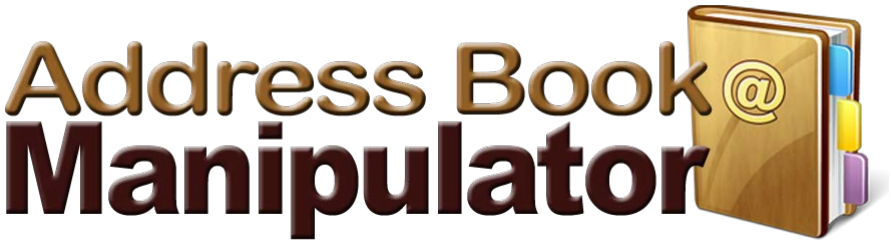
Productive Computing

Creating Efficiency Through Automation



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 Address Book Manipulator plug-in offers functions that support a bidirectional data exchange between FileMaker® and Apple® Address Book or Contacts. With this plug-in FileMaker users are able to add, edit, and delete records into the address book, pull records from the address book, and search for specific records in the address book. 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.

Product Version History:

http://www.productivecomputing.com/address-book-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 Guides
3. Download a demo: http:// www.productivecomputing.com/address-book-integration
4. Watch video tutorials: http://www.productivecomputing.com/video
5. Familiarize yourself with Apple's Address Book/Contacts

II. INTEGRATION STEPS

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

1) Installing the Plug-in with the Installer

This 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.

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: The installer comes 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) 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 download from our website.

FileMaker 12 or later:

1. Open the FileMaker demo file available in the plug-in bundle (www.productivecomputing.com).
2. Select the "Install" button.

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 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. **NOTE:** 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) 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 troubleshooting step above does 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.

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 called “PCAB.plist” and is located in the User/Library/Preferences folder.

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 PCAB_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 PCAB_GetOperatingMode and PCAB_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 PCAB_Register, PCAB_GetOperatingMode and PCAB_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 [ PCAB_Version( "short" ) = "" or PCAB_Version( "short" ) = "?" ]
Show Custom Dialog [ Title: "Warning"; Message: "plugin not installed."; Buttons: "OK" ]
If [ PCAB_GetOperatingMode ≠ “LIVE” ]
Set Field [Main::gRegResult; PCAB_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) Creating Integration Scripts

Developer's Note: From Mac OS X 10.8 and newer, Apple Inc. has renamed the Address Book app to "Contacts". Any reference to Address Book also applies to Contacts.

In order to push or pull data between FileMaker and Apple's Address Book you must create integration scripts in your FileMaker solution. The plug-in is not a canned solution, but rather a developer tool allowing the FileMaker developer to integrate the plug-in functionality into your FileMaker solution. The FileMaker demo file that comes with the plug-in is just a demo used to show the basic concept of how the plug-in functions operate. Your FileMaker file will take the place of the FileMaker demo file. This document was designed for FileMaker developers or persons who are versed in FileMaker scripting, calculations, and relationships.

Using the plug-in functions you are able to push information from FileMaker to Address Book, search for specific Address Book records and pull information from Address Book into FileMaker. Let's have a closer look at these three actions below.

Tip: It is recommended that you create an archive or back up copy of your Address Book contacts. From Address Book go to File > Export > Contacts Archive.

A. 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.

B. Push to Address Book

You have the option to push a new record from FileMaker to Address Book, update an existing Address Book record, or delete an existing Address Book record. We recommend pushing no more than 1,000 records into Address Book at a time due to a memory management issue that causes memory to be saturated. Pushing more than 1,000 contacts from FileMaker to Address Book could result in FileMaker crashing. This appears to be a known Apple issue, and it's reported that the same scenario happens with users pushing multiple contacts between iCloud and Address Book (outside of FileMaker). Productive Computing will continue to investigate the issue to overcome this limitation, but currently it appears that the responsibility lies with Apple. In the meantime, you can simply push 1,000 or less contacts per session. Please quit FileMaker between each batch of 1,000 contacts you push. This issue affects pushing contacts only. Pulling contacts well above 1,000 appears to have no issue and should work fine.

Let's take a look at the functions involved in pushing records to Address Book and sample script steps below.

Add a New Record to Address Book

Pushing a new contact record to Address Book is accomplished by first creating a new record in Address Book, setting the values for the different properties and then saving the new Address Book record.

A list of all the functions necessary to create and populate a new Address Book record are:
PCAB_New(RecordType ; optType)
PCAB_SetValueForProperty(Property ; Value ; optType
) PCAB_SetImage(BinaryData)
PCAB_AddAddress(Label ; Street ; City ; State ; Zip ; Country ; CountryCode)
PCAB_AddService(IMorSocialProfile ; Label ; Type ; Username ; optSocialURL ; optSocialUserID
) PCAB_AddMV(Property ; Value ; Label)
PCAB_Save

Each of the above functions is described in the Functions Guide that accompanies this document. The Functions Guide lists the complete description, parameters, and return values for each function. For the sake of brevity, we will cover only the basics of each function.

PCAB_New(RecordType ; optType) creates a new empty "Person" or "Group" record type in Address Book. The

Address Book's unique ID for the newly created record is returned.

PCAB_SetValueForProperty(Property ; Value ; optType) is used to set the value for any of the Address Book record's single valued properties. A single valued property is one that can have only one value in Address Book, such as a "First Name" or “Last Name.”

PCAB_AddMV(Property ; Value ; Label) is used to add a value to a multi-valued property. Multi-valued properties are those in Address Book that may have more than one value, such as a work phone number and home phone number or a work email address and home email address, etc.

PCAB_AddAddress(Label ; Street ; City ; State ; Zip ; Country ; CountryCode) adds an address to the Address Book record with the values specified. An address is a multi-valued property so several addresses can be added to any Person record in Address Book.

PCAB_AddService(IMorSocialProfile ; Label ; Type ; Username ; optSocialURL ; optSocialUserID) adds a social profile or instant message service to the Address Book record with the values specified. Like normal addresses, a service is a multi-valued property, so several services can be added to any Person record in Address Book.
Note that the Social URL and Social UserID parameters are optional; they are only valid if adding a social profile service, but are not required.

PCAB_Save saves the record and its contents to the Apple Address Book.

An example script adding a new contact is:

```
...
Set Field [ Contacts::ABID ; PCAB_New( "Person" ) ]
If [ Contacts::ABID = "!!ERROR!!" ]
   Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
   Halt Script
End If
Set Field [ Main::gResult ; PCAB_SetValueForProperty( "First Name" ; “Joe” ) ]
Set Field [ Main::gResult ; PCAB_SetValueForProperty( "Last Name" ; “Smith” )]
Set Field [ Main::gResult ; PCAB_SetImage( someTable::somePicture )]
Set Field [ Contacts::addr1UID ; PCAB_AddAddress( "work" ; "123 Some Street" ; "AnyTown" ;"AnyState";
"12345"; "USA"; "us" ) ]
Set Field [ Contacts::service1UID ; PCAB_AddService( “InstantMessage” ; “Home Messenger” ; “Yahoo” ;
“WhoIsJoeSmith12” ) ]
Set Field [ Contacts::service2UID ; PCAB_AddService( “SocialProfile” ; “Facebook” ; “Facebook” ; “jsmith.123” ;
“http://www.facebook.com/jsmith.123" ) ]
Set Field [ Contacts::phone1UID ; PCAB_AddMV( "Phones" ; “760-510-1200” ; "work" ) ]
Set Field [ gResult ; PCAB_Save ]
```

```
If [ gResult ≠ 0 ]
   Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
   Halt Script
End If
#Retrieve Created and Modified AB Values
Set Field [ Contacts::AB_Created ; PCAB_GetValueForProperty( "Created Date" ) ]
Set Field [ Contacts::AB_Modified ; PCAB_GetValueForProperty( "Modified Date" ) ]
```

Update an Existing Record in Address Book

Updating an existing contact record in Address Book follows the same steps as adding a new contact with one exception. Instead of calling the PCAB_New function, you call the PCAB_Open function in order to open the existing record to be modified.

PCAB_Open(UID) opens an existing record in Address Book for editing or reading. The Address Book must be available to the FileMaker solution before a record can be accessed using this function.

An example script updating an existing contact is:

```
...
Set Field [ Contacts::ABID ; PCAB_Open( “SomeABID” ) ]
If [ Contacts::ABID = "!!ERROR!!" ]
   Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
   Halt Script
End If
Set Field [ Main::gResult ; PCAB_SetValueForProperty( "First Name" ; “Joe” ) ]
Set Field [ Main::gResult ; PCAB_SetValueForProperty( "Last Name" ; “Smith” )]
Set Field [ Main::gResult ; PCAB_SetImage( someTable::somePicture )]
Set Field [ Contacts::addr1UID ; PCAB_AddAddress( "work" ; "123 Any Street" ; "AnyTown" ;"AnyState";
"12345"; "USA"; "us" ) ]
Set Field [ Contacts::service1UID ; PCAB_AddService( “InstantMessage” ; “Home Messenger” ; “Yahoo” ;
“WhoIsJoeSmith12” ) ]
Set Field [ Contacts::service2UID ; PCAB_AddService( “SocialProfile” ; “Facebook” ; “Facebook” ; “jsmith.123” ;
“http://www.facebook.com/jsmith.123" )]
Set Field [ Contacts::phone1UID ; PCAB_AddMV( "Phones" ; “760-510-1200” ; "work" ) ]
Set Field [ Contacts::phone2UID ; PCAB_AddMV( "Phones" ;”760-123-4567” ; "home" ) ]
Set Field [ gResult ; PCAB_Save ]
If [ gResult ≠ 0 ]
   Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
   Halt Script
End If
#Retrieve Created and Modified AB Values
```

```
Set Field [ Contacts::AB_Created ; PCAB_GetValueForProperty( "Created Date" ) ]
Set Field [ Contacts::AB_Modified ; PCAB_GetValueForProperty( "Modified Date" ) ]
...
```

Delete an Existing Record in Address Book

You also have the option to delete existing contact records in Address Book. The PCAB_Delete(UID) function can be used to perform this action. You first need to open the record to be deleted, then call the PCAB_Delete function as shown below.

```
...
Set Field [ Contacts::ABID ; PCAB_Open( “SomeABID” ) ]
If [ Contacts::ABID = "!!ERROR!!" ]
   Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
   Halt Script
End If
Set Field [ gResult ; PCAB_Delete( “SomeABID” )]
If [ gResult ≠ 0 ]
   Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
   Halt Script
End If
...
```

The PCAB_ClearAll function can also be used to remove/delete a person or group record according to which group they belong. Handling groups will be discussed later in this document. The PCAB_ClearAll function can also be used to permanently delete all Address Book records (including the record marked as “Me”) from the Address Book by setting the parameters of this function to PCAB_ClearAll(“” ; “” ; “False”).

Deleting single or all records should be used with caution as deletions from Address Book are permanent and cannot be recovered by the plug-in.

C. Search Address Book Records

Searching for specific Address Book records is a powerful tool offered by the plug-in. The PCAB_Search(Type ; Property ; Label ; Key ; Value ; Comparison ; optExpandConstrain) function easily locates records that contain certain property values and is an efficient tool for finding records in the Address Book. On success the PCAB_Search function will return the number of records meeting the search criteria. This value returned is then used to determine if any records are in the found set. If there are records in the found set then you can begin iteration through the found set, opening each record for reading and writing. The plug-in retains the resulting found record set in memory for access by the PCAB_OpenFirstRecord and PCAB_OpenNextRecord functions.

For example, use this function to search for records modified after a specific time interval or search for contacts by group name, zip code or other specified criteria. The optExpandConstrain parameter allows for the function to expand or constrain the last search results giving you the power to perform a multi-valued search. This function proves quite useful when creating a bi-directional exchange, as you can specify your specific search criteria.

An example script of searching is:

```
...
Set Field [ gResult ; PCAB_Search("Person" ; "Modified Date" ;"" ;""; 3600 ; "WithinIntervalAroundToday" ; “” ) ;
If [gResult > 0 ]
Set Field [ gResult ; PCAB_OpenFirstRecord ]
Loop
Exit Loop If [LeftWords( gResult ; 1 )="!!ERROR!!" or LeftWords( gResult ; 1 )="END" )  ]
#script steps to pull property/field values
...
#done with getting fields, so open next record
Set Field [ gResult ; PCAB_OpenNextRecord ]
End Loop
...
```

D. Pull from Address Book

You also have the option to pull information from Address Book into FileMaker. The Functions Guide lists a complete description, parameters, and return values for each function. For the sake of brevity, we will cover only the basics of each function.

A list of all the functions necessary to pull records from Address Book are:
PCAB_Open(UID)
PCAB_Search(Type ; Property ; Label ; Key ; Value ; Comparison ; optExpandConstrain)
PCAB_OpenFirstRecord(optType)
PCAB_OpenNextRecord(optType)
PCAB_GetValueForProperty(Property ; optType)
PCAB_OpenFirstMVProperty(Property ; optType)
PCAB_OpenNextMVProperty
PCAB_GetLabelForUID(Property ; UID)
PCAB_GetValueForUID(Property ; UID ; optKey)

When pulling from Address Book the developer needs to first open a record in Address Book before its properties can be accessed. There are two different ways to open Address Book records, directly using PCAB_Open, or indirectly using PCAB_Search together with PCAB_OpenFirstRecord and PCAB_OpenNextRecord.

Once the desired record is open, then we can get the properties values of that record. Pulling the values of the different properties may require the use of a single function or a combination of functions depending on the type of property being accessed. Properties of Address Book Records are divided into two types: single valued properties and multi-valued properties. Single valued properties are very easy to access and require only the PCAB_GetValueForProperty function.

The PCAB_GetValueForProperty(Property ; optType) will return the value stored in the single valued property identified by the "Property" parameter. See the accompanying Functions Guide for a list of valid single valued properties.

Multi-valued properties require a bit more work to get at the data because Address Book holds a unique identifier for each value stored in the multi-valued property. Accessing the value (and its label) stored in a multi-valued property requires knowing what the identifier is for the particular value you desire. Gathering these identifiers is accomplished with these two functions: PCAB_OpenFirstMVProperty and PCAB_OpenNextMVProperty.

The PCAB_OpenFirstMVProperty(Property ; optType) returns the identifier for the first value stored in the property identified by "Property." See the Functions Guide for a list of valid multi-valued properties.

The PCAB_OpenNextMVProperty returns the identifier for the next value stored in the property of the active property. The active property is set with the PCAB_OpenFirstMVProperty function.

Once the identifier for the values stored in the property are returned, then the developer can access the values and labels stored in that property. This is done with the PCAB_GetLabelForUID and PCAB_GetValueForUID functions.

The PCAB_GetLabelForUID(Property ; UID) returns the label (home , work, etc.) for the value identified by the "UID" parameter. "Property" is the name of the property that is holding the UID.

The PCAB_GetValueForUID(Property ; UID ; optKey) returns the value identified by the "UID" parameter. "Property" is the property holding the UID. "optKey" is optional and is used only when the property is an Address.

An example script of pulling a contact is:

```
#perform a search
...
#open the record
Set Field [ Contacts::ABID ; PCAB_OpenFirstRecord ]
#
#get a couple single valued  properties
Set Field [ Contacts::NameFirst ; PCAB_GetValueForProperty( "First Name" ) ]
Set Field [ Contacts::NameLast ; PCAB_GetValueForProperty( "Last Name" ) ]
#
#get the UIDs for the 2 addresses and 2 phones
Set Field [ Contacts::Addr1UID ; PCAB_OpenFirstMVProperty( "Address" )]
Set Field [ Contacts::Addr2UID ; PCAB_OpenNextMVProperty ]
Set Field [ Contacts::ph1UID ; PCAB_OpenFirstMVProperty( "Phones" ) ]
Set Field [ Contacts::ph2UID ; PCAB_OpenNextMVProperty ]
#
#get first Address
Set Field [ Contacts::addrLabel1 ; PCAB_GetLabelForUID( "Address" ; Contacts ::Addr1UID )]
Set Field [ Contacts ::Street1 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr1UID ; "Street" )]
Set Field [ Contacts::City1 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr1UID ; "City" )]
Set Field [ Contacts::State1 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr1UID ; "State" )]
Set Field [ Contacts::ZIP1 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr1UID ; "Zip" )]
#
#get second address
Set Field [ Contacts::addrLabel2 ; PCAB_GetLabelForUID( "Address" ; Contacts ::Addr2UID )]
Set Field [ Contacts::Street2 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr2UID ; "Street" )]
```

```
Set Field [ Contacts::City2; PCAB_GetValueForUID( "Address" ; Contacts ::Addr2UID ; "City" )]
#
#get an instant messenger service
Set Field [ Contacts::imServiceLabel ; PCAB_GetLabelForUID( "InstantMessage" ; Contacts ::service1UID )]
Set Field [ Contacts ::imServiceUsername ; PCAB_GetValueForUID( "InstantMessage" ; Contacts :: service1UID ;
"InstantMessageUsername" )]
Set Field [ Contacts::imServiceType ; PCAB_GetValueForUID( "InstantMessage" ; Contacts :: service1UID ;
"InstantMessageService" )]
#
#get a social profile service
Set Field [ Contacts::spServiceLabel ; PCAB_GetLabelForUID( "SocialProfile" ; Contacts ::service2UID )]
Set Field [ Contacts ::spServiceUsername ; PCAB_GetValueForUID( "SocialProfile" ; Contacts :: service2UID;
"SocialProfileUsername" )]
Set Field [ Contacts::spServiceType ; PCAB_GetValueForUID( "SocialProfile" ; Contacts :: service2UID ;
"SocialProfileService" )]
Set Field [ Contacts::spServiceURL ; PCAB_GetValueForUID( "SocialProfile" ; Contacts :: service2UID;
"SocialProfileURL" )]
#
#get first phone
Set Field [ Contacts::phLabel1 ; PCAB_GetLabelForUID( "Phones" ; Contacts ::ph1UID )]
Set Field [ Contacts::phone1 ; PCAB_GetValueForUID( "Phones" ; Contacts ::ph1UID )]
#
#get second phone
Set Field [ Contacts::phLabel2; PCAB_GetLabelForUID( "Phones" ; Contacts ::ph2ID )]
Set Field [ Contacts::phone2 ; PCAB_GetValueForUID( "Phones" ; Contacts ::ph2UID )]
#
#all done
#
```

E. Syncing with Address Book Manipulator

When syncing a contact with Address Book from FileMaker, the general practice is to store the Address Book ID ("ABID") from the contact in a text field in FileMaker. This Address Book ID is a reference to that contact; if you want to open the contact in memory or display the contact in Address Book, you would pass this ABID to say "This is the contact I want to open" or "This is the contact I want to display". You would also use this ABID when assigning or removing contacts from Groups, and Group records themselves in Address Book also have an ABID of their own.

We have experienced situations that can arise when syncing an Address Book contact that are stored in an online account. In some instances, the contact's ABID will change as a result of an online sync. This, in turn, will disconnect the link that FileMaker was originally maintaining; for example, FileMaker has one ABID for contact "John Smith", but Address Book has an entirely different ABID for "John Smith". This appears to be more common with Google Contacts accounts, but iCloud also can experience this particular problem.

There are two different methods that can be used to resolve this particular kind of problem: Syncing based off of a custom field, and syncing based off of a multi-value property.

Custom Field-Based Syncing

One of the methods we recommend is to make use of a custom field to store the FileMaker record ID (which is unique per record), and utilize that as the primary way to acquire the correct ABID for a contact or group. In such a case, the process would perform PCAB_Search and provide the custom field's name and the record ID, open the record that is returned, and acquire the ABID. This would then be used to either read more data into FileMaker, open the record to push data from FileMaker, display in Address Book, etc.

Multi-Value Property Syncing

In situations where a custom field would not be suitable (such as syncing with an iCloud account), another sync method involves using a multi-value property field to store a FileMaker identifier. For example, the developer can make use of the Notes, URL, Related Contacts, or Email Address fields. In general, it would be good practice to use the URL field, as that is a multi-value property and can have a custom label such as "FMID" to denote that the contents of that url field store the FileMaker ID. Consideration should be made if the contacts are shared, however; non-custom fields that store a FileMaker ID are editable, and can be overwritten if users set data into those fields from their devices.

6) Handling Groups

The plug-in allows you to add, modify, and remove groups and subgroups. The functions described in this section can all be found in the Functions Guide that accompanies this document. The Functions Guide lists a complete description, parameters, and return values for each function.

When handling groups, you must first find the desired group, open the group record and then modify the group as needed. You locate the group record using the PCAB_Search function. For example, PCAB_Search("Group" ; "Name" ; "" ; "" ; "Sales" ; "Equal" ; “”) will find the group with the name "Sales". The search will return the number of records found that match the search criteria. Next you use the PCAB_OpenFirstRecord(“group”) function to get the Address Book ID of the group record. With the group record open, you can now add a person to this group using the PCAB_AddPerson(UID) function or you can remove a person from this group using PCAB_RemovePerson(UID) function.

The PCAB_ClearAll(UID ; Whom ; ExcludeMe) function can also be used to delete persons or group records according to the group that they belong. Please be cautious when permanently deleting records as once deleted the plug-in does not have the ability to recover records.

In order to obtain a list of all group names, then perform a search for PCAB_Search("Group" ; “Creation Date” ; “” ; “” ; “1” ; “NotWithinIntervalAroundToday”) as all records have a creation date so this search will return the number of found group records in Address Book. The use PCAB_OpenFirstRecord(“Group”) and PCAB_OpenNextRecord(“Group”) to obtain a list of the all group IDs. Next use PCAB_Open(“someGroupID”) and PCAB_GetValueProperty(“Name” ; “”) to iterate through all the groups to obtain a list of all group names in your Address Book. Since we do not yet have a “get group names” function, this is an alternate way to obtain a complete list of all group names. There are many ways to work with groups depending on your needs, so let’s get creative and design a solution to meet all your needs.

Sample script of how to create a new group and add an existing person to the new group:

```
...
Set Field [ Groups::ABID ; PCAB_New( "Group" ; “Marketing” ) ]
If [ Groups::ABID = "!!ERROR!!" ]
   Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
   Halt Script
End If
Set Field [ Main::gResult ; PCAB_AddPerson( $PersonUID ) ]
Set Field [ gResult ; PCAB_Save ]
If [ gResult ≠ 0 ]
   Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
   Halt Script
End If
...
```

Sample script of how to delete a person from an existing group:

```
...
Set Field [ gResult ; PCAB_Search( "Group" ; "Name" ; "" ; "" ; "Sales" ; "Equal" ; “” ) ] ;
If [gResult > 0 ]
Set Field [ Groups::ABID ; PCAB_OpenFirstRecord( “Group” )]
Loop
  Exit Loop If [LeftWords( Groups::ABID ; 1 )="!!ERROR!!" or LeftWords( Groups::ABID ; 1 )="END" ) ]
  Set Field [ gResult ; PCAB_RemovePerson( $PersonUID )]
  If [ gResult ≠ 0 ]
     Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
     Halt Script
  End If
  #open next record (if multiple “Sales” groups exist)
  Set Field [ gResult ; PCAB_OpenNextRecord( “Group” ) ]
End Loop
...
```

Sample script of how to **permanently** delete all persons from Address Book in a specific group:

```
...
Set Field [ gResult ; PCAB_Search( "Group" ; "Name" ; "" ; "" ; "Sales" ; "Equal" ; “” ) ;
If [gResult > 0 ]
Set Field [ Groups::ABID ; PCAB_OpenFirstRecord( “group” ) ]
Loop
  Exit Loop If [LeftWords( Groups::ABID ; 1 )="!!ERROR!!" or LeftWords( Groups::ABID ; 1 )="END" )  ]
  Set Field [ gResult ; PCAB_ClearAll( Groups::ABID ; “Members” ; “True” ) ]
  If [ gResult ≠ 0 ]
    Show Custom Dialog[ “Error” ; PCAB_GetLastError( “text” ) ]
    Halt Script
  End If
  #open next record (if multiple “Sales” groups exist)
  Set Field [ Groups::ABID ; PCAB_OpenNextRecord( “Group” ) ]
End Loop
...
```

There are various other functions relating to groups and subgroups such as:

PCAB_GetFirstMemberID
PCAB_GetNextMemberID

PCAB_AddSubgroup(UID)
PCAB_RemoveSubgroup(UID)
PCAB_GetFirstSubgroupID
PCAB_GetNextSubgroupID

PCAB_GetListOfValues(RecordType ; GroupOrMember ; FieldName)

PCAB_OpenFirstMVProperty(Property ; “group”)
PCAB_OpenNextMVProperty
PCAB_SetValueForProperty(Property ; value ; “group”)
PCAB_DeleteValueForProperty(Property ; “group”)
PCAB_GetValueForProperty(Property ; “group”)

The accompanying Functions Guide will provide you with further details of each of these functions.

7) Working with Custom Fields

The plug-in has the ability to add a custom field to a group or person record in Apple Address Book. A word of caution is that custom fields in Address Book are not visible in the Address Book user interface, cannot be deleted in Address Book and are not synced with any other application such as MobileMe, etc. This limitation is imposed by Apple rather than the plug-in. All custom fields created with this function will only be of single valued string type. A sample use of why you may desire to use the custom field would be to store a FileMaker ID to an Address Book contact record.

The plug-in functions pertaining to custom fields are:
PCAB_AddCustomField(RecordType ; FieldName)
PCAB_GetCustomField(RecordType ; FieldName)
PCAB_SetCustomField(RecordType ; FieldName ; Value)

Please see the accompanying Functions Guide for a complete description, parameters, and return values for each function.

8) Handling Errors

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.

Typically the plug-in functions will return a 0 for a success. However, any of the plug-in functions may encounter an error during processing and will return the !!ERROR!! string. When an !!ERROR!! occurs during processing, immediately call the PCAB_GetLastError(optType) function in order to obtain a full description of the error. This function returns a textual or numeric description of the last error used to help troubleshoot script or logic failures.

This makes it simple to check for errors. If a plug-in function does not return a 0 or returns !!ERROR!!, then immediately after call PCAB_GetLastError function for a detailed description of what the exact error was. In order to determine the to trap for 0 or !!ERROR!! in your scripts, please refer to the Functions Guide for the exact return values of each function. Here are a few samples of how you can check for errors.

Example 1:
Set Field [Main::gResult = PCAB_SomePluginFunction(“a” ; “b”)]
If [Main::gResult = “!!ERROR!!”]
Show Custom Dialog [“Error:“ & PCAB_GetLastError(“text”)]
End If

Example 2:
Set Field [Main::gResult = PCAB_SomePluginFunction(“a” ; “b”)]
If [Main::gResult ≠ 0]
Show Custom Dialog [“Error: “ & PCAB_GetLastError(“number”)]
End If

It is good practice to ALWAYS trap for errors.

A. Error/Return Codes

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

Error Number	Error Text
-10	Failed Registration
-4	Invalid Parameter value(s)
-3	Invalid # of Parameters
-1	Plug-in not registered or session expired
0	Success
8000	Exception caught in function
8001	Record does not exist
8002	Failed to delete record
8003	No Active record
8004	Invalid Call Order
8005	Failed to Save Record
8006	Invalid Property type
8007	UID Does Not Exist
8008	Invalid value for Key
8009	Property is not an Address
8010	Invalid Property Name
8011	Error in Property
8012	Error setting property
8013	Error replacing Multi-value array
8014	Error setting label
8015	Failed to create address
8016	Failed to add property
8017	Record is read-only
8018	Invalid record type
8019	Multi-value array does not exist
8020	Unable to create property
8021	No properties added
8022	Unable to remove property
8023	No properties removed
8024	Unable to set custom property
8025	Unable to obtain bitmap of image
8026	Unable to set image date
8027	Unsupported image type
8028	Unable to access label
8029	Property value cannot be empty
8030	UID is missing and is required for this function
8031	Invalid Country Code passed to function
8032	Address Book not started
8033	Invalid service type specified
8034	Failed to create service

9) Known Issues

- **Multi-Value Properties**

  When pushing more than one multi-value property there was an issue in which only the first and the most recent value would be accepted for the property. This was affecting all multi-value properties. In the recent update, the issue has been resolved for all multi-value properties with the exception of the "Dates" property.

- **iCloud**

  In some cases, when working with a Contacts account that is linked to iCloud, there may be situations that cause an Address Book contact record's Address Book ID to be rewritten after Contacts syncs to iCloud. This causes the contact to have a different ID than FileMaker may reference, which can cause syncing issues.

  We recommend using one of the two options listed in the "Syncing with Address Book Manipulator" section above in order to mitigate the potential problems this situation may cause.

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 that 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!