Developer's Guide

Revised September 28, 2017

950 Boardwalk, Suite 205, San Marcos, CA 92078 • (760) 510-1200 • www.productivecomputing.com

Table of Contents

I. Introduction

Description

The PDF Manipulator DC plug-in offers the ability to exchange data between FileMaker® and Adobe® Acrobat DC. With the plug-in FileMaker users are able to “push” and “pull” information to and from PDF form fields. The FileMaker user can also extract page text, extract or modify PDF metadata, combine several PDF documents into a single PDF, delete and print PDF files. These operations are accomplished by using FileMaker function calls from within FileMaker calculations. These calculations are generally determined from within FileMaker “SetField” or “If” script steps. For a list of available plug-in functions and their functionality please see the accompanying Functions Guide.

Product Version History

http://www.productivecomputing.com/pdf-integration/version_history

Intended Audience

FileMaker developers or persons who have knowledge of FileMaker scripting, calculations and relationships as proper use of the plug-in requires that FileMaker integration scripts be created in your FileMaker solution.

Successful Integration Practices:

1) Read the Developer’s Guide
2) Read the Functions Guide
3) Review our FileMaker Demo and video tutorials
   Demo and video tutorials: http://www.productivecomputing.com/pdf-integration
4) Familiarize yourself with Adobe Acrobat DC and how to create PDFs with form fields

Technical Note

File and full paths are formatted as follows:

Mac - Paths take the form of MountedVolume/path. Paths can also be user- relative (e.g., ~/Desktop)
For Example: /Volumes/MacHD/PDF Template.pdf

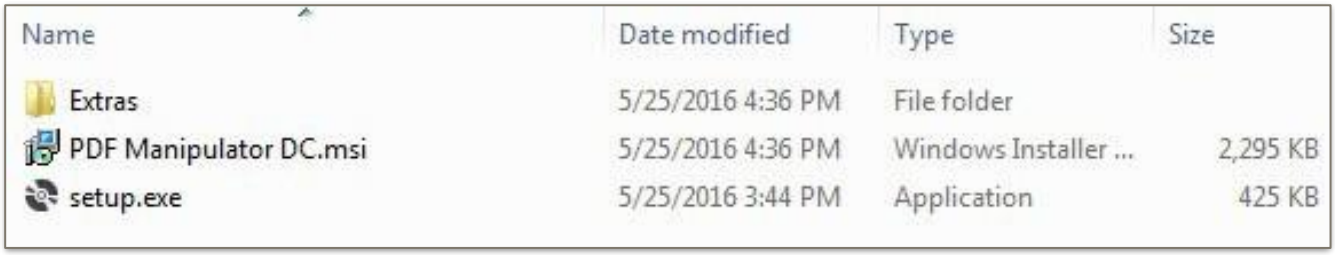
Windows - Paths take the form of DriveLetter:\path or \\ServerName\path
For Example: C:\Documents and Settings\User\Desktop\PDF Template.pdf or \\Server1\Docs\PDF Template.pdf

II. Basic Integration Steps

Accessing and using the plug-in functions involve the following steps.

1) Install the Plug-in with the Installer

With the release of PDF Manipulator DC, we've also introduced installers to make installing PDF Manipulator DC even easier. These installers will not only install the FileMaker plug-in, but will also install third party software needed for the plug-in to function, the demo file, and additional resources you may need. We recommend using the installers to ensure that all components necessary for the plug-in to function are properly installed.



Windows Installer:

1) Run the "setup.exe" file that you downloaded from our website.
2) If prompted, install the Visual C++ 2013 Runtime Libraries.
3) If you are currently running FileMaker, please close filemaker so that the plug-in will be installed correctly.
4) Accept the License.
5) Select the location to install the plug-in*
6) Confirm the installation.
7) If prompted by Windows security, allow the installer to run.
8) Your installation is complete!

*In order for FileMaker to properly recognize the plug-in, we suggest you do not change this default location. FileMaker plug-ins need to be installed in Extensions folders recognized by the application. By default, the plug-in will be installed to the base FileMaker/Extensions folder and will be available across multiple versions of FileMaker. However, if you wish to install the plug-in at a version specific location like FileMaker Pro Advanced 14/Extensions, you may browse to the folder location to do so.

OS X Installer:

1) Run the “Install PDF Manipulator DC.dmg” file that you downloaded from our website.
2) Run the “Install PDF Manipulator DC” application that is in the installer.
3) If you are currently running FileMaker, please close filemaker so that the plug-in will be installed correctly.
4) Continue through the Licensing Information, Destination Select, and Installation Type screens
5) Select “Install” if you wish to install the FileMaker plug-in, acrobat plug-in, demo file, and sample pdfs.
6) If prompted, enter your machine credentials to approve the installation.
7) Your installation is complete!

Note: Both installers come with an application (.exe or .dmg) to install the plug-in and an Extras folder. In the Extras folder, you will find additional resources such as License, README, Sample Pdfs, FileMaker Demo file, and plug-ins.

2) Install the Plug-in with the Demo File

Alternatively, you may install the plug-in using the Demo file provided in the Extras folder that came with the download from our website. Note: If you have not already, you will need to run the Visual C++ installers that are in the Extras folder in order for the plug-in to function properly.

FileMaker 12 or later:

1) Open the FileMaker demo file available in the plug-in bundle (www.productivecomputing.com).

2) Select the "Install" button.

For FileMaker 11 or earlier, follow the steps below to manually install the plug-in into the FileMaker Extensions folder.

1) Quit FileMaker Pro completely.me

2) Locate the plug-in in your download which will be located in a folder called "Plug-in." On Windows the plug-in will have a ".fmx" extension. On Mac the plug-in will have a ".fmplugin" extension.

3) Copy the actual plug-in and paste it to the Extensions folder which is inside the FileMaker program folder.
   - On Windows this is normally located here: C:\Program Files\FileMaker\FileMaker X\Extensions
   - On Mac this is normally located here: Volume/Applications/FileMaker X/Extensions (Volume is the name of the mounted volume)

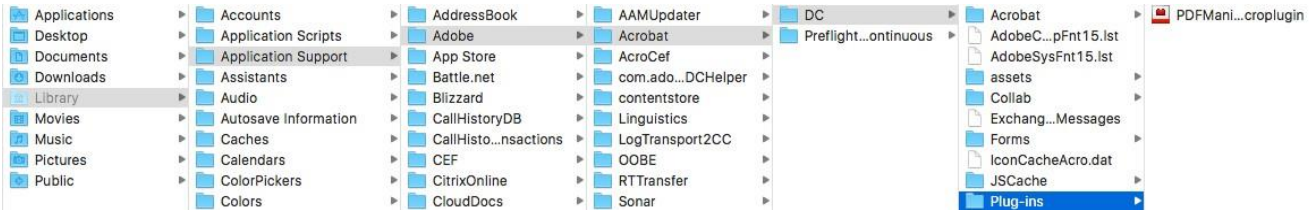
4) Start FileMaker Pro. Confirm that the plug-in has been successfully installed by navigating to "Preferences" in FileMaker, then select the "Plug-ins" tab. There you should see the plug-in listed with a corresponding check box. This indicates that you have successfully installed the plug-in.

3) Installing the Acrobat Plug-in (Mac OS X ONLY)

On Macintosh platforms you will need to install a second plug-in for Acrobat called "PDFManipulatorDCAcrobatPlugin.acroplugin"

To install this plug-in, do the following:

1) Quit Adobe Acrobat DC.
2) Navigate to the following directory ~/Library/Application Support/Adobe/Acrobat/DC/Plug-ins/
3) Copy and paste the Acrobat plug-in provided by Productive Computing directly into this folder called Plug-ins as shown.



4) Install Component for Windows 8

Installing the Microsoft Visual C++ 2008 Redistributable Package on Windows 8:

Included in the package is a download link for all users of Windows 8.
Name of link is: “Download Microsoft Visual C++ 2008 Redistributable Package (x86) (Windows 8 Install)”

This link will direct you to download the Microsoft Visual C++ Redistributable Package (x86). Windows 8 does not have a Visual C++ 2008 Redistributable Package installed by default. However, certain programs may have added it to your machine during their installation process.

If the plug-in fails to be recognized by FileMaker after installation (ie. does not show up in the Edit > Preferences > Plug-ins section), then please install the included redistributable package.

Machines running 64-bit versions of Windows 8 need to install the 64-bit ("x64") version of the redistributable package, which is also available from Microsoft.

5) Troubleshooting Plug-in Installation

When installing the plug-in using the “Install Plug-in” script step, there are certain situations that may cause a 1550 or 1551 error to arise. If such a situation occurs, please refer to the troubleshooting steps involving the most common problems that may cause those errors.

1) Invalid Bitness of FileMaker
    a. In some cases, FileMaker Pro may be attempting to install a plug-in with a different bitness than the FileMaker Pro application. This is most common with Windows plug-ins. The general rule is that the plug-in and FileMaker Pro must be the same bitness.
    b. To resolve this, ensure that the container field holding the plug-in contains the correct bitness of the plug-in. You can verify the plug-in’s bitness by checking the file extension: if the extension is .fmx, the plug-in is a 32-bit plug-in; if the extension is .fmx64, the plug-in is a 64-bit plug-in. You can verify the bitness of FileMaker Pro itself by viewing the “About FileMaker Pro” menu option in the Help menu, and clicking the “Info” button to see more information; bitness is found under “Architecture”.

2) Missing Dependencies
    a. Every plug-in has dependencies, which are system files present in the machine’s operating system that the plug-in requires in order to function. If a plug-in is “installed” into an Extensions folder, but the plug-in does not load or is not visible in the Preferences > Plug-ins panel in FileMaker Pro’s preferences, it’s likely that there are files missing.
    b. To ensure that the appropriate dependencies are installed, please verify that the Visual Studio 2013 C++ Redistributable Package is installed. This can be located by opening Control Panel and checking the Installed Programs list (usually found under “Add/Remove Programs”). Older plug-ins may require the Visual C++ 2008 redistributable package, instead of the 2013 version.
    c. Some plug-ins also have a .NET Framework component that is also required. All such plug-ins of ours will require the .NET Framework 3.5, which can be downloaded from the following link: https://www.microsoft.com/en-us/download/details.aspx?id=21

3) Duplicate Plug-in Files
    a. When installing plug-ins, it is possible to have the plug-in located in different folders that are considered “valid” when FileMaker Pro attempts to load plug-ins for use. There is a possibility that having multiple versions of the same plug-in in place in these folders could cause FileMaker Pro to fail to load a newly-installed plug-in during the installation process.
    b. To resolve this, navigate to the different folders listed in the earlier installation steps and ensure that the plug-in is not present there by deleting the plug-in file(s). Once complete, restart FileMaker and attempt the installation again. If you installed the plug-in using a plug-in installer file, if on Windows, run the installer again and choose the “Uninstall” option, or if on Mac, run the “uninstall.tool” file to uninstall the plug-in.

If the three troubleshooting steps above do not resolve the issue, please feel free to reach out to our support team for further assistance.

6) Register the Plug-in

The next step is to register the plug-in which enables all plug-in functions.

1) Confirm that you have access to the internet and open our FileMaker demo file, which can be found in the “PDF Manipulator DC Demo” folder on your Desktop.

2) If you are registering the plug-in in Demo mode, then simply click the “Register” button and do not change any of the fields. Your plug-in should now be running in “DEMO” mode. The mode is always noted on the Setup tab of the FileMaker demo.

3) If you are registering a licensed copy, then simply enter your license number in the “LicenseID” field and select the “Register” button. Ensure you have removed the Demo License ID and enter your registration information exactly as it appears in your confirmation email. Your plug-in should now be running in “LIVE” mode. The mode is always noted on the Setup tab of the FileMaker demo.

Congratulations! You have now successfully installed and registered the plug-in!

Why do I need to Register?

In an effort to reduce software piracy, Productive Computing, Inc. has implemented a registration process for all plug-ins. The registration process sends information over the internet to a server managed by Productive Computing, Inc. The server uses this information to confirm that there is a valid license available and identifies the machine. If there is a license available, then the plug-receives an acknowledgment from the server and installs a certificate on the machine. This certificate never expires. If the certificate is ever moved, modified or deleted, then the client will be required to register again. On Windows this certificate is in the form of a ".pci” file. On Mac this certificate is in the form of a “.plist” file.

How do I hard code the registration process?

You can hard code the registration process inside a simple “Plug-in Checker” script. The “Plug-in Checker” script should be called at the beginning of any script using a plug-in function and uses the PCPF_Register, PCPF_GetOperatingMode and PCPF_Version functions. This eliminates the need to manually register each machine and ensures that the plug-in is installed and properly registered. Below are the basic steps to create a “Plug-in Checker” script.

```
If [ PCPF_Version( "short" ) = "" or PCPF_Version( "short" ) = "?" ]
Show Custom Dialog [ Title: "Warning"; Message: "Plug-in not installed."; Buttons: “OK” ]
If [ PCPF_GetOperatingMode ≠ “LIVE” ]
Set Field [Main::gRegResult; PCPF_Register( “licensing.productivecomputing.com” ; “80” ; “/PCIReg/pcireg.php” ;
“your license ID” )
If [ Main::gRegResult  ≠ 0 ]
Show Custom Dialog [ Title: "Registration Error"; Message: “Plug-in Registration Failed”; Buttons: “OK”]
```

7) Open the PDF

Now that the plug-in is installed and registered, let's discuss some of the basic steps involved in successfully using the plug-in. Moving information between FileMaker and Acrobat DC first involves opening the desired PDF file by calling the PCPF_Open function(FullPath). The FullPath parameter is the system path to the desired PDF file.
If multiple versions of Acrobat exist on the system, it is best to have the desired version of Acrobat DC opened before calling the PCPF_OpenPDF function.

8) Get/Set Field Data

After the desired PDF document is opened, then you can either SET or GET the value of the PDF form fields. If you are pushing data from FileMaker to the PDF, then you will need to use the PCPF_SetPDFFieldValue function. If you are pulling data from the PDF into FileMaker, then you will need to use the PCPF_GetPDFFieldValue function. Both the PCPF_SetPDFFieldValue and the PCPF_GetPDFFieldValue functions require the FieldName or the PDF form field name. If you do not know the name of PDF form field to pass to these functions, then you can retrieve the form field names by calling the PCPF_GetFieldNames function.

Setting Field Data (Pushing)

The PCPF_SetPDFFieldValue(FieldName ; FieldValue) function sets the FileMaker values into the corresponding PDF form fields. The name of the PDF form field is passed to the function along with the FileMaker value to populate the field.
For example:
PCPF_SetPDFFieldValue("Form1.Subform.FirstName" ; "Tom") or
PCPF_SetPDFFieldValue(SomePDFFormFieldName ; SomeFileMakerTable::SomeFileMakerFieldName)

Getting Field Data (Pulling)

The PCPF_GetPDFFieldValue(FieldName) function retrieves the values of the specified PDF form field. The FieldName is the full name of the PDF form field from which to retrieve a value. The form field name is passed to the function and whatever value is contained in the PDF form field is pulled into FileMaker.
For example:
PCPF_GetPDFFieldValue("First Name") or PCPF_GetPDFFieldValue(SomePDFFormFieldName)

9) Save the PDF (optional)

After you have set the values in the PDF or in other words you have pushed values from FileMaker into the corresponding PDF form fields, you can save the changes to the PDF. You have the option to use PCPF_SavePDF function to save the changes to the currently opened PDF or you can call the PCPF_SaveAsPDF(FullPath) function to save the currently opened PDF to the new path specified.

10) Close the PDF (recommended)

Lastly you will then need to close the PDF using the PCPF_ClosePDF function. We always recommend closing the PDF once you are finished.

III. FileMaker 16 Plug-in Script Steps

Newly introduced in FileMaker Pro 16, all plug-ins have been updated to allow a developer to specify plug-in functions as script steps instead of as calculation results. The plug-in script steps function identically to calling a plug-in within a calculation dialog.

In this example, we use the FM Books Connector plug-in's script steps to demonstrate the difference. The same scripting differences would be found for any of the Productive Computing plug-in product lines.

For an example of using plug-in script steps, compare two versions of the same script from the FM Books Connector demo file: Pull Customer__Existing Session.

Script 1 - Pull Customer__Existing Session with calculation ("traditional") plug-in scripting:

```
Set Error Capture [On]
Allow User Abort [Off]
# It is assumed the session is already opened from the previous script calling this script.
# Query customers in QB (Request)

Set Variable [$$Result; Value: PCQB_RqNew( "CustomerQuery" ; "" )]
Set Variable [$$Result; Value: PCQB_RqAddFieldWithValue( "ListID" ; Main::gCust_ListID )]
If [0 <> PCQB_RqExecute]
       Exit Script [Text Result:PCQB_SGetStatus]
End If

# Pull customer info into FileMaker (Response)
Set Variable [$$Result; Value: PCQB_RsOpenFirstRecord]
Set Field [main_CUST__Customers::ListID; PCQB_RsGetFirstFieldValue( "ListID" )]
Set Field [main_CUST__Customers::FullName; PCQB_RsGetFirstFieldValue( "FullName" )]
Set Field [main_CUST__Customers::First Name; PCQB_RsGetFirstFieldValue( "FirstName" )]
Set Field [main_CUST__Customers::Last Name; PCQB_RsGetFirstFieldValue( "LastName" )]
Set Field [main_CUST__Customers::Company; PCQB_RsGetFirstFieldValue( "CompanyName" )]
Set Field [main_CUST__Customers::Bill_Address 1; PCQB_RsGetFirstFieldValue( "BillAddress::Addr1" )]
Set Field [main_CUST__Customers::Bill_Address 2; PCQB_RsGetFirstFieldValue( "BillAddress::Addr2" )]
Set Field [main_CUST__Customers::Bill_Address 3; PCQB_RsGetFirstFieldValue( "BillAddress::Addr3" )]
Set Field [main_CUST__Customers::Bill_Address 4; PCQB_RsGetFirstFieldValue( "BillAddress::Addr4" )]
Set Field [main_CUST__Customers::Bill_City; PCQB_RsGetFirstFieldValue( "BillAddress::City" )]
Set Field [main_CUST__Customers::Bill_State; PCQB_RsGetFirstFieldValue( "BillAddress::State" )]
Set Field [main_CUST__Customers::Bill_Postal Code; PCQB_RsGetFirstFieldValue( "BillAddress::PostalCode" )]
Set Field [main_CUST__Customers::Phone; PCQB_RsGetFirstFieldValue( "Phone" )]
Set Field [main_CUST__Customers::Email; PCQB_RsGetFirstFieldValue( "Email" )]

Exit Script [Text Result:0]
```

Script 2 – Pull Customer__Existing Session with plug-in script steps:

```
Set Error Capture [On]
Allow User Abort [Off]
# It is assumed the session is already opened from the previous script calling this script.
# Query customers in QB (Request)

PCQB_RqNew [Select; Results:$$Result; Request Type:”CustomerQuery”]
PCQB_RqAddFieldWithValue [Select; Results:$$Result; QB Field Name:“ListID”; Field Value:Main::gCust_ListID]
PCQB_RqExecute [Select; Results:$$Result]
If [$$Result <> 0]
       Exit Script [Text Result:PCQB_SGetStatus]
End If

# Pull customer info into FileMaker (Response)
PCQB_RsOpenFirstRecord [Select; Results:$$Result]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::ListID; Field Name:”ListID”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::FullName; FieldName:”FullName”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::First Name; Field Name:” FirstName”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Last Name; Field Name:” LastName”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Company; Field Name:” CompanyName”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 1; Field Name:”
BillAddress::Addr1”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 2; Field Name:”
BillAddress::Addr2”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 3; Field Name:”
BillAddress::Addr3”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 4; Field Name:”
BillAddress::Addr4”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_City; Field Name:” BillAddress::City”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_State; Field Name:” BillAddress::State”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Postal Code; Field Name:”
BillAddress::PostalCode”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Phone; Field Name:”Phone”]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Email; Field Name:”Email”]

Exit Script [Text Result:0]
```

Using script steps instead of the more traditional methods can make scripting within a solution more direct, as well as help with data entry validation. Some functions accept calculation-style input, while others accept a Boolean “true” or “false” option, and others employ a drop-down list for the developer to choose an option from. As stated earlier, the functionality of the plug-in script step is identical to its functionality as a calculation function; PCQB_RsOpenFirstRecord as a script step will still open the first record in the response, and store the value in the $$Result global variable (as seen in Script 2), just the same as the Set Variable script step calls PCQB_RsOpenFirstRecord (which opens the first response record) and stores the result in the $$Result variable.

For all Productive Computing, Inc., plug-ins that provide plug-in script step functionality, calculation functions will still be provided for use in development. This is to ensure that scripts already integrated with any of our plug-ins will still be viable and functional, and the developer now has the option to utilize the plug-in script steps at their discretion.

IV. Sample Scripts

Now that you understand the basic integration steps involved for getting or setting PDF form field values, let's have a look at some sample script steps. Please understand that there are many different ways to construct your scripts and these are just a few examples to get you started. For example, you could hard code the FieldValues such as "Tom" or you could reference the FileMaker field names such as Main::FirstName. You can also add loops to populate or retrieve the values depending on your design. The design depends on the experience and creativity of the FileMaker developer.

1) Push data from FileMaker to PDF form fields

Sample Script 1:

This script assumes you have a PDF with "Form1.Subform.FirstName," "Form1.Subform.LastName," "Form1.Subform.Company," "Form1.Subform.Date," "Form1.Subform.Email" and "Form1.Subform.Phone" form field names and a FileMaker table named "Main" with the "FirstName," "LastName," "Company," "Date," "Email," "Phone" and "Result" fields.

```
# Opens the desired PDF
Set Field[ Main::Result ; PCPF_OpenPDF ( "Some PDF file path" ) ]

# Sets the specified form field with values from the corresponding FileMaker field
Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.FirstName" ; Main::FirstName
) ] Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.LastName" ;
Main::LastName ) ] Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.Company"
; Main::Company ) ] Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.Date" ;
Main::Date ) ]
Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.Email" ; Main::Email ) ]
Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.Phone" ; Main::Phone
) ]

# Save the PDF
Set Field[ Main::Result ; PCPF_SaveAsPDF( "Some other PDF file path" ) ]

# Close the PDF
Set Field[ Main::Result ; PCPF_ClosePDF ]
```

Sample Script 2:

This script assumes that you have a PDF with "Form1.Subform.FirstName," "Form1.Subform.LastName," "Form1.Subform.Company," "Form1.Subform.Date," "Form1.Subform.Email" and "Form1.Subform.Phone" form field names and that you are hard coding the FieldValues.

```
# Opens the desired PDF
Set Field[ Main::Result ; PCPF_OpenPDF ( "Some PDF file path" ) ]

# Sets the specified form field with the hard coded field values
Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.FirstName" ; "Tom" ) ]
Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.LastName" ; "Smith" )
]
Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.Company" ; "ABC Company"
) ] Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.Date" ; "10/31/2009" ) ]
Set Field[ Main::Result ; PCPF_SetPDFFieldValue ( "Form1.Subform.Email" ;
"tom@abc.co.com")] Set Field[ Main::Result ; PCPF_SetPDFFieldValue (
"Form1.Subform.Phone" ; "760-510-1200" ) ]

# Save the PDF
Set Field[ Main::Result ; PCPF_SaveAsPDF ( "Some other PDF file path" ) ]
```

```
# Close the PDF
Set Field[ Main::Result ; PCPF_ClosePDF ]
```

2) Get PDF form fields Names

Sample script 1 and 2 assume that you know the PDF form field names. If you do not know the PDF form field names, then you will need to use the PCPF_GetFieldNames function to obtain the PDF form field names. Let's look at a basic and more advanced method of obtaining the PDF form field names.

The basic method will simply pull a list of all form field names into a single text field for future reference. The advanced method uses the same principle except that after the field list is gathered, then we parse through the list and add a record in FileMaker for every individual field name. In addition we also pull the value for each field from the form with this script. Let's take a look at these two basic and advanced samples.

Sample Script (Basic):

The script that best demonstrates this concept in our demo is named: "Extract Fields From PDF." We encourage you to perform this script while running debugger in the demo for a better understanding of this concept as documentation alone may not suffice.

```
# Opens the desired PDF
Set Field [ Main::Result; PCPF_OpenPDF( "Some PDF file path" ) ]

# Gets a list of the PDF form fields
Set Field [ Main::gPDF Field Listing; PCPF_GetFieldNames( 0 )]
```

Sample Script (Advanced):

** The script that best represents this concept in our demo is named: "Extract From PDF." We encourage you to perform this script while running debugger in the demo for a better understanding of this concept as documentation alone may not suffice.**

```
Go to Layout [ "Form Field Names" (Form Field Names) ]
Show All Records
Delete All Records
New
Record/Request
# Opens the desired PDF
Set Field [ Main::Result; PCPF_OpenPDF( "Some PDF file path" ) ]
# Get List of Field Names
Set Field [ Form Field Names::gListOfNames; PCPF_GetFieldNames( 0 ) ]
Set Field [ Form Field Names::gCount; ValueCount ( Form Field Names::gListOfNames )
] Set Field [ Form Field Names::gCounter; 1 ]
#Enter loop, create a new record, and get field name
    Loop
    Set Field [ Form Field Names::Name; Substitute (Trim( MiddleValues ( Form Field
    Names::gListOfNames; Form Field Names::
    gCounter; 1)) ; ¶; "") ]
    Set Field [ Form Field Names::Path Current File; Main::Path Currently Open File ]
    Set Field [ Form Field Names::Value; PCPF_GetPDFFieldValue( Form Field Names::Name
    )] Set Field [ Form Field Names::gCounter; Form Field Names::gCounter + 1
     #Exit loop when you reach the last field on the list
    Exit Loop If [ Form Field Names::gCounter > Form Field Names::gCount ]
    New Record/Request
    End
Loop End If
Go to Layout [ original layout ]
```

3) Pull data from PDF form fields into FileMaker

Now that you know how to push from FileMaker to PDF form fields and how to obtain the exact PDF form field names, let's look at some sample scripts of how to pull data from PDF form fields into FileMaker.

Sample Script 5:

This script assumes you have a FileMaker table named "Main" with the "FirstName," "LastName," "Company," "Date," "Email," "Phone" and "Result" fields and that you have a PDF with "Form1.Subform.FirstName," "Form1.Subform.LastName," "Form1.Subform.Company," "Form1.Subform.Date," "Form1.Subform.Email" and "Form1.Subform.Phone" form field names.

```
#Opens the desired PDF
Set Field[ Main::Result ; PCPF_OpenPDF (PDF file path)]

# Get field data from the PDF
Set Field[ Main::FirstName ; PCPF_GetPDFFieldValue ( "Form1.Subform.FirstName"
) ] Set Field[ Main::LastName ; PCPF_GetPDFFieldValue (
"Form1.Subform.LastName" ) ] Set Field[ Main::Company ; PCPF_GetPDFFieldValue
( "Form1.Subform.Company" ) ] Set Field[ Main::Date ; PCPF_GetPDFFieldValue (
"Form1.Subform.Date" ) ]
Set Field[ Main::Email ; PCPF_GetPDFFieldValue ( "Form1.Subform.Email" ) ]
Set Field[ Main::Phone ; PCPF_GetPDFFieldValue ( "Form1.Subform.Phone"
) ]

# Close the PDF
Set Field[ Main::Result ; PCPF_ClosePDF ]
```

Sample Script 6:

This script assumes you do not yet know the exact PDF form field names.

```
# Opens the desired PDF
Set Field[ Main::Result ; PCPF_OpenPDF (PDF file path)]

# Gets the PDF form field names. You can then enter the exact form field names obtained into the
GetPDFFieldValue function as shown below
Set Field[ Form Field Names::gListOfNames ; PCPF_GetFieldNames( 0 ) ]
Set Field[ Main::FirstName ; PCPF_GetPDFFieldValue ( "SomeFormFieldName 1"
) ] Set Field[ Main::LastName ; PCPF_GetPDFFieldValue ( "SomeFormFieldName
2" ) ] Set Field[ Main::Company ; PCPF_GetPDFFieldValue (
"SomeFormFieldName 3" ) ] Set Field[ Main::Date ; PCPF_GetPDFFieldValue (
"SomeFormFieldName 4" ) ]
Set Field[ Main::Email ; PCPF_GetPDFFieldValue ( "SomeFormFieldName 5" )
] Set Field[ Main::Phone ; PCPF_GetPDFFieldValue ( "SomeFormFieldName
6" ) ]

# Close the PDF
Set Field[ Main::Result ; PCPF_ClosePDF ]
```

V. Additional Functionality

1) Access Text from a Page or Number of Pages

The plug-in can also access all of the text contained in a PDF document on a page by page basis. To obtain all of the text content of the currently opened PDF document call the PCPF_GetTextFromPage function for each page in the document.
To access the number of PDF pages in the currently opened PDF document, then call the PCPF_GetNumPages function.

2) Access Document Information (metadata)

The plug-in can also access various metadata or document information contained within the PDF file. With a PDF document opened call either PCFP_SetPDFInfo or PCPF_GetPDFInfo depending if you desire to set or get the metadata. A list of the available metadata or information fields are:
Title, Author, Subject, Keywords, Creator, Producer, CreationDate, ModDate, or Trapped

3) Combine, Delete, or Print PDF Files

The plug-in can also combine or append PDF files using the PCPF_AppendPDF(FilePaths) function. Before appending a PDF please ensure that the PDF is first opened and afterwards ensure that you save and close the PDF. In addition, the plug-in can delete a PDF file by calling the PCPF_DeletePDF function and also print a PDF by calling PCPF_PrintPDF function.

While combining PDF files together with PCPF_AppendPDF(FilePaths), the opened PDF file will show additional pages being appended. This is standard Adobe Acrobat DC behavior. Appended pages will not be saved to the opened PDF unless the PCPF_SavePDF function is called without a specified file path. Our demo file demonstrates saving the changed PDF with the appended PDF files to a new file, through the use of PCPF_SavePDF(FullPath).

The accompanying FileMaker demo file also demonstrates many of these available plug-in functions in a FileMaker file. Additional Windows only functions are documented in the accompanying “Functions Guide.” Please see the accompanying “Functions Guide” for a detailed description of all functions. We recommend reading all documentation, reverse engineering the scripts in the demo file and watching all videos to get a better understanding of how to use the plug-in functions.

4) Inserting and Deleting Pages (NEW)

With the release of PDF Manipulator DC, we've added two new functions: PCPF_InsertPages and PCPF_DeletePages. You know have the ability merge pages from one pdf into another and to easily remove pages from a pdf.

Here are some example scripts from the Demo file that will show you how these functions can be used:

Inserting Pages:

This function requires the following parameters:

1. The page number in the currently open PDF to begin inserting at.
2. A file path to the second PDF where you will be pulling pages from.
3. The page in the second PDF to start from.
4. The number of pages to pull from the second PDF.
5. A flag to specify if you wish to retain the bookmarks from the second PDF when inserting.

```
# Open the desired PDF to insert into
Set Field[ Main::Result ; PCPF_OpenPDF( Some PDF Path ) ]

# Insert pages from a second PDF into the currently open PDF
Set Field[ Main::Result ; PCPF_InsertPages( Main::AfterPageNumber ;
Main::SecondPDFPath ; Main::StartPage ; Main::NumPages ; 0 ) ]
```

Deleting Pages:

This function requires the following parameters

1. The page number to start deleting at.
2. The page number to end deleting

```
# Open the desired PDF to delete pages from
Set Field[ Main::Result ; PCPF_OpenPDF( Some PDF Path ) ]

# Delete pages from the currently open PDF
Set Field [ Main::Result ; PCPF_DeletePages( Main::StartPage ; Main::EndPage ) ]
```

VI.Contact Us

Successful integration of a FileMaker plug-in requires the creation of integration scripts within your FileMaker solution. A working knowledge of FileMaker Pro, especially in the areas of scripting and calculations is necessary. If you need additional support for scripting, customization or setup (excluding registration) after reviewing the videos, documentation, FileMaker demo and sample scripts, then please contact us via the avenues listed below.

Phone: 760-510-1200
Email: support@productivecomputing.com
Forum: www.productivecomputing.com/forum

Please note assisting you with implementing this plug-in (excluding registration) is billable at our standard hourly rate. We bill on a time and materials basis billing only for the time in minutes it takes to assist you. We will be happy to create your integration scripts for you and can provide you with a free estimate if you fill out a Request For Quote (RFQ) at www.productivecomputing.com/rfq. We are ready to assist and look forward to hearing from you!