



Productive
Computing



Developer's Guide

Revised October 26, 2011

Table of Contents

I. Introduction	3
II. Integration Steps	4
1) Installing the Plug-in	4
2) Registering the Plug-in	4
3) Choosing a Gateway	6
4) Working with Authorize.net	7
5) Working with PayPal.....	8
6) Working with Ogone	9
7) Working with Eway	10
8) Handling Errors	11
III. Contact Us	12

I. Introduction

Description:

The FM Credit Card is a credit card processing plug-in for FileMaker. With this plug-in you are able to automatically process secure SSL encrypted credit card payments through your gateway, all from within FileMaker. The plug-in works with an array of gateways to meet all your needs and provide you with more options. These operations are accomplished using FileMaker function calls from within FileMaker calculations. These calculations are generally determined from within FileMaker "SetField," "SetVariable" or "If" script steps. For a list of the basic integration steps, please see the accompanying Functions Guide document.

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 can be found here: <http://www.productivecomputing.com/plugins/fm-credit-card-id-133/>

- 4) Familiarize yourself with your credit card gateway

Error Handling:

Any of the plug-in functions may encounter an error during processing. When an error occurs during processing, immediately call the PCCC_GetLastError function in order to obtain a full description of the error or error number. This function returns the error message or error number associated with the last error in order to troubleshoot script or logic failures. Please see PCCC_GetLastError function description in the Functions's Guide and the "Handling Errors" section in the Developer's Guide for further clarification on how to properly trap for errors.

II. Integration Steps

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

1) Installing the Plug-in

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

- 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. Confirm that the plug-in has been successfully installed by navigating to "Preferences" in FileMaker Pro, then click 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.

2) Registering the Plug-in

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

- 5) 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.
- 6) If you are registering the plug-in in Demo mode, then simply click the "Register the Plug-in" button and do not change any of the fields. Your plug-in should now be running in "DEMO" mode. The mode is noted in our FileMaker Demo file on the Setup tab.
- 7) If you are registering a licensed copy, then simply enter your license number in the "LicenseID" field and click the "Register the Plug-in" button. Make sure you remove 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 indicated on the Setup tab in our FileMaker Demo file or by calling the PCCC_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 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 PCCC_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 PCCC_GetOperatingMode and PCCC_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 PCCC_Register, PCCC_GetOperatingMode and PCCC_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 [ PCCC_Version( "short" ) = "" or PCCC_Version( "short" ) = "?" ]
Show Custom Dialog [ Title: "Warning"; Message: "Plug-in not installed."; Buttons: "OK" ]
If [ PCCC_GetOperatingMode ≠ "LIVE" ]
Set Field [Main::gRegResult; PCCC_Register( "licensing.productivecomputing.com" ; "80" ; "/PCIReg/pcireg.asp" ;
"your license ID" )
If [ Main::gRegResult ≠ 0 ]
Show Custom Dialog [ Title: "Registration Error"; Message: "Plug-in Registration Failed"; Buttons: "OK" ]
```

Please also watch our setup video found here: <http://www.productivecomputing.com/video/?p=1345> for additional setup information.

3) Choosing a Gateway

In order to automate credit card processing in FileMaker, you must first choose a gateway to work with. All transactions in any of the gateways listed below are secure, SSL encrypted. The FM Credit Card works with the following four gateways:

- Authorize.net = <https://developer.authorize.net/testaccount/>
- PayPal = https://merchant.paypal.com/cgi-bin/marketingweb?cmd=_render-content&content_ID=merchant/payment_gateway
- OGone = <http://www.ogone.com/>
- Eway = <http://www.eway.com.au/>

You can set up test (or live) accounts by clicking the corresponding links above. Authorize.net and PayPal tend to be very popular in the US, Canada, China and South/Central America while OGone is more Euro friendly and Eway services Australia. Between these four gateways you should be able to find one to suit your needs. Particularly because you can use your bank or other financial institutions in conjunction with your gateway as your merchant account service provider for processing your transactions.

Below are a few links to provide you with more information about merchant accounts and banking partners.

Authorize.net

<http://www.authorize.net/solutions/merchantsolutions/resellerdirectory/>

(you can even check by state)

<http://www.authorize.net/solutions/merchantsolutions/stateresellerdirectory/>

Eway

<http://www.eway.com.au/about-eway-payment-service-provider/eway-explained/connected-payment-banks.aspx>

Ogone

<http://cps.ogone.com/en/Extra%20Services/Merchant%20Account%20Facilitation.aspx> and this pdf

http://cps.ogone.com/en/Extra%20Services/~//media/PDF/Acquiring_partners_PM_ENG1.ashx

Now that you have selected a gateway, let's discuss your various options with each gateway.

4) Working with Authorize.net

Authorize.net offers the following features:

Authorize and Capture using the PCCC_AN_AuthCapt(APILoginID ; TransactionKey ; Amount ; CardNum ; ExpDate; OptParam) function. This function authorizes that the requested dollar amount is available on a customer's credit card, and then immediately captures the authorized amount to the merchant's account. It returns the transaction ID, to be stored for further processing, such as a Void or Credit transaction.

Authorize using the PCCC_AN_Authorize(APILoginID ; TransactionKey ; Amount ; CardNum ; ExpDate; OptParam) function. This function authorizes that a customer has the requested dollar amount available on their credit card, but does not capture those funds. The transaction ID is returned to be stored for capture at a later date.

Capture using the PCCC_AN_Capture(APILoginID ; TransactionKey ; TransID ; Amount; OptParam) function. This function captures funds from a previous call to Authorize. A developer can also use the "x_amount=" key value pair to Capture less than the originally authorized amount.

Capture Only using the PCCC_AN_CaptOnly(APILoginID ; TransactionKey ; AuthCode ; Amount ; CardNum ; ExpDate; OptParam) function. This function can capture funds from an Authorization not performed on the gateway. This requires a special Authorization number from the customer's bank, which would be entered into the highlighted field.

Credit using the PCCC_AN_Credit(APILoginID ; TransactionKey ; TransID ; Amount ; CardNum; OptParam) function. This function refunds a customer from a previously captured transaction. It is a good rule of thumb to try and void the transaction before refunding the customer, because the funds may not have yet cleared. Simply provide transaction ID, the amount you wish to refund (up to the original amount of the transaction) and at least the last four digits of the customer's credit card.

UCredit using the PCCC_AN_UCredit(APILoginID ; TransactionKey ; Amount ; CardNum ; ExpDate; OptParam) function. This function allows the merchant to refund a customer for a transaction not performed on the gateway, or a transaction that is past the typical 90 day refund window. This function requires special permission from the merchant's bank, and carries a higher risk.

Void using the PCCC_AN_Void(APILoginID ; TransactionKey ; TransID; OptParam) function. This function can void an Authorized or Captured transaction. It can be used on a previously Authorized transaction while it is still valid, or on a previously captured transaction while the funds have not yet cleared (typically before the end of the day).

Custom Posts can be created by calling the PCCC_AN_RawPost(APILoginID ; TransactionKey; OptParam) function. This is an advanced function that allows you to completely customize the post request.

Please see the Functions Guide and reverse engineer the demo file for exact functionality and use. A tip is that you may want to store this resulting transaction ID returned as you may need it to CREDIT or VOID the transaction later and this can only be done with the transaction key.

There are also numerous additional optional parameters "OptParam" you can provide, such as addresses, email, PO number, etc. These are not required to process an order, but can be useful for address verification purposes or linking a transaction with a customer or invoice. To supply additional parameters, add them as extra parameters onto the end of your function call in the form 'key=value', like this: PCCC_AN_AuthCapt ("PCI3u3zM6b" ; "92cFg7PCIG89cY"; "3.50" ; "601100000000012" ; "122020" ; "x_email=test@test.com" ; "x_address=123 Fake St")

5) Working with PayPal

PayPal offers the following features:

Sale using the `PCCC_PP_Sale(Partner ; Vendor ; User ; Password ; Amount ; CardNum ; ExpDate; OptParam)` function. This function authorizes that the requested dollar amount is available on a customer's credit card, and then immediately captures the authorized amount to the merchant's account. It returns the transaction ID, to be stored for further processing, such as a Void or Credit transaction.

Authorize using the `PCCC_PP_Authorize(Partner ; Vendor ; User ; Password ; Amount ; CardNum ; ExpDate; OptParam)` function. This function authorizes that the requested dollar amount is available on a customer's credit card, and then immediately captures the authorized amount to the merchant's account. It returns the transaction ID, to be stored for further processing, such as a Void or Credit transaction.

Capture using the `PCCC_PP_Capture(Partner ; Vendor ; User ; Password ; OrigID; OptParam)` function. This function captures funds from a previous call to `Authorize`, identified by the transaction ID.

VoiceAuth using the `PCCC_PP_VoiceAuth(Partner ; Vendor ; User ; Password ; AuthCode ; Amount ; CardNum ; ExpDate; OptParam)` function. This function can capture funds from an Authorization that was performed over the phone. This requires a special Authorization Code from the customer's bank, which would be entered into the Authorization code field.

Credit using the `PCCC_PP_Credit(Partner ; Vendor ; User ; Password ; OrigID; OptParam)` function. This function can refund a customer for a previously captured transaction, identified by the Authorization Code from that transaction.

Void using the `PCCC_PP_Void(Partner ; Vendor ; User ; Password ; OrigID; OptParam)` function. This function can void a previous call to the `Authorize` function, identified by the transaction ID.

Inquire using the `PCCC_PP_Inquire(Partner ; Vendor ; User ; Password ; OrigID; OptParam)` function. This function returns the status of any transaction identified by its transaction ID.

Custom Posts can be created by calling the `PCCC_PP_RawPost(Partner ; Vendor ; User ; Password; OptParam)` function. This is an advanced function that allows you to completely customize the post request.

Please see the Functions Guide and reverse engineer the demo file for exact functionality and use. A tip is that you may want to store this resulting transaction ID returned as you may need it to CREDIT or VOID the transaction later and this can only be done with the transaction key.

There are also numerous additional optional parameters "OptParam" you can provide, such as addresses, email, PO number, etc. These are not required to process an order, but can be useful for address verification purposes or linking a transaction with a customer or invoice. To supply additional parameters, add them as extra parameters onto the end of your function call in the form 'key=value', like this: `PCCC_PP_Sale("PayPal" ; "ProductiveComputing" ; "someUser1" ; "somePassword" ; "10.01" ; "5105105105105100" ; "1220" ; "STREET=123 Fake St." ; "EMAIL=test@test.com")`

6) Working with Ogone

Ogone offers the following features:

Sale using the PCCC_OG_Sale(PSPID ; USERID ; PSWD ; OrderID ; Amount ; Currency ; CardNum ; ExpDate ; CVC; OptParam) function. It authorizes that the requested dollar amount is available on a customer's credit card, and then immediately captures the authorized amount to the merchant's account. It returns the transaction ID, to be stored for further processing, such as a Void or Credit transaction.

Authorize using the PCCC_OG_Authorize(PSPID ; USERID ; PSWD ; OrderID ; Amount ; Currency ; CardNum ; ExpDate ; CVC; OptParam) function. This function authorizes that a customer has the request dollar amount available on their credit card, but does not capture those funds. The transaction ID is returned to be stored for capture at a later date.

Capture using the PCCC_OG_Capture(PSPID ; USERID ; PSWD ; PAYID ; Amount ; Final; OptParam) function. This function captures funds from a previous call to Authorize, identified by the transaction ID. The "Final" parameter represents whether or not more transactions can be performed on the provided ID. For example, if an authorization was made for \$100, the merchant could perform two captures for \$50 if "false" is passed to the Final parameter. A value of "true" will close the transaction for further processing.

Renew using the PCCC_OG_Renew(PSPID ; USERID ; PSWD ; PAYID ; Amount; OptParam) function. This function can renew an authorization to provide more time before capturing the customer's funds.

DelAuth using the PCCC_OG_DelAuth(PSPID ; USERID ; PSWD ; PAYID ; Amount ; Final; OptParam) function. This function can delete a previous call to the Authorize function, identified by the transaction ID. If a value of "false" is passed to the "Final" parameter, then the original Authorization can be renewed with "Renew".

URefund using the PCCC_OG_URefund(PSPID ; USERID ; PSWD ; OrderID ; Amount ; Currency ; CardNum ; ExpDate ; CVC; OptParam) function. This function allows the merchant to refund a customer for a transaction not performed on the gateway, or a transaction that is past the typical 90 day refund window. This function requires special permission from the merchant's bank and carries a higher risk.

Refund using the PCCC_OG_Refund(PSPID ; USERID ; PSWD ; PAYID ; Amount ; Final; OptParam) function. This function can refund a customer for a previously captured transaction, identified by the Authorization Code from that transaction. A value of "false" provided to the "Final" parameter will allow further refunds to be performed on the original transaction.

Query using the PCCC_OG_Query(PSPID ; USERID ; PSWD ; PAYID; OptParam) function. This function returns the status of any transaction identified by its transaction ID.

Custom Posts can be created by calling the PCCC_OG_RawPost(PSPID ; USERID ; PSWD ; Type; OptParam) function. This is an advanced function that allows you to completely customize the post request.

Please see the Functions Guide and reverse engineer the demo file for exact functionality and use. A tip is that you may want to store this resulting transaction ID returned as you may need it to CREDIT or VOID the transaction later and this can only be done with the transaction key.

There are also numerous additional optional parameters "OptParam" you can provide, such as addresses, email, PO number, etc. These are not required to process an order, but can be useful for address verification purposes or linking a transaction with a customer or invoice. To supply additional parameters, add them as extra parameters onto the end of your function call in the form 'key=value', like this: PCCC_OG_Sale ("ProductiveComputing" ; "PCPay" ; "PCI123" ; "123456789" ; "10.00" ; "EUR" ; "4111111111111111" ; "122020" ; "111" ; "Owneraddress=123 Fake St." ; "EMAIL=test@test.com")

7) Working with Eway

Eway offers the following features:

Sale using the PCCC_EW_Sale(CustomerID ; TotalAmount ; CardHoldersName ; CardNumber ; ExpMonth ; ExpYear) function. This function authorizes that the requested dollar amount is available on a customer's credit card, and then immediately captures the authorized amount to the merchant's account. It returns the transaction ID, to be stored for further processing, such as a Void or Credit transaction.

Refund using the PCCC_EW_Refund(CustomerID ; TotalAmount ; ExpMonth ; ExpYear ; OrigTrxnNum ; RefundPassword) function. This function can refund a customer for a previously captured transaction, identified by the Authorization Code from that transaction.

Custom posts and "OptParam" are not available with this gateway.

8) Handling Errors

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 PCCC_GetLastError("Text") function for a detailed description of what the exact error was or PCCC_GetLastError("Number") for an error number.

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: " & PCCC_GetLastError( "Text" ) ]
End If
```

The PCCC_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 PCCC_GetLastError("Number"), then an error number will be returned. If format parameter is empty such as PCCC_GetLastError or PCCC_GetLastError("Text"), then an English error description will be returned. The error numbers and their meanings can be found below.

Error Number	Error Text
0	Success
-1	Plug-in not registered or session expired
-3	Invalid # of Parameters
-4	Invalid Parameter value(s)
-10	Failed Registration
-11001	Gateway operation failed.
-11005	KeyValue pair must not be empty.
-11006	Unrecognized key passed as parameter.

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!