



Address Book@ Manipulator

Developer's Guide

Table of Contents

I. Introduction	3
II. Integration Steps	4
1) Installing and Registering the Plug-in	4
2) Creating Integration Scripts	5
<i>A. Push to Address Book</i>	<i>5</i>
<i>Add a New Record to Address Book</i>	<i>5</i>
<i>Update an Existing Record in Address Book</i>	<i>7</i>
<i>Delete an Existing Record in Address Book</i>	<i>8</i>
<i>B. Search Address Book Records</i>	<i>9</i>
<i>C. Pull from Address Book</i>	<i>10</i>
3) Handling Groups.....	13
4) Working with Custom Fields.....	16
5) Handling Errors	17
III. Contact us.....	18

I. Introduction

Description:

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

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 from: <http://www.addressbookmanipulator.com/downloads/AddressBookManipv3.zip>
- 4) Watch video tutorials found here: <http://www.productivecomputing.com/video>
- 5) Familiarize yourself with Apple's Address Book

II. Integration Steps

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

1) Installing and Registering the Plug-in

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 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 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

Registration:

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 the FileMaker Demo file on the Setup tab.
- 7) If you are registering a licensed copy, then simply enter your license number in the "License ID" 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 noted in the upper right hand corner of our FileMaker Demo file on the Setup tab.

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.

2) Creating Integration Scripts

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.

A. 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. Let's look at the functions involved 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 AddressBook, setting the values for the different properties and then saving the new AddressBook record.

A list of all the functions necessary to create and populate a new AddressBook record are:

PCAB_New(RecordType ; optType)

PCAB_SetValueForProperty(Property ; Value ; optType)

PCAB_SetImage(BinaryData)

PCAB_AddAddress(Label ; Street ; City ; State ; Zip ; Country ; CountryCode)

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 a 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 e-mail 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_Save saves the record and its contents to the Apple Address Book.

An example script adding a new contact is:

...

```
SetField[ Contacts::ABID ; PCAB_New( "Person" ) ]
If [ Contacts::ABID = "!!ERROR!!" ]
    Show Custom Dialog[ "Error" ; PCAB_GetLastError( "text" ) ]
    Halt Script
End If
SetField[ Main::gResult ; PCAB_SetValueForProperty( "First Name" ; "Joe" ) ]
SetField[ Main::gResult ; PCAB_SetValueForProperty( "Last Name" ; "Smith" ) ]
SetField[ Main::gResult ; PCAB_SetImage( someTable::somePicture ) ]
SetField[ Contacts::addr1UID ; PCAB_AddAddress( "work" ; "123 Some Street" ; "AnyTown" ; "AnyState" ; "12345" ; "USA" ;
"us" ) ]
SetField[ Contacts::phone1UID ; PCAB_AddMV( "Phones" ; "760-510-1200" ; "work" ) ]
SetField[ gResult ; PCAB_Save ]
If [ gResult ≠ 0 ]
    Show Custom Dialog[ "Error" ; PCAB_GetLastError( "text" ) ]
    Halt Script
End If
#Retrieve Created and Modified AB Values
SetField[ Contacts::AB_Created ; PCAB_GetValueForProperty( "Created Date" ) ]
SetField[ 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:

```
...
SetField[ SetField[ Contacts::ABID ; PCAB_Open( "SomeABID" ) ]
If [ Contacts::ABID = "!!ERROR!!" ]
    Show Custom Dialog[ "Error" ; PCAB_GetLastError( "text" ) ]
    Halt Script
End If
SetField[ Main::gResult ; PCAB_SetValueForProperty( "First Name" ; "Joe" ) ]
SetField[ Main::gResult ; PCAB_SetValueForProperty( "Last Name" ; "Smith" ) ]
SetField[ Main::gResult ; PCAB_SetImage( someTable::somePicture ) ]
SetField[ Contacts::addr1UID ; PCAB_AddAddress( "work" ; "123 Any Street" ; "AnyTown" ; "AnyState"; "12345"; "USA";
"us" ) ]
SetField[ Contacts::phone1UID ; PCAB_AddMV( "Phones" ; "760-510-1200 ; "work" ) ]
SetField[ Contacts::phone2UID ; PCAB_AddMV( "Phones" ; "760-123-4567 ; "home" ) ]
SetField[ gResult ; PCAB_Save ]
If [ gResult ≠ 0 ]
    Show Custom Dialog[ "Error" ; PCAB_GetLastError( "text" ) ]
    Halt Script
End If
#Retrieve Created and Modified AB Values
SetField[ Contacts::AB_Created ; PCAB_GetValueForProperty( "Created Date" ) ]
SetField[ 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.

```
...
SetField[ SetField[ Contacts::ABID ; PCAB_Open( "SomeABID" ) ]
If [ Contacts::ABID = "!!ERROR!!" ]
    Show Custom Dialog[ "Error" ; PCAB_GetLastError( "text" ) ]
    Halt Script
End If
SetField[ 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.

B. 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 now 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:

```
...
SetField[ gResult ; PCAB_Search("Person" ; "Modified Date" ; "" ; "" ; 3600 ; "WithinIntervalAroundToday" ; "" ) ;
If [gResult > 0 ]
SetField[ 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
SetField[ gResult ; PCAB_OpenNextRecord ]
End Loop
...
```

C. Pull from Address Book

You also have the option to pull information from Address Book into FileMaker. The Functions Guide lists the 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 AddressBook 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 value 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
SetField[ Contacts::ABID ; PCAB_OpenFirstRecord ]
#
#get a couple single valued properties
SetField[ Contacts::NameFirst ; PCAB_GetValueForProperty( "First Name" ) ]
SetField[ Contacts::NameLast ; PCAB_GetValueForProperty( "Last Name" ) ]
#
#get the UIDs for the 2 addresses and 2 phones
SetField[ Contacts::Addr1UID ; PCAB_OpenFirstMVProperty( "Address" ) ]
SetField[ Contacts::Addr2UID ; PCAB_OpenNextMVProperty ]
SetField[ Contacts::ph1UID ; PCAB_OpenFirstMVProperty( "Phones" ) ]
SetField[ Contacts::ph2UID ; PCAB_OpenNextMVProperty ]
#
#get first Address
SetField[ Contacts::addrLabel1 ; PCAB_GetLabelForUID( "Address" ; Contacts ::Addr1UID ) ]
SetField[ Contacts ::Street1 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr1UID ; "Street" ) ]
SetField[ Contacts::City1 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr1UID ; "City" ) ]
SetField[ Contacts::State1 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr1UID ; "State" ) ]
SetField[ Contacts::ZIP1 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr1UID ; "Zip" ) ]
#
#get second address
SetField[ Contacts::addrLabel2 ; PCAB_GetLabelForUID( "Address" ; Contacts ::Addr2UID ) ]
SetField[ Contacts::Street2 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr2UID ; "Street" ) ]
SetField[ Contacts::City2 ; PCAB_GetValueForUID( "Address" ; Contacts ::Addr2UID ; "City" ) ]
#
#get first phone
SetField[ Contacts::phLabel1 ; PCAB_GetLabelForUID( "Phones" ; Contacts ::ph1UID ) ]
SetField[ Contacts::phone1 ; PCAB_GetValueForUID( "Phones" ; Contacts ::ph1UID ) ]
#
#get second phone
SetField[ Contacts::phLabel2 ; PCAB_GetLabelForUID( "Phones" ; Contacts ::ph2UID ) ]
SetField[ Contacts::phone2 ; PCAB_GetValueForUID( "Phones" ; Contacts ::ph2UID ) ]
#
#all done
#
```

3) 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 the 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, which they belong. Please be cautious when permanently deleting records as once deleted the plug-in does not have the ability to recover records.

Since we do not offer have a FileMaker demo of how to work with groups and subgroups, please find various sample scripts below.

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

```
...
SetField[ Groups::ABID ; PCAB_New( "Group" ; "Marketing" ) ]
If [ Groups::ABID = "!!ERROR!!" ]
    Show Custom Dialog[ "Error" ; PCAB_GetLastError( "text" ) ]
    Halt Script
End If
SetField[ Main::gResult ; PCAB_AddPerson( "SomeABID" ) ]
SetField[ 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 a existing group:

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

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

```
...
SetField[ gResult ; PCAB_Search("group" ; "Name" ; "" ; "" ; "Sales" ; "Equal" ; "" ) ;
If [ gResult > 0 ]
SetField[ Groups::ABID ; PCAB_OpenFirstRecord( "group" ) ]
Loop
  Exit Loop If [ LeftWords( Groups::ABID ; 1 )="!!ERROR!!" or LeftWords( Groups::ABID ; 1 )="END" ]
  SetField[ 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)
  SetField[ 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.

4) 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.

5) 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.

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!