdfC CSTRING(512)
Loc:QueryC CSTRING(128)
Loc:Result LONG
```

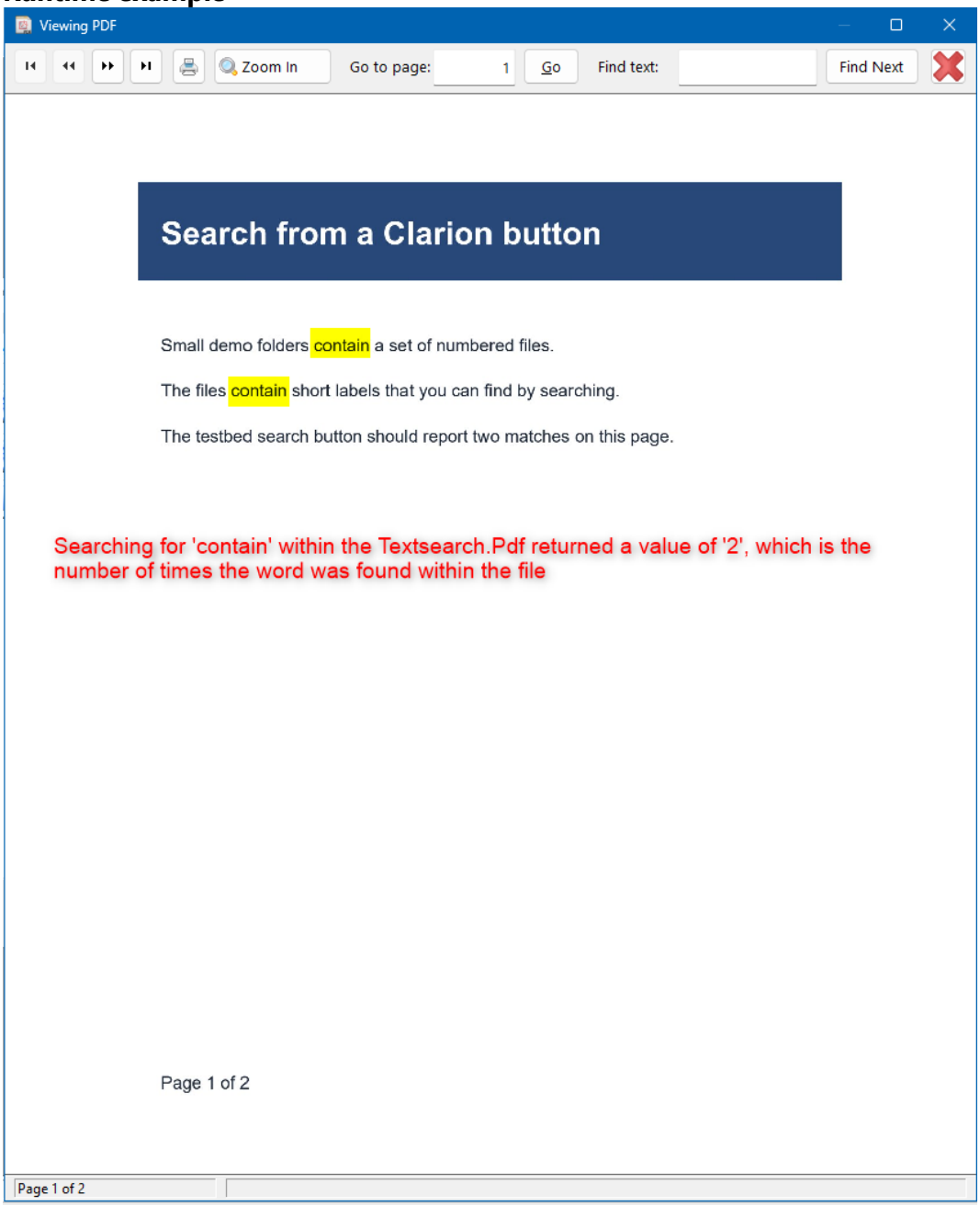
Button Accepted embed

```
Loc:PdfC = 'D:\ClarionApps\PDFium\Textsearch.pdf'
Loc:QueryC = 'contain'
Loc:Result = PT_SearchPDFPage(Loc:PdfC, 1, Loc:QueryC)
MESSAGE('PT_SearchPDFPage returned ' & Loc:Result)
```

Expected result

The result is 2 for page 1.

Runtime example



The annotations mark the two matches on page 1. The function counts matches on the requested page.

3.5.3. PT_FindPDFText

Searches forward from the starting page, including it, and stops at the first matching page. Pages without extractable text are skipped.

Global MAP declaration

```
PT_FindPDFText(*CSTRING,LONG,*CSTRING,*LONG),LONG,RAW,PASCAL,NAME('_PT_FindPDFText@16')
```

Parameters

PdfC - Full PDF path in a UTF-8 CSTRING.
StartPage - First page to search, starting at 1.
QueryC - UTF-8 search text.
Matches - LONG output with the match count on the returned page, or zero.

Returns

> 0 - First matching page number.
0 - No match through the end.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Starting page is beyond the end.
-4 - Page could not be loaded.
-6 - Text loading or searching failed.

Local Data

```
Loc:PdfC CSTRING(512)
Loc:QueryC CSTRING(128)
Loc:Matches LONG
Loc:Found LONG
```

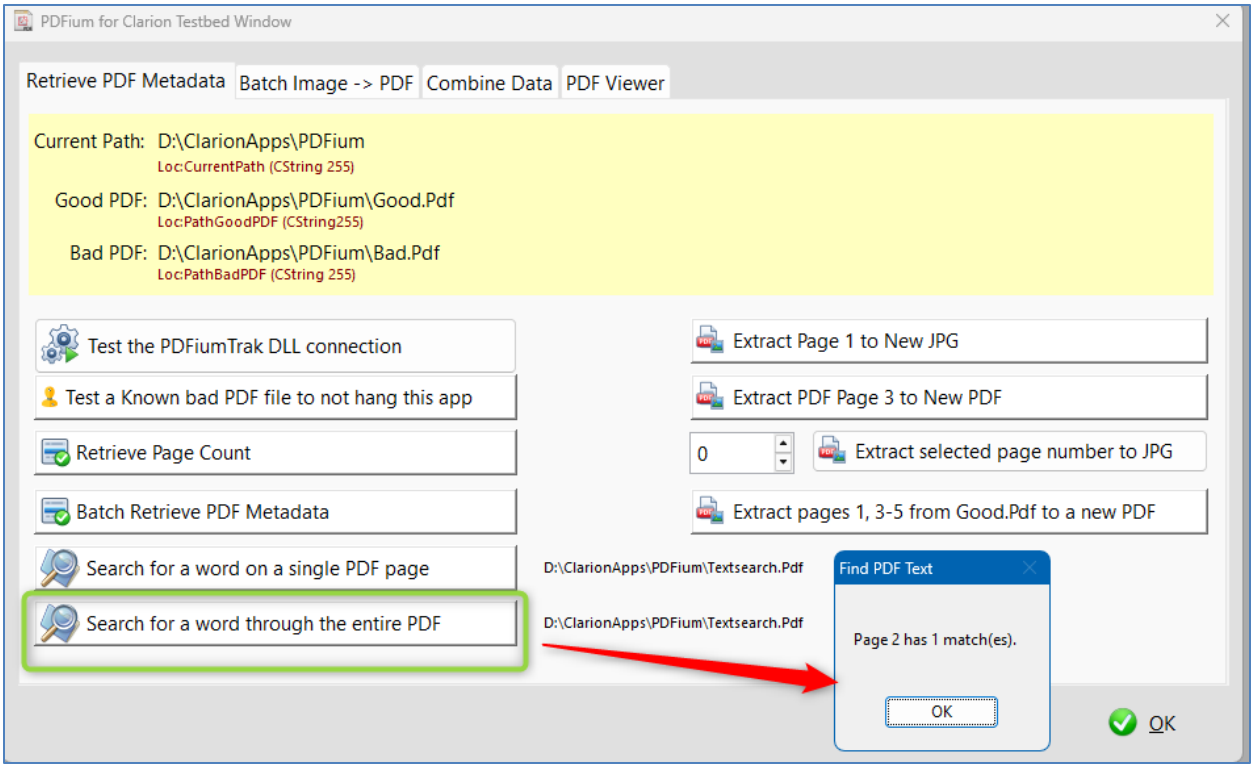
Button Accepted embed

```
Loc:PdfC = 'D:\ClarionApps\PDFium\Textsearch.pdf'
Loc:QueryC = 'orbit'
Loc:Found = PT_FindPDFText(Loc:PdfC, 1, Loc:QueryC, Loc:Matches)
IF Loc:Found > 0
 MESSAGE('Page ' & Loc:Found & ' has ' & Loc:Matches & ' match(es).')
ELSE
 MESSAGE('Search result ' & Loc:Found)
END
```

Expected result

The popup reports page 2 with one match.

Runtime example



The popup reports page 2 with one match.

3.5.4. PT_GetPDFPageText

Extracts the existing text layer of one PDF page into a Clarion CSTRING.

Global MAP declaration

```
PT_GetPDFPageText(*CSTRING,LONG,*CSTRING,LONG),LONG,RAW,PASCAL,NAME('_PT_GetPDFPageText@16')
```

Parameters

PdfC - Full PDF path in a UTF-8 CSTRING.
Page - One-based page number, passed by value.
TextC and capacity - Caller buffer and its byte size including the NUL.

Returns

> 0 - UTF-8 byte count excluding the NUL.
0 - No extractable text.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Page is outside the source PDF.
-4 - Page could not be loaded.
-6 - Extraction or conversion failed.
-7 - Buffer too small. No partial text is returned.

Local Data

```
Loc:PdfC CSTRING(512)
Loc:TextC CSTRING(8192)
Loc:Result LONG
```

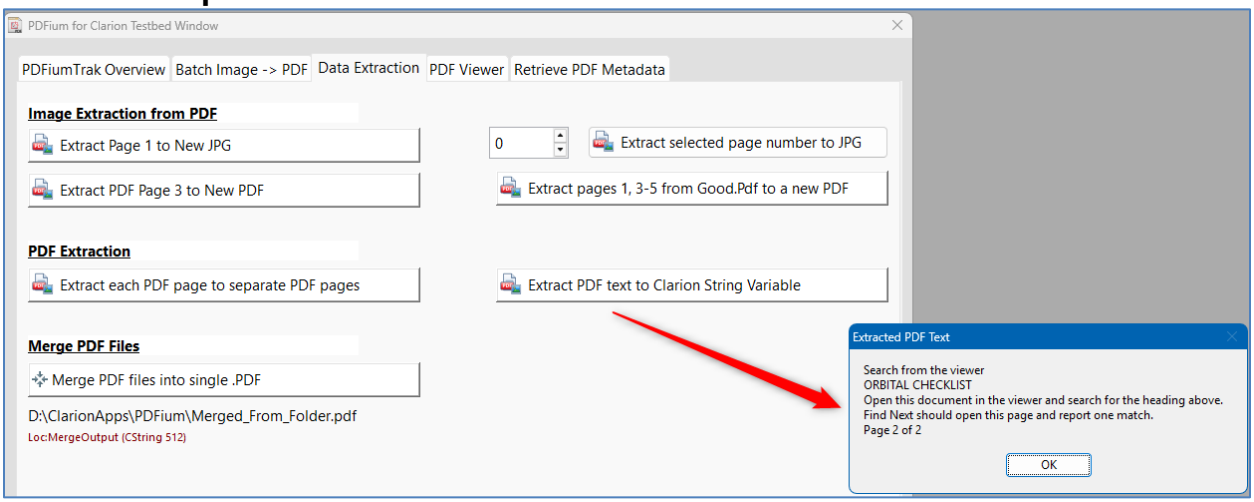
Button Accepted embed

```
Loc:PdfC = 'D:\ClarionApps\PDFium\Textsearch.pdf'
Loc:Result = PT_GetPDFPageText(Loc:PdfC, 2, Loc:TextC, SIZE(Loc:TextC))
IF Loc:Result > 0
 MESSAGE(Loc:TextC, 'Extracted PDF Text')
ELSE
 MESSAGE('PT_GetPDFPageText returned ' & Loc:Result)
END
```

Expected result

The popup shows ORBITAL CHECKLIST and the remaining text from page 2.

Runtime example



The popup shows ORBITAL CHECKLIST and the remaining text from page 2.

3.5.5. PT_GetPDFBookmarks

Lists bookmark titles and local page destinations in outline order.

Global MAP declaration

```
PT_GetPDFBookmarks(*CSTRING,*CSTRING,LONG,LONG),LONG,RAW,PASCAL,NAME('_PT_GetPDFBookmarks@16')
```

Parameters

PdfC - Full PDF path in a UTF-8 CSTRING.
RowsC and capacity - Output buffer and its byte size including the NUL.
Encoding - 0 for UTF-8 or 1 for Windows ANSI.
Rows use DD|PPPPPPPPPP|Title followed by CRLF. The first two pipes delimit depth and 1-based page.

Returns

> 0 - Number of bookmark rows.

0 - No bookmarks with destinations in this PDF.
-1 - Invalid argument.
-2 - Source PDF could not be opened.
-3 - Output buffer too small.
-4 - Outline traversal or encoding failed.

Local Data

```
Loc:PdfC CSTRING(512)
Loc:RowsC CSTRING(2048)
Loc:Result LONG
```

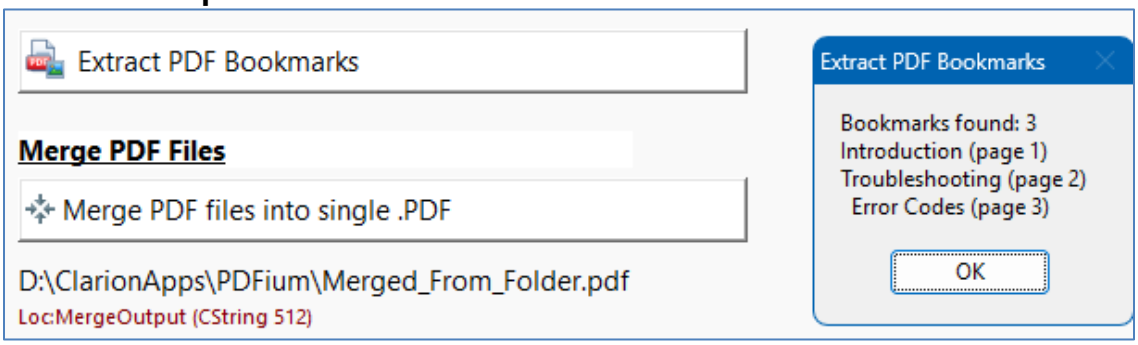
Button Accepted embed

```
Loc:PdfC = 'D:\ClarionApps\PDFium\Bookmarks.pdf'
Loc:Result = PT_GetPDFBookmarks(Loc:PdfC, Loc:RowsC, SIZE(Loc:RowsC), 1)
IF Loc:Result > 0
 MESSAGE('Bookmarks found: ' & Loc:Result)
ELSE
 MESSAGE('PT_GetPDFBookmarks returned ' & Loc:Result)
END
```

Expected result

The popup reports three bookmarks. Loc:RowsC holds their titles, depths, and page destinations.

Runtime example



The testbed's expanded popup displays the three decoded rows.

3.6. View and Print PDFs

Select a function below for its declaration, return values, and Clarion example.

PT_ViewPDF
PT_PrintPDF

3.6.1. PT_ViewPDF

Opens a PDF in a resizable Clarion window with page navigation, zoom, text searching, bookmarks, page rotation, and optional printing.

Global MAP declaration
Add this declaration inside the existing MODULE('PDFiumTrakViewer.clw') block in your Global MAP.

PT_ViewPDF(*CSTRING,<STRING>,BYTE=1)

The PDFiumTrakViewer.clw source module must also be included in your project. Follow the setup instructions in the PDFiumTrakViewer.clw help topic.

Parameters

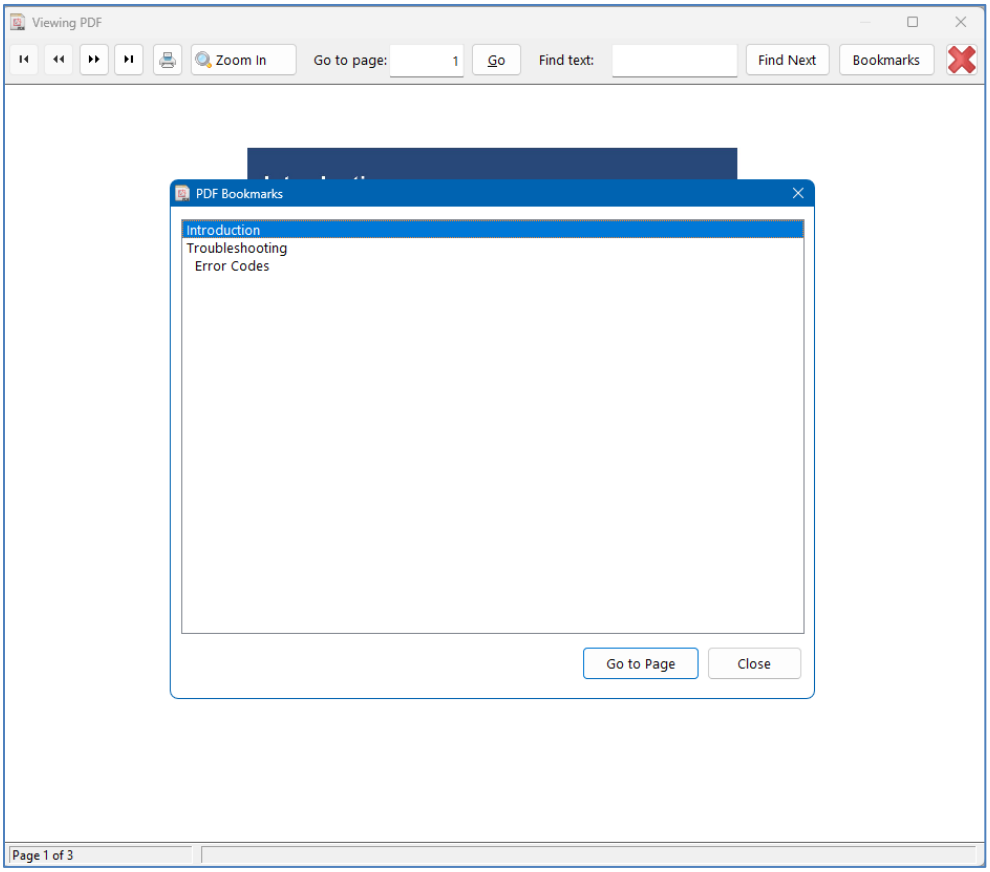
Parameter	Type	Description
InputPathC	*CSTRING	Required. Full path and filename of the PDF to open. Supply UTF-8 for paths containing non-ASCII characters.
ViewerTitle	STRING, optional	Window title. Defaults to “Viewing PDF” when omitted.
ShowPrint	BYTE, optional	Controls whether the Print button appears. Defaults to 1. Pass 0 to hide it.

Returns
This Clarion procedure has no return value. The viewer displays a message if it cannot open the PDF or render a requested page.

Local Data
Loc:PdfC CSTRING(512)

Button Accepted embed
```
Loc:PdfC = 'D:\ClarionApps\PDFium\Bookmarks.pdf'
PT_ViewPDF(Loc:PdfC, 'Bookmark Demo', 0)
```

Expected result
The viewer opens Bookmarks.pdf with the title “Bookmark Demo.” The Print button is hidden. Click Bookmarks, select Error Codes, and click Go to Page to display page 3.



Other calling examples

Open a PDF using the default title and display the Print button.
PT_ViewPDF(Loc:PdfC)

Specify a title and display the Print button.
PT_ViewPDF(Loc:PdfC, 'Customer Invoice', 1)

Specify a title and hide the Print button.
PT_ViewPDF(Loc:PdfC, 'Customer Invoice', 0)

To display a PDF whose filename is stored in a database field, assign that field's value to the filename variable before calling PT_ViewPDF.

Using the viewer

- Page navigation — Use First, Previous, Next, and Last, or enter a page number and click Go. Page numbers start at 1.
- Zoom — Click Zoom In to enlarge the page. Use the scrollbars or mouse wheel to move through it. Click Fit Page to display the complete page.
- Text search — Enter text and click Find Next. The viewer moves to a page containing a match and displays the match count in the status bar. Searching ignores case and uses the PDF's existing searchable text.
- Bookmarks — The Bookmarks button appears when the PDF contains bookmarks with destinations inside the document. Select a bookmark and click Go to Page to display its destination.

- Rotate 180 — Choose whether to rotate the current page or all pages. The viewer saves a new PDF and opens the saved copy at the same page number.
- Print — When enabled, opens the Windows print dialog for printer selection, page ranges, and copies.
- Close — Closes the viewer and returns control to the calling procedure.

Customizing the viewer icons

You can change the toolbar icons to match your application. Open the copy of PDFiumTrakViewer.clw used by your project and locate the BUTTON declarations within the viewer's WINDOW definition.

Each button's ICON(...) attribute specifies its icon. The source-code screenshot below shows the built-in Clarion icons used by the viewer:

```
ViewerWindow WINDOW('Viewing PDF'),AT(,,605,360),FONT('Segoe UI',10), |
                CENTER,RESIZE,MAX,IMM,SYSTEM,STATUS(100,-180)
              ! Replace any ICON: equate below with your own .ICO filename.
              TOOLBAR,AT(0,0,605,20),USE(?ViewerTools)
                BUTTON,AT(4,2,17,15),USE(?FirstPage),ICON(ICON:VCRtop), |
                  TIP('Go to the first PDF page')
                BUTTON,AT(23,2,17,15),USE(?PreviousPage),ICON(ICON:VCRrewind), |
                  TIP('Go to the previous PDF page')
                BUTTON,AT(42,2,17,15),USE(?NextPage),ICON(ICON:VCRfastforward), |
                  TIP('Go to the next PDF page')
                BUTTON,AT(61,2,17,15),USE(?LastPage),ICON(ICON:VCRbottom), |
                  TIP('Go to the last PDF page')
                BUTTON,AT(82,2,17,15),USE(?PrintPDF),ICON(ICON:Print),HIDE, |
                  TIP('Open the Windows print dialog')
                BUTTON('Zoom In'),AT(103,2,59,15),USE(?ZoomPage),LEFT, |
                  ICON(ICON:Zoom),TIP('Enlarge the page and enable scrolling')
                PROMPT('Go to page:'),AT(170,5,40,11),USE(?JumpPrompt)
                ENTRY(@N_10),AT(212,3,40,14),USE(JumpPage),RIGHT, |
                  TIP('Enter the PDF page number, then click Go')
                BUTTON('&Go'),AT(256,2,28,15),USE(?JumpGo), |
                  TIP('Display the entered page number')
                PROMPT('Find text:'),AT(293,5,38,11),USE(?FindPrompt)
                ENTRY(@S128),AT(333,3,68,14),USE(FindTextC), |
                  TIP('Enter text to search for in the PDF. Case is ignored.')
                BUTTON('Find Next'),AT(405,2,47,15),USE(?FindPage), |
                  TIP('Jump to the next page containing the search text')
                BUTTON('Bookmarks'),AT(456,2,53,15),USE(?Contents),HIDE, |
                  TIP('Open the list of PDF bookmarks')
                BUTTON('Rotate 180'),AT(513,2,64,15),USE(?RotatePDF), |
                  TIP('Rotate the current page or all pages and save a new PDF')
                BUTTON,AT(582,2,18,15),USE(?CloseViewer),ICON(ICON:Cross), |
                  TIP('Close the PDF viewer')
```

Replace the value inside ICON(...) with another Clarion icon equate or the quoted filename of your preferred .ICO file. For example, ICON('PreviousPage.ico') selects a custom icon. Make that file available through your project's resource search path when compiling.

Preserve the button's USE(...) identifier so its existing event handling continues to work. You can also change its TIP(...) text to customize the tooltip displayed when the mouse rests over the button.

Save PDFiumTrakViewer.clw and rebuild your Clarion application to apply the changes.

3.6.2. PT_PrintPDF

Opens the Windows print dialog and submits the selected PDF pages.

Global MAP declaration

```
PT_PrintPDF(*CSTRING,ULONG),LONG,RAW,PASCAL,NAME('_PT_PrintPDF@8')
```

Parameters

PdfC - Full PDF path in a UTF-8 CSTRING.
Owner - Handle of an open Clarion window, passed as ULONG.

Returns

0 - Job submitted.
1 - Dialog canceled.
-1 - Invalid path or owner.
-2 - PDF open or page count failed.
-3 - Print dialog or printer setup failed.
-4 - Invalid page ranges.
-5 - Page load or rendering failed.
-6 - Print job failed.

Local Data

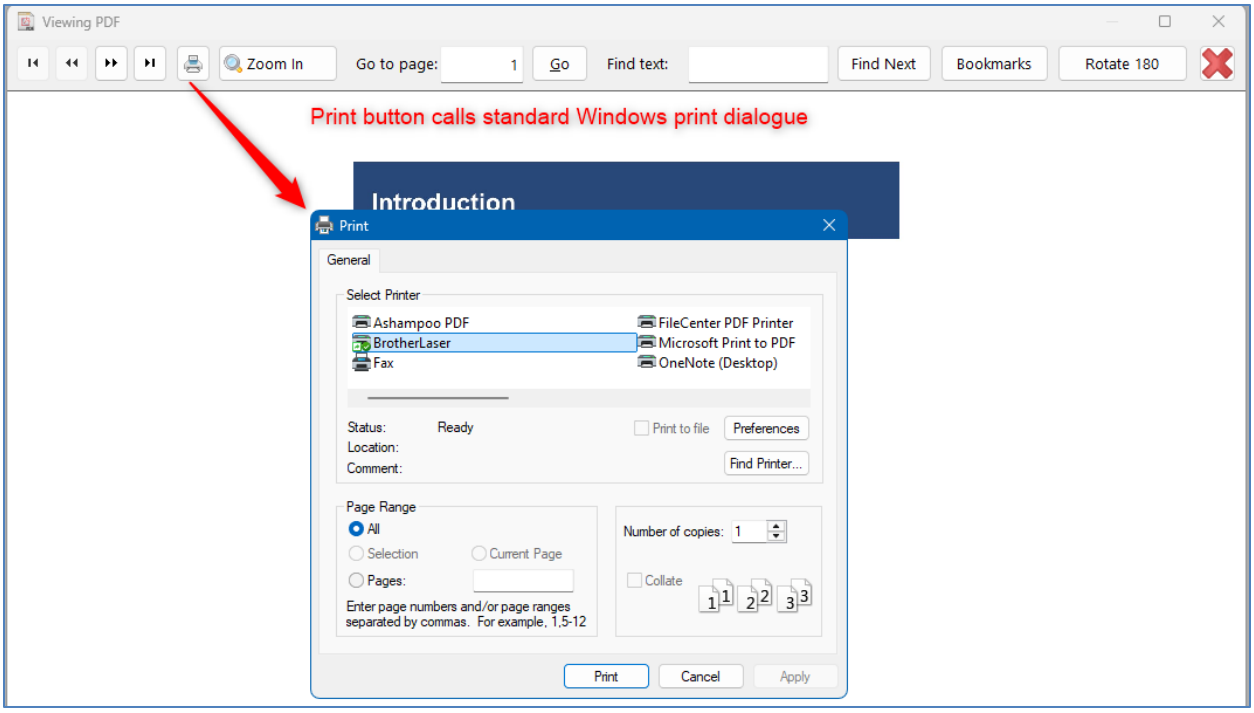
```
Loc:PdfC CSTRING(512)
Loc:Owner ULONG
Loc:Result LONG
```

Button Accepted embed

```
Loc:PdfC = 'D:\ClarionApps\PDFium\Good.pdf'
! Replace Window with the label of your open Clarion window.
Loc:Owner = Window{PROP:Handle}
Loc:Result = PT_PrintPDF(Loc:PdfC, Loc:Owner)
MESSAGE('PT_PrintPDF returned ' & Loc:Result)
```

Expected result

The Windows print dialog opens. Choose Cancel to verify a return of 1 without printing.



3.7. PDFiumTrak Return Codes

Interpret the return value according to the function you called. For example, 0 means success for PT_SavePDFPages, no matches for PT_FindPDFText, and "open and first-page check passed" for PT_CheckPDF. Page numbers and input positions start at 1.

PT_AbortImagePDF

- 0 — The job was cleaned up or was already absent.
- -4 — Cleanup failed.

PT_AddImageFile

- 0 — The image was added. A supported multipage TIFF can add more than one page.
- -1 — Invalid image path argument.
- -2 — The job ID is invalid or the job has already failed.
- -3 — The image could not be read or its format is unsupported.
- -4 — Image processing or writing to the PDF failed.
- -7 — An image, page count, or output size limit was reached.

PT_AddJPGPage

- 0 — The JPEG was added as one page.
- -1 — Invalid JPEG path argument.
- -2 — The job ID is invalid or the job has already failed.
- -3 — The JPEG could not be read or its format is unsupported.
- -4 — Writing to the PDF failed.
- -7 — A page count or output size limit was reached.

PT_BeginImagePDF

- 0 — The job started. The output job ID is ready to use.
- -1 — Invalid output path or job ID argument.
- -4 — The job or its temporary output could not be created.
- -6 — The destination already exists.
- -7 — No new job ID was available.

PT_CheckPDF

- 0 — PDFium opened the file and loaded page 1.
- 1 — The check failed. Skip this file.

This check runs inside your application without a timeout. A successful result covers opening the file and loading its first page.

PT_ExtractJPG

- 0 — The JPEG was saved.
- -1 — Invalid argument.
- -2 — The source PDF could not be opened.
- -3 — The requested page is beyond the end of the PDF.
- -4 — The requested page could not be loaded.
- -5 — The page could not be rendered at the requested size.
- -6 — JPEG encoding or output failed.
- -7 — COM or Windows Imaging Component initialization failed.

PT_ExtractTest

- 0 — The JPEG was saved.
- -1 — Invalid command string or argument.
- -2 — The source PDF could not be opened.
- -3 — The requested page is beyond the end of the PDF.
- -4 — The requested page could not be loaded.
- -5 — The page could not be rendered at the requested size.
- -6 — JPEG encoding or output failed.
- -7 — COM or Windows Imaging Component initialization failed.

PT_FindPDFText

- Positive value — The first matching page number. The match-count output variable holds the number of matches on that page.
- 0 — No match from the starting page through the end of the PDF.
- -1 — Invalid argument, such as a blank query or a starting page below 1.
- -2 — The PDF could not be opened.
- -3 — The starting page is beyond the end of the PDF.
- -4 — A page could not be loaded.
- -6 — Text could not be loaded or searched.

Pages without extractable text are skipped. The match count is zero whenever the function returns no match or an error.

PT_FinishImagePDF

- 0 — The PDF was saved and verified, and the job was released.
- -1 — The job has no pages.
- -2 — The job ID is invalid or the job has already failed.
- -4 — Writing or closing the output failed.
- -5 — The saved PDF failed verification.
- -6 — The finished PDF could not be moved to the destination, for example because a file already exists there.

PT_GetPageCount

- Zero or greater — Number of pages in the PDF.
- -1 — Invalid PDF path argument.
- -2 — The PDF could not be opened or the operation failed.

PT_GetPDFBookmarks

- Positive value — Number of bookmark rows returned.
- 0 — No bookmarks point to pages in this PDF.
- -1 — Invalid argument.
- -2 — The PDF could not be opened.
- -3 — The output buffer is too small.
- -4 — The bookmark outline could not be read or encoded.

PT_GetPDFMetadata

- 0 — The requested field was returned.
- 1 — The field is absent or empty.
- -1 — Invalid argument or unsupported metadata field.
- -2 — The PDF could not be opened.
- -3 — The output buffer is too small.
- -4 — The metadata could not be read or converted.

PT_GetPDFPageText

- Positive value — Number of UTF-8 bytes written, excluding the terminating NUL.
- 0 — The page has no extractable text.
- -1 — Invalid argument or page number below 1.
- -2 — The PDF could not be opened.
- -3 — The requested page is beyond the end of the PDF.
- -4 — The page could not be loaded.
- -6 — Text extraction or encoding failed.
- -7 — The output buffer is too small. The function returns an empty string.

PT_MergePDFs

- 0 — The merged PDF was saved and verified.

- -1 — Invalid argument or source list entry.
- -2 — A source PDF could not be opened.
- -3 — A source PDF has no pages.
- -4 — The merged document could not be created or an input could not be imported.
- -5 — The output could not be written, verified, or saved to its final location.
- -6 — The destination already exists.

The failed-input output variable holds the input's position in the source list when the error is specific to that input. It is zero on success or for a general error.

PT_PrintPDF

- 0 — The print job was submitted.
- 1 — The dialog closed without submitting a print job.
- -1 — Invalid PDF path or owner window.
- -2 — The PDF could not be opened or its page count was unavailable.
- -3 — The print dialog or printer setup failed.
- -4 — The selected page range is invalid.
- -5 — A page could not be loaded or rendered for printing.
- -6 — The print job failed.

PT_RotatePDFAllPages

- 0 — The rotated copy was saved.
- -1 — Invalid argument or unsupported rotation angle.
- -2 — The source PDF could not be opened.
- -3 — The source PDF has no pages.
- -4 — A page could not be loaded or rotated.
- -5 — The output could not be written, verified, or saved to its final location.
- -6 — The destination already exists.

PT_RotatePDFPage

- 0 — A copy of the entire PDF was saved with the requested page rotated.
- -1 — Invalid argument, unsupported rotation angle, or page number below 1.
- -2 — The source PDF could not be opened.
- -3 — The source PDF has no pages or the requested page is beyond the end.
- -4 — The page could not be loaded or rotated.
- -5 — The output could not be written, verified, or saved to its final location.
- -6 — The destination already exists.

PT_SavePDFPages

- 0 — The selected pages were saved to a new PDF.
- -1 — Invalid argument or page selection.
- -2 — The source PDF could not be opened.
- -3 — A requested page is beyond the end of the source PDF.
- -4 — The output document could not be created or the pages could not be imported.
- -5 — The output could not be written, verified, or saved to its final location.

- -6 — The destination already exists.

PT_SearchPDFPage

- Positive value — Number of matches on the requested page.
- 0 — No matches on that page.
- -1 — Invalid argument, such as a blank query or a page number below 1.
- -2 — The PDF could not be opened.
- -3 — The requested page is beyond the end of the PDF.
- -4 — The requested page could not be loaded.
- -5 — The page has no extractable text.
- -6 — Text could not be loaded or searched.

PT_SplitPDF

- Positive value — Number of one-page PDFs created.
- -1 — Invalid source path or output folder argument.
- -2 — The source PDF could not be opened.
- -3 — The source PDF has no pages.
- -4 — A one-page PDF could not be created or a page could not be imported.
- -5 — The output could not be created, written, verified, or saved to its final location.
- -6 — The destination already exists.

PT_TestConnection

- 12345 — Clarion successfully called the expected DLL export.

PT_ViewPDF

This Clarion procedure has no return value. The viewer displays a message if it cannot open the PDF or render a requested page.

4. PDFiumTrak Testbed Application

4.1. Overview

During the creation of PDFiumTrak, a Clarion application was created/utilized to develop each function. This testbed application serves several purposes:

1. Support continued development of PDFiumTrak features
2. Provide a visual demonstration of PDFiumTrak functionality as a compiled demo application
3. For registered PDFiumTrak users, the testbed source code is provided as a learning tool

APPLICATION & SUPPORT FILES
ClaASC.dll (Clarion 12 runtime file)
ClaDOS.dll (Clarion 12 runtime file)
ClaRUN.dll (Clarion 12 runtime file)
msvcp140.dll (Microsoft Visual C++ v14 Redistributable (x86))
pdfium.dll (Open source PDFium runtime file)
PDFiumTrak.dll (PDFiumTrak compiled runtime file)
PDFium.exe (Clarion 12 compiled executable)
vcruntime140.dll (Microsoft Visual C++ v14 Redistributable (x86))

Keep the listed DLLs beside PDFium.exe. Place the four sample PDFs in the same folder and keep the Bmp, Jpgs, Mergepdf, Png, and Tiff subfolders intact.

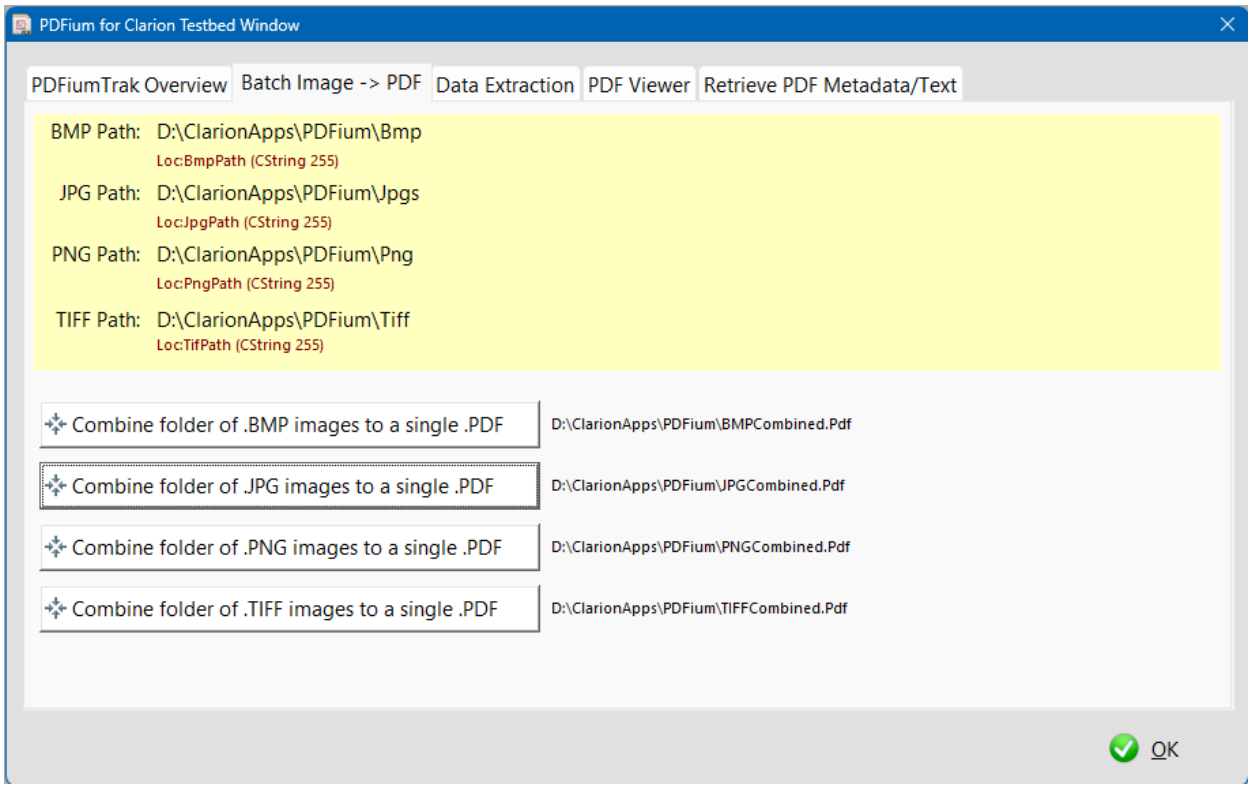
APPLICATION TEST FILES
Bad.pdf (known bad PDF sample file)
Bookmarks.pdf (Tests bookmark extraction and navigation in the PDF viewer.)
Good.pdf (known good PDF sample file)
Textsearch.pdf (used to test PDFiumTrak text search functionality)

FOLDERS THAT CONTAIN TESTBED TEST FILES
\Bmp (contains 5 sample image files)
\Jpgs (contains 5 sample image files)
\Mergepdf (contains 5 sample PDF files)
\Png (contains 5 sample image files)
\Tiff (contains 5 sample image files)

ADDITIONAL FOLDER CONTENT (OPTIONAL)
\PDFiumTrak Test Fixtures (contains a backup copy of the entire PDFiumTrak test file library)

4.2. Tab 1

A snapshot of this tab:



Tab 1 — Batch Image to PDF

Note: The buttons sort the input filenames and add them to the PDF in that order.

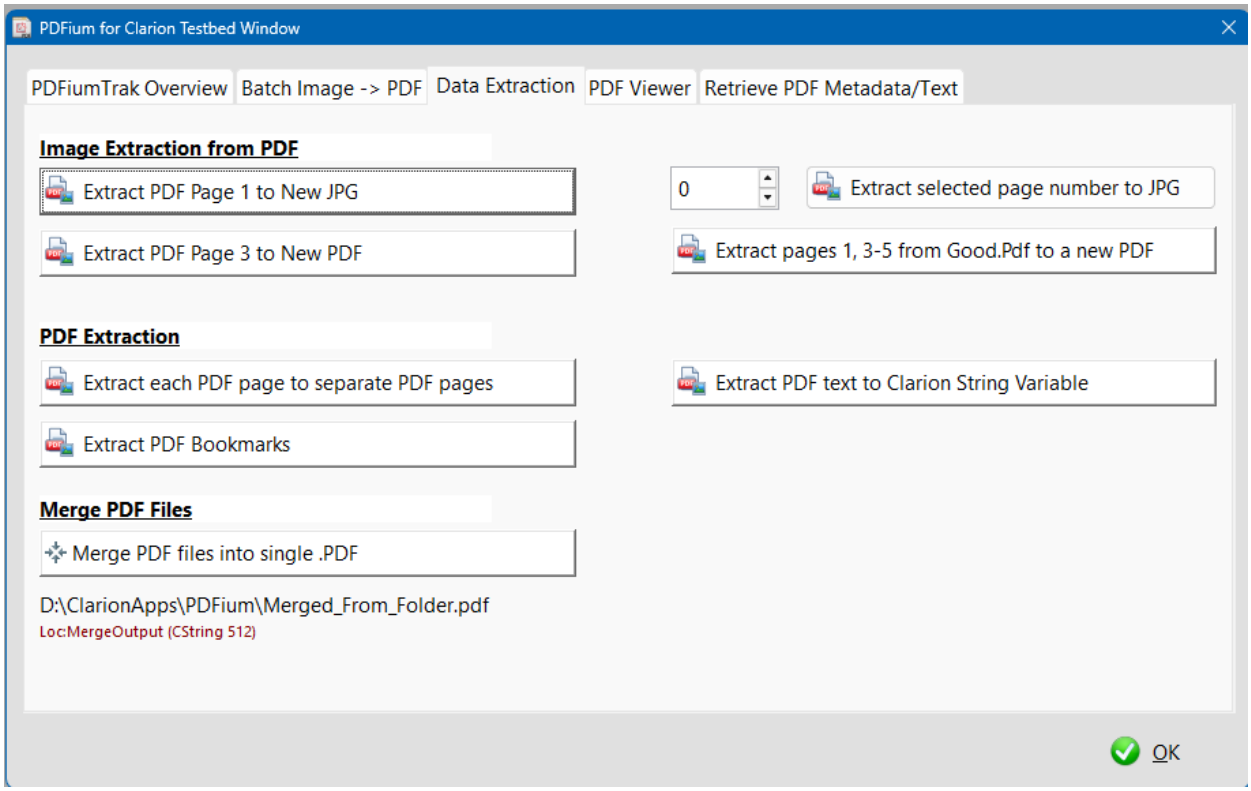
Button	PDFiumTrak calls
Combine folder of .BMP images to a single .PDF	PT_BeginImagePDF → PT_AddImageFile for each file → PT_FinishImagePDF → PT_GetPageCount to report the output page count.
Combine folder of .JPG images to a single .PDF	PT_BeginImagePDF → PT_AddJPGPage for each file → PT_FinishImagePDF. The button reports the number of JPGs from the queue.
Combine folder of .PNG images to a single .PDF	PT_BeginImagePDF → PT_AddImageFile for each file → PT_FinishImagePDF → PT_GetPageCount to report the output page count.
Combine folder of .TIFF images to a single .PDF	PT_BeginImagePDF → PT_AddImageFile for each file → PT_FinishImagePDF → PT_GetPageCount to report the output

page count. A multipage TIFF can contribute several PDF pages.

If a batch starts but cannot finish, the testbed calls PT_AbortImagePDF to clean up the temporary output.

4.3. Tab 2

A snapshot of this tab:



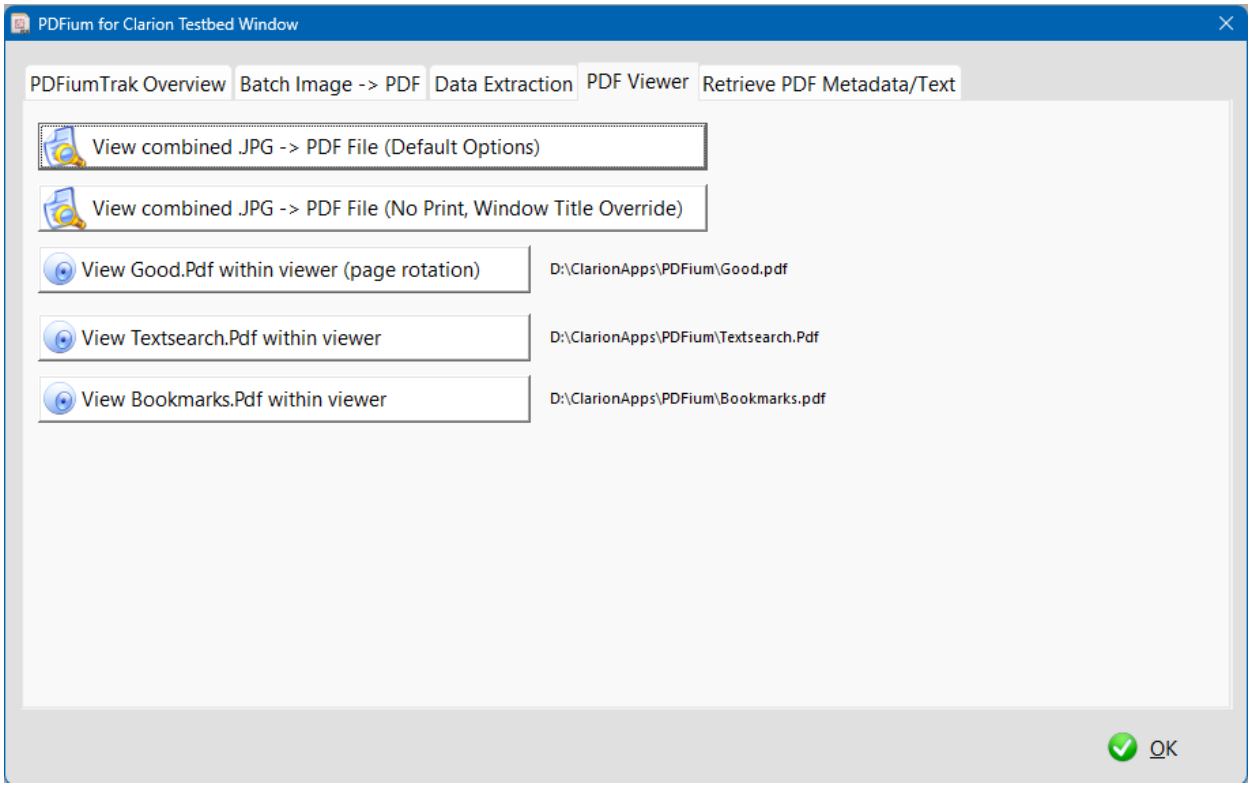
Tab 2 — Data Extraction

Button	PDFiumTrak calls
Extract PDF Page 1 to New JPG	PT_CheckPDF checks Good.pdf, then PT_ExtractTest exports page 1 using a single command string.
Extract selected page number to JPG	PT_CheckPDF checks Good.pdf, then PT_ExtractJPG exports the page number entered on screen.
Extract PDF Page 3 to New PDF	PT_SavePDFPages copies page 3 into a one-page PDF.
Extract pages 1, 3-5 from Good.Pdf to a new PDF	PT_SavePDFPages copies the four requested pages into one new PDF.
Extract each PDF page to separate PDF pages	PT_SplitPDF creates one numbered PDF per source page.

Extract PDF text to Clarion String Variable	PT_GetPDFPageText reads page 2 of Textsearch.pdf into a Clarion string.
Extract PDF Bookmarks	PT_GetPDFBookmarks reads the bookmark titles, nesting levels, and destination pages from Bookmarks.pdf.
Merge PDF files into single .PDF	PT_MergePDFs combines the PDFs from the Mergepdf folder in filename order.

4.4. Tab 3

A snapshot of this tab:



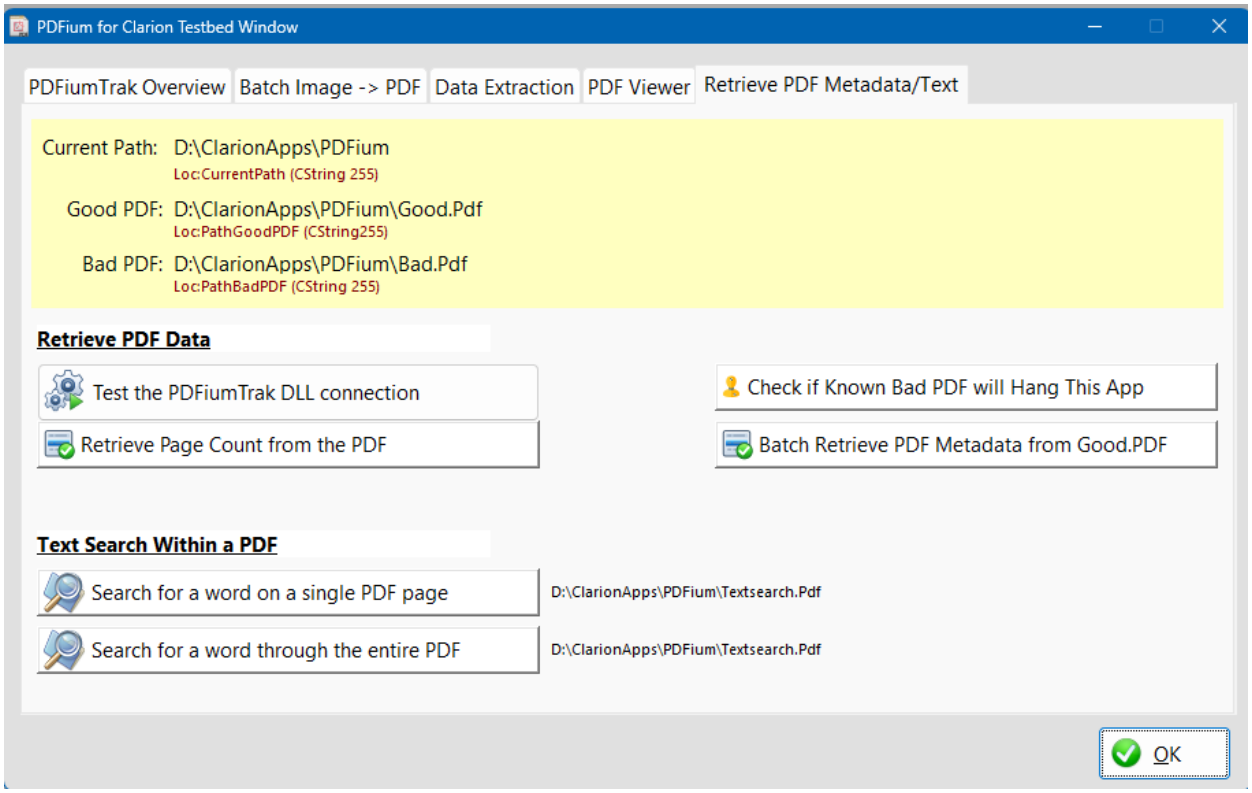
Tab 3 — PDF Viewer

Button	PDFiumTrak call
View combined JPG → PDF File (Default Options)	PT_ViewPDF opens the combined JPG output with the default title and the Print button visible.
View combined JPG → PDF File (No Print, Window Title Override)	PT_ViewPDF opens the same PDF with a custom title and hides the Print button.
View Good.Pdf within viewer (page rotation)	PT_ViewPDF opens Good.pdf. In the viewer, Rotate 180 calls PT_RotatePDFPage or PT_RotatePDFAllPages, depending on the choice.

View Textsearch.Pdf within viewer	PT_ViewPDF opens Textsearch.pdf. The viewer's Find Next uses PT_SearchPDFPage.
View Bookmarks.Pdf within viewer	PT_ViewPDF opens Bookmarks.pdf. The viewer uses PT_GetPDFBookmarks to populate its Bookmarks list.

4.5. Tab 4

A snapshot of this tab:



Tab 4 — Retrieve PDF Metadata and Text

Button	PDFiumTrak calls
Test the PDFiumTrak DLL connection	PT_TestConnection returns 12345 when Clarion reaches the expected DLL export.
Retrieve Page Count from the PDF	PT_GetPageCount counts the pages in Good.pdf.
Check if Known Bad PDF will Hang This App	PT_CheckPDF tries to open Bad.pdf and load page 1. The supplied fixture returns 1, meaning skip the file. The function runs in-process without a timeout. The testbed calls PT_CheckPDF with the deliberately incomplete Bad.pdf fixture. The first popup reports PT_CheckPDF returned 1,

	meaning the check failed and the application should skip the file. After you dismiss that popup, the testbed displays Bad PDF. In this test, the check returned control to the application. PT_CheckPDF runs without a timeout, so this result applies to the supplied fixture.
Batch Retrieve PDF Metadata	PT_GetPDFMetadata is called once for each of the eight supported fields in Good.pdf.
Search for a word on a single PDF page	PT_SearchPDFPage counts matches on page 1 of Textsearch.pdf.
Search for a word through the entire PDF	PT_FindPDFText searches forward from page 1 and reports the first matching page and its match count.

5. PDFiumTrak Support

PDFiumTrak is developed and maintained by Software by Daughtry.

For questions, bug reports, or problems using PDFiumTrak, please contact Software by Daughtry.

- When reporting a problem, please include:
- Clarion version
- PDFiumTrak version
- Description of the problem
- PDFiumTrak function being used
- Return code, if applicable
- A small code example demonstrating the problem

See Contact Us for contact information.

5.1. Troubleshooting

Clarion reports "No matching prototype available"
Check the function declaration in the Global MAP Embed against the reference. Pay particular attention to *CSTRING and *LONG. For example, pass the page number, width, and quality to PT_ExtractJPG as LONG variables so Clarion can pass them by reference.

The linker reports an unresolved external
Check that PDFiumTrak.lib is included in the application project and matches the PDFiumTrak.dll you're using. If you updated the DLL, update the import library too. For an unresolved PT_ViewPDF, also check that the project includes its own copy of PDFiumTrakViewer.clw and that line 1 names the application's generated main .clw file in the MEMBER declaration.

Windows cannot load the DLL
Check that PDFiumTrak.dll and the supplied 32-bit pdfium.dll are both beside the application executable. Once the application starts, call PT_TestConnection(). A return of 12345 confirms that Clarion reached the DLL.

A PDF could not be opened
Check the full filename your code passed and confirm the file exists at that location. If the path is correct, try the same call with the supplied Good.pdf. This helps you tell whether the problem is with the setup or with that particular PDF.

Search or text extraction produces no text
Try the same operation with Textsearch.pdf. Search page 1 for contain to check for two matches. Image-only scans need a searchable text layer before the toolkit can find or extract text from them.

A save operation reports that the destination already exists
Choose a new output filename or, for PT_SplitPDF, a new output folder. See the function's entry in PDFiumTrak Return Codes for the exact meaning of the value returned.

5.2. Contact Us

Include the Clarion version, toolkit build, function name, return value, and a thorough description of the problem and how we can reproduce the same error. Use an original test fixture or a document you can share - do NOT send us sensitive/proprietary information.

For PDFiumTrak support, questions, or bug reports:

Software by Daughtry
Website: www.sdaughtry.com
Email: scott@sdaughtry.com

When contacting support regarding a problem, please include the PDFiumTrak version, Clarion version, and any applicable PDFiumTrak return code. We answer emails as rapidly as possible.

5.3. Revision History

November 2026 Version 1.0