Productive Computing

File Manipulator

Developer's Guide

Revised June 8, 2017

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

© Copyright 2017 Productive Computing, Inc.

Table of Contents

I. Introduction

Description

The File Manipulator plug-in offers functions that support manipulating files via a FileMaker® Pro interface. The basic operation of the plug-ins is to offer the FileMaker user the ability to move, rename, copy, delete, zip, unzip, encrypt and decrypt files and folders. The plug-in also offers access to file and folder metadata such as a file's creation date or a folder's last modification date. These operations are accomplished 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 FileMaker functions and their functionality please see the accompanying Functions Guide.

Product Version History

http://www.productivecomputing.com/file-manipulator/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 Guides
3) Review our FileMaker Demo

Technical Note

All functions return a text string. It is up to the end user to convert this returned value when necessary. If an error occurs the function returns a representation of a negative number as a character string. This may help with capturing errors within your scripts.

For the following function definitions a file or folder path is formatted as follows:
Mac: Paths take the form of MountedVolume/path. Paths can also be user- relative (e.g., ~/Desktop)
Windows: Paths take the form of DriveLetter:\path (e.g. C:\Program Files) or \\ServerName\path

II. Integration Steps

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

1) Installing the Plug-in with the Installer

We are happy to announce the introduction of installers for our latest plug-in releases. These installers will not only install the FileMaker plug-in, but will also install the 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 is included in the bundle 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 and initialized 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. The FileMaker plug-in needs to be installed in a valid Extensions folder 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.

Mac Installer:

1) Run the "Install File Manipulator.dmg" file that is included in the bundle you downloaded from our website.
2) Run the "Install File Manipulator" application that is in the installer.
3) If you are currently running FileMaker, please close FileMaker so that the plug-in will be installed and initialized correctly.
4) Continue through the Licensing Information, Destination Select, and Installation Type screens.
5) Select "Install" if you wish to the install the FileMaker Plug-in and Demo File.
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, FileMaker Demo File, and plug-ins.

2) Installing 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 bundle 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.

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) 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.

4) Registering 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 "FileMaker Demo File" folder in your original download.

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, or by calling the PCFM_GetOperatingMode function.

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-in 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.

The registration process also offers developers the ability to automatically register each client machine behind the scenes by hard coding the license ID in the PCFM_Register function. This proves beneficial by eliminating the need to manually enter the registration number on each client machine. There are other various functions available such as PCFM_GetOperatingMode and PCFM_Version which can assist you when developing an installation and registration process in your FileMaker solution.

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 PCFM_Register, PCFM_GetOperatingMode and PCFM_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 [ PCFM_Version( "short" ) = "" or PCFM_Version( "short" ) = "?" ]
Show Custom Dialog [ Title: "Warning"; Message: "Plug-in not installed."; Buttons: "OK" ]

If [ PCFM_GetOperatingMode ≠ "LIVE" ]
Set Field [Main::gRegResult; PCFM_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" ]
```

5) 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.

6) File Actions

The basic operation of the plug-in offers FileMaker the ability to copy, move, rename and delete files.

<u>Copy Files</u>
Calling the PCFM FileCopy(SourcePath ; DestinationPath ; OverwriteFlag ; optDestFileName) function will copy a file from the source path to the destination path and rename if the destination file name is provided. The Overwrite Flag (Y/N) is N by default.

<u>Move Files</u>
Calling the PCFM FileMove(SourcePath ; DestinationPath ; OverwriteFlag ; optDestFileName) function will move a file from the source path to the destination path and rename if the destination file name is provided. The Overwrite Flag (Y/N) is N by default.

<u>Rename Files</u>
Calling the PCFM FileRename(SourcePath ; DestFileName) function will rename a file in the source path to the destination file name.

<u>Delete Files</u> Calling the PCFM FileDelete(SourcePath) function deletes a file from the source path.

7) File Information

The plug-in also offers access to the following file metadata:

File Size

Calling the PCFM_GetFileSize(SourcePath)function will get the size of the file in bytes.

Creation Date

Calling the PCFM_GetFileCreationDate(SourcePath)function will get the creation date and time of the file.

Last Modification Date

Calling the PCFM_GetFileModificationDate(SourcePath) function will get the modification date and time of the file.

Last Access Date

Calling the PCFM_GetFileAccessDate(SourcePath) function will get the last access date and time of the file.

File Exists

Calling the PCFM_FileExists(SourcePath) function will return a zero if the file exists and a 1 if the file does not exist.

FileMaker Path Info

Calling the PCFM_PCFM GetFileMakerPath(SourcePath)function will return the path of the FileMaker Pro application.

8) Folder Actions

The basic operation of the plug-in offers FileMaker the ability to create, copy, move, rename and delete folders.

Create Folders

Calling the PCFM_FolderCreate(SourcePath)function will create the folder (or folders) specified in the path. This function also requires the folder name in the Source Path.

Copy Folders

Calling the PCFM_FolderCopy(SourcePath ; DestinationPath)function will copy a folder (and its contents) from the source path to the destination path.

Move Folders

Calling the PCFM_FolderMove(SourcePath ; DestinationPath)function will copy a folder (and its contents) from the source path to the destination path.

Rename Folders

Calling the PCFM_FolderRename(SourcePath ; DestFolderName) function will rename a folder.

Delete Folders

Calling the PCFM_FolderDelete(SourcePath) function will delete a folder (and its contents).

9) Folder Information

The plug-in offers access to the following folder metadata:

Folder Size
Calling the PCFM_GetFolderSize(SourcePath) function will get the size of the folder (and its contents) in bytes.

Creation Date
Calling the PCFM_GetFolderCreationDate(SourcePath) function will get the creation date and time of the folder.

Last Modification Date
Calling the PCFM_GetFolderModificationDate(SourcePath) function will get the modification date and time of the folder.

Lat Access Date
Calling the PCFM_GetFolderAccessDate(SourcePath) function will return a zero (in the result field above) if the folder exists and a 1 if the folder doesn't.

File Exists
Calling the PCFM_GetFolderExists(SourcePath) function will get the access date and time of the folder.

10) Compression and Encryption

The plug-in is able to compress and encrypt file data, as well as uncompress and decrypt file data:

Compress File (Zip)

Calling the PCFM_CompressFile(Source ; Destination) function will compress the file or folder at the source path provided as a .zip file and place the .zip file at the destination path. The destination path also renames the resulting .zip file.

Expand File (Unzip)

Calling the PCFM_ExpandFile(Source ; Destination) function will expand a compressed file at the source path provided and place its contents at the destination path.

Encrypt File

Calling the PCFM_EncryptFile(Source ; Destination ; Password) function will encrypt the file at the source path and store the encrypted file at the destination path. The destination path also renames the resulting encrypted file. The file is password locked, and can be unlocked using the PCFM_DecryptFile function.

Decrypt File

Calling the PCFM_DecryptFile(Source ; Destination ; Password) function will decrypt the file at the source path using the provided password, and places the resulting decrypted file at the destination path. The destination path also renames the resulting encrypted file.

Compression with Encryption

Calling the PCFM_CompressAndEncrypt(Source ; Destination ; Password) function will both compress the file or folder at the source path and encrypt the resulting .zip file with the provided password, storing the compressed and encrypted file in the destination path. The destination path will also rename the file to the provided .zip file name.

Decryption with Expansion

Calling the PCFM_DecryptAndExpand(Source ; Destination ; Password) function will both decrypt the file using the provided password and expand the zipped archive into the destination path.

11) Drive Information

The plug-in also offers retrieval of the following drive information:

Drive Letters

Calling the PCFM_GetDrives function will get all valid drive letters separated by return characters.

Folder Names

Calling the PCFM_GetFolders(SourcePath)function will get all folder names in the path specified, separated by returns.

File Names

Calling the PCFM_GetFiles(SourcePath)function will get all file names in the path specified, separated by returns.

Number of Subfolders

Calling the PCFM_GetNumSubfolders(SourcePath)function will get the number of subfolders in the path specified.

Number of Files

Calling the PCFM_GetNumFiles(SourcePath)function will get the number of files in the path specified.

Drive Size

Calling the PCFM_GetDriveSize(SourcePath)function will get the total size of the drive in bytes.

Drive Size Free

Calling the PCFM_GetDriveSizeFree(SourcePath) function will get the free size of the drive in bytes.
Note: If accessing the primary drive of the machine, you may see inconsistent results of this function because the drive is currently in use and writing memory while in use. We recommend rounding the result to the nearest MB or GB.

Drive Size Used

Calling the PCFM_GetDriveSizeUsed(SourcePath) function will get the used size of the drive in bytes.

Note: If accessing the primary drive of the machine, you may see inconsistent results of this function because the drive is currently in use and writing memory while in use. We recommend round the result to the nearest MB or GB.

III. 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!