Productive Computing

Developer's Guide

Revised June 6, 2017

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

Table of Contents

I. Introduction

Description

The Change Printer plug-in is a simple tool designed to aid in the task of printing records in a FileMaker® Pro database. The plug-in allows for the FileMaker user to design scripts that will dynamically change printing from one printer to another allowing you to set a series of printing attributes. The plug-in also has tools that can identify all printers currently available to a given user and ways to get and set the operating system default printer. 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.

Product Version History:

http://www.productivecomputing.com/change-printer/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) Reverse engineer our FileMaker demo file and review video tutorials
Demo and video tutorials: http://www.productivecomputing.com/change-printer
4) Familiarize yourself with printing in FileMaker Pro

II. Integration Steps

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

1) Prerequisites for Plug-in Installation (Windows)

32-bit vs 64-bit:

New plug-in version 4.0.3.2 Change Printer for Windows is available in 32-bit or 64-bit mode. The plug-in bit version that you use depends upon your FileMaker Pro bit version.

Note: 32-bit applications and 32-bit plug-ins will work on a 64-bit operation system.

Installing the Microsoft Visual C++ 2013 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++ 2013 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++ 2013 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.

Please note: For older versions, use the 2008 redistributable package.

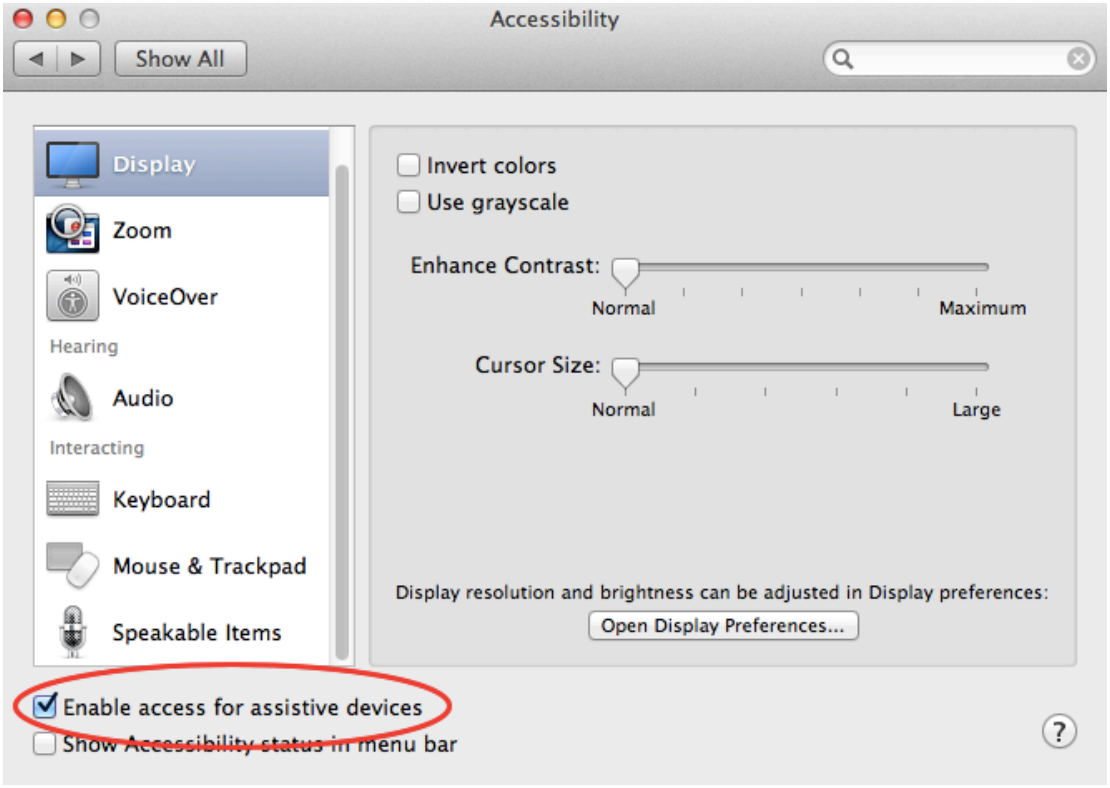
2) Prerequisites for Plug-in Installation (Mac)

32-bit vs 64-bit:

New plug-in version 4.0.6.2 Change Printer for Mac is compatible with 32-bit or 64-bit mode and will run in the mode you have selected to use for FileMaker Pro.

Accessibility:

On Mac, ensure that access for assistive devices is enabled as shown below. This can be found by navigating to "System Preferences," then "Accessibility."



3) Installing the Plug-in with the Installer

We have introduced installers to make installation of our plug-ins even easier. These installers will not only install the FileMaker plug-in file, but will also install any 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 Change Printer.dmg" file that you downloaded from our website.
2) Run the "Install Change Printer" 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.

4) Installing the Plug-in Manually

The first step is to install the plug-in into FileMaker Pro.

FileMaker 12 or later:

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

2) Select the "Install" button.

   NOTE: If you experience FileMaker error 1550 during installation, please ensure you install the Visual C++ 2013 Redistributable Package. See Known Issues for further information.

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.

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) 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.

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 PCCP_Register, PCCP_GetOperatingMode and PCCP_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 [ PCCP_Version( "short" ) = "" or PCCP_Version( "short" ) = "?" ]
Show Custom Dialog [ Title: "Warning"; Message: "Plug-in not installed."; Buttons: "OK" ]
If [ PCCP_GetOperatingMode ≠ "LIVE" ]
Set Field [Main::gRegResult; PCCP_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) 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.

8) Changing Printers

The PCCP_ChangePrinter(PrinterName ; optShowDlg ; optPause ; optCopies ; optSource ; optOrientation ; optPageSize ; optRecordSet) function is used to change the name of the printer in FileMaker to which the print job is sent and set optional attributes. The function has eight parameters. The first parameter "PrinterName" is the name of the desired printer. The second parameter "optShowDlg" tells the plug-in to either initiate the print job automatically, or allow the user to initiate the print job. The third parameter "optPause" allows the print dialog to be displayed for the desired number of milliseconds. The fourth parameter "optCopies" determines the number of copies the job will print. The fifth parameter "optSource" determines which source tray the paper will come from on the printer. The sixth parameter "optOrientation" determines the orientation of the page that prints and is only applicable on a Windows machine as on a Mac you can use the FileMaker "Print Setup" script step to select the orientation. The seventh parameter "optPagesize" determines what size paper the printer should print with. The eighth parameter "optRecordSet" determines whether the print job will be printing the records being browsed or the currently opened record. All parameters are explained in further details in the "Functions Guide."

The plug-in requires that the print dialog be initially displayed, as when called by the Print[] script step. The plug-in will "Grab" the print dialog and make the requested changes. If the "optShowDlg" parameter is set to "1", the plug-in will allow the user to initiate the print job by clicking "OK". If the "optShowDlg" parameter is set to "0", the plug-in will "click" OK for the user, thus initiating the print job itself.

The PCCP_ChangePrinter function is used in conjunction with FileMaker's Print[] script step. Immediately after calling the PCCP_ChangePrinter function a subsequent call to FileMaker's Print[] script step should be called.

Before calling the PCCP_ChangePrinter function it is recommended that you obtain a list of all printers available. The PCCP_GetPrinterAt(index) function returns the name of a printer located in the systems internal list of printers. The value returned by the function is valid to use with the PCCP_ChangePrinter function.

The easiest way to describe how to use these functions to change printers and get valid printer names is to show some example scripts for different scenarios.

In the example scripts below these assumptions are made:
- A layout named 'Printable Layout' exists in the FileMaker solution
- There is a table named 'Main' with a global variable named 'gResult'
- There is a table named 'Printers' with a field named 'Names'
- One of the names returned by PCCP_GetPrinterAt is "\\Brother on DC1"

9) Get Valid Printer Names

The first step with any scenario is to decide to which printer you wish to print. The valid names of available printers can be retrieved with the PCCP_GetPrinterAt(index) function. This function is used to determine the exact name of a printer as it will be used in the PCCP_ChangePrinter function. To demonstrate using the function the following example script iterates through all of the printers on the local machine and grabs the appropriate name. Each name is stored in its own record in a "Printers" table.

```
###
Go To Layout[ Printers ]
Show All Records
Delete All Records[ No Dialog ]
Set Field[ Printers::Counter ; 1 ]
Loop
  New Record
  Set Field[ Printers::Name ; PCCP_GetPrinterAt( Printers::Counter ) ]
  Exit Loop If [ Left( Name ; 9 ) = "!!ERROR!!" ] //there are no more printers
  Set Field[ Printers::Counter ; Printers::Counter + 1 ]
End Loop
#remove the last record as it does not hold a printer name
Delete Record[ No Dialog ]
###
```

Each record in the Printers table now holds a valid name for a printer. Any of these names can be used in the PCCP_ChangePrinter function call.

10) Automatic Printing

You have the option of printing automatically without requiring user interaction to press "OK" in the print dialog screen. This is accomplished by omitting the second parameter in the ChangePrinter function or by passing "False" or 0 as the second parameter. This can be written one of the following four ways:

a. PCCP_ChangePrinter("\\Brother on DC1")

b. PCCP_ChangePrinter("\\Brother on DC1" ; "")

c. PCCP_ChangePrinter("\\Brother on DC1" ; "False")

d. PCCP_ChangePrinter("\\Brother on DC1" ; "0").

Please note that although the function "presses" the print button before quitting, the user will briefly see the print dialog displayed by FileMaker.

In the following example script below the first step navigates to the proper layout. The next step then sets the desired printer. Then we finally print the document. The plug-in will dismiss the print dialog as soon as the printer has been properly changed and automatically print the document o the desired printer without user interaction.

```
###
Got To Layout[ "Printable Layout" (Main) ]
Set Field[ Main::gResult ; PCCP_ChangePrinter( "\\Brother on DC1" ) ]
Print[]
Go To Layout[ original layout ]
###
```

11) Automatic Printer Selection Without Printing

You also have the option of selecting the printer without printing. This will require user interaction to press "OK" in the print dialog screen. This is accomplished by passing "True" or 1 as the second parameter in the ChangePrinter function. This is written the following way: PCCP_ChangePrinter("\\Brother on DC1" ; "True") or PCCP_ChangePrinter("\\Brother on DC1" ; "1"). The function quits after setting the printer name.

This proves desirable when your want to select the printer for the end user, but want the end user to actually select "OK" to send the print job to the printer. **In the following example script we add a second parameter to the PCCP_ChangePrinter function which tells the plug-in to only select the proper printer but does not dismiss the dialog.** This lets the end user make any other adjustments to the job before sending it to the printer.

```
###
Got To Layout[ "Printable Layout" (Main) ]
#pass a boolean 'true' value (True or 1) in the second parameter
# This leaves the Print dialog on the monitor
SetField[ Main::gResult ; PCCP_ChangePrinter( "\\Brother on DC1" ; 1 ) ]
Print[]
Go To Layout[ original layout ]
###
```

12) Optional Printing Parameters or Attributes

You also have the option of setting optional parameters giving you more flexibility over the various printing attributes. The PCCP_ChangePrinter(PrinterName ; optShowDlg ; optPause ; optCopies ; optSource ; optOrientation) function now offers parameters to add a pause into the print job, select the number of copies to print, select the source paper tray and determine the orientation of the page. Let's briefly explore these parameters and for a complete detailed explanation see the "Functions Guide."

optPause sets the desired number of milliseconds to pause the print dialog before the print job proceeds.
optCopies determines the number of copies the job will print.
optSource determines which source tray the paper will come from on the printer.
optOrientation determines the orientation of the page that prints and is only applicable on a Windows machine as on a Mac you can use the FileMaker "PrintSetup" script step to select the orientation.
optPageSize determines the page size to format the print page to.
optRecordSet determines the set of records to print, such as all records in the found set or just the current record.

For example:
PCCP_ChangePrinter("\\Brother on DC1" ; 0 ; 1000 ; 3 ; "Tray 1" ; 0) or
PCCP_ChangePrinter("\\Brother on DC1" ; 1 ; "" ; 5 ; "Tray 2" ; 1)

When setting up the scripting to handle the changing of printers, it may be efficient to omit or leave certain parameters blank. By "blanking" a parameter, the plug-in will use the settings as defined by the Print[] script step. For example, if the Print[] script step is defined to use the page size "A4" and the page source "Tray 1", then the Change Printer function call could be written as such:

Set Field [someTable::someField ; PCCP_ChangePrinter($printerName ; $showDlg ; $pauseTime ; $numCopies ; "" ; $orientation ; "")]

In this case, the script would define the values for the printer name, the option to show the print dialog and how long to pause and display the dialog, the number of copies, and the orientation of the page, but default to Print[]'s definitions of "Tray 1" for the page source and "A4" for the page size.

Additionally, parameters can be omitted to achieve the same effect. Using the above example, if $orientation is also desired to be set to the Print[] default, the function call can be written as such:

Set Field[someTable::someField ; PCCP_ChangePrinter($printerName ; $showDlg ; $pauseTime ; $numCopies)

Note that parameters can be omitted only from the end of the parameter list; if a parameter is to be omitted that has another parameter after it (i.e. omitting optSource and defining optPageSize), the following parameter must be omitted, or the first parameter must be defined as "".

Please see the "Functions Guide" under the PCCP_ChangePrinter function for further examples and clarification.

13) Get or Set System Printer

You also have the option of getting or selecting the default system printer. This is accomplished using either the PCCP_GetSystemPrinter or PCCP_SetSystemPrinter functions.

We recommend reading the "Functions Guide" in order to realize the full potential of the plug-in functions. If there is missing functionality that you would like to have in the plug-in, please feel free to e-mail a request to support@productivecomputing.com.

14) Known Issues

Plug-in Installation - FileMaker Error 1550

When attempting to install the plug-in, FileMaker may return an error 1550, which is defined as "Unable to install plug-in". The most common case is that the system lacks the required dependencies in order to run the plug-in.

To resolve the error, there is a download link in the bundle for the plug-in that points the Visual C++ 2013 Redistributable Package, available from Microsoft's support website. Save the installer file and run it on your machine. This should add the required dependencies into your environment. Once the installer is finished, re-open FileMaker, and you should see a splash screen upon initialization.

Behavior Issue with Multiple FileMaker Sessions in Yosemite

Reliable results for print switching requires that only one version of FileMaker is running in the environment.

III. Error Handling

When something unexpected happens, a plug-in function will return a result of !!ERROR!!. This makes it simple to check for errors. If a plug-in function returns !!ERROR!!, then immediately after call PCCP_GetLastError(Type) function for a detailed description of what the exact error was.

We find that most developers run into issues due to a lack of error trapping. Please ensure that you properly trap for errors in your solutions. Here are a few samples of how you can check for errors.

```
Set Variable [ $result = MyPluginFunction( "a" ; "b" ; "c" ) ]
If [ $result = !!ERROR!! ]
Show Custom Dialog [ "An error occurred: " & PCCP_GetLastError ]
End If
```

The PCCP_GetLastError(format) function gives you the option to display the error description or error number. Displaying the error number is more user friendly in international environments, where an English error description may not be desired. If the format parameter is set to "Number" such as PCCP_GetLastError ("Number"), then an error number will be returned. If format parameter is empty such as PCCP_GetLastError or PCCP_GetLastError(""), then an English error description will be returned.

Please find a list of return codes and descriptions below for your reference.

Error Number	Error Text
-1	Plug-in not registered or session expired
-3	Invalid # of Parameters
-4	Invalid Parameter value(s)
-10	Failed Registration
-5000	Specified printer not found
-5001	Index out of bounds
-5002	Unable to get system printer
-5003	Still waiting for last call to finish
-5004	No printers listed
-5005	Unable to set system printer

IV. 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!