Productive Computing

iCal Manipulator

Developer's Guide

Revised June 8, 2017

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

Table of Contents

I. Introduction

Description

The iCal Manipulator plug-in offers functions that support a bi-directional data exchange between FileMaker® and Apple® Calendar. With this plug-in FileMaker users are able to push new records into Apple Calendar, pull records from Apple Calendar, and modify existing records in Apple Calendar. These operations are accomplished by using FileMaker function calls from within FileMaker calculations. These calculations are generally determined from within FileMaker "Set Field" or "If" script steps.

Product Version History

http://www.productivecomputing.com/ical-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 Guide
3) Download a demo: http://www.productivecomputing.com/dl/iCal_Manipulator/iCalManip.zip
4) Watch video tutorials: http://productivecomputing.com/video/
5) Familiarize yourself with Apple Calendar and Reminders

II. Integration Steps

Accessing and using the plug-in involves the following steps:

1) Prerequisites for Plug-in Installation

New plug-in version 2.0.0.0 iCal Manipulator for Mac is available in 64-bit mode only, requiring FileMaker Pro 14 or later.

2) Installing the Plug-in with the Installer

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

Note: 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, FileMaker Demo File, and plug-ins.

3) Installing the Plug-in with the Demo File

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

FileMaker Pro 14:

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

2) Select the "Install" button.

4) 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.

5) 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 "License ID" field and select the "Register" button. Ensure you have removed the Demo License ID and enter your registration information exactly as it appears in your confirmation email. Your plug-in should now be running in "LIVE" mode. The mode is always noted on the Setup tab of the FileMaker demo, or by calling the PCIM_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-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 PCIM_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 PCIM_GetOperatingMode and PCIM_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 PCIM_Register, PCIM_GetOperatingMode and PCIM_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 [ PCIM_Version( "short" ) = "" or PCIM_Version( "short" ) = "?" ]
    Show Custom Dialog [ Title: "Warning"; Message: "Plug-in not installed."; Buttons: "OK" ]
If [ PCIM_GetOperatingMode  ≠ "LIVE" ]
    Set Field [Main::gRegResult; PCIM_Register( "licensing.productivecomputing.com" ; "80" ; "/PCIReg/pcireg.php" ;
"your license ID" )
If  [ Main::gRegResult   ≠  0 ]
    Show Custom Dialog [ Title: "Registration Error"; Message: "Plug-in Registration Failed"; Buttons: "OK" ]
```

7) FileMaker 16 Plug-in Script Steps

Newly introduced in FileMaker Pro 16, all plug-ins have been updated to allow a developer to specify plug-in functions as script steps instead of as calculation results. The plug-in script steps function identically to calling a plug-in within a calculation dialog.

In this example, we use the FM Books Connector plug-in's script steps to demonstrate the difference. The same scripting differences would be found for any of the Productive Computing plug-in product lines.

For an example of using plug-in script steps, compare two versions of the same script from the FM Books Connector demo file: Pull Customer__Existing Session.

Script 1 - Pull Customer__Existing Session with calculation ("traditional") plug-in scripting:

```
Set Error Capture [On]
Allow User Abort [Off]
# It is assumed the session is already opened from the previous script calling this
script.
# Query customers in QB (Request)

Set Variable [$$Result; Value: PCQB_RqNew( "CustomerQuery" ; "" )]
Set Variable [$$Result; Value: PCQB_RqAddFieldWithValue( "ListID" ;
Main::gCust_ListID )]
If [0 <> PCQB_RqExecute]
      Exit Script [Text Result:PCQB_SGetStatus]
End If

# Pull customer info into FileMaker (Response)
Set Variable [$$Result; Value: PCQB_RsOpenFirstRecord]
Set Field [main_CUST__Customers::ListID; PCQB_RsGetFirstFieldValue( "ListID" )]
Set Field [main_CUST__Customers::FullName; PCQB_RsGetFirstFieldValue( "FullName" )]
Set Field [main_CUST__Customers::First Name;
PCQB_RsGetFirstFieldValue( "FirstName" )]
Set Field [main_CUST__Customers::Last Name; PCQB_RsGetFirstFieldValue( "LastName" )]
Set Field [main_CUST__Customers::Company; PCQB_RsGetFirstFieldValue( "CompanyName" )]
Set Field [main_CUST__Customers::Bill_Address 1;
PCQB_RsGetFirstFieldValue( "BillAddress::Addr1" )]
Set Field [main_CUST__Customers::Bill_Address 2;
PCQB_RsGetFirstFieldValue( "BillAddress::Addr2" )]
Set Field [main_CUST__Customers::Bill_Address 3;
PCQB_RsGetFirstFieldValue( "BillAddress::Addr3" )]
Set Field [main_CUST__Customers::Bill_Address 4;
PCQB_RsGetFirstFieldValue( "BillAddress::Addr4" )]
Set Field [main_CUST__Customers::Bill_City;
PCQB_RsGetFirstFieldValue( "BillAddress::City" )]
Set Field [main_CUST__Customers::Bill_State;
PCQB_RsGetFirstFieldValue( "BillAddress::State" )]
Set Field [main_CUST__Customers::Bill_Postal Code;
PCQB_RsGetFirstFieldValue( "BillAddress::PostalCode" )]
Set Field [main_CUST__Customers::Phone; PCQB_RsGetFirstFieldValue( "Phone" )]
Set Field [main_CUST__Customers::Email; PCQB_RsGetFirstFieldValue( "Email" )]

Exit Script [Text Result:0]
```

Script 2 – Pull Customer__Existing Session with plug-in script steps:

```
Set Error Capture [On]
Allow User Abort [Off]
# It is assumed the session is already opened from the previous script calling this
script.
# Query customers in QB (Request)

PCQB_RqNew [Select; Results:$$Result; Request Type:"CustomerQuery"]
PCQB_RqAddFieldWithValue [Select; Results:$$Result; QB Field Name:"ListID"; Field
Value:Main::gCust_ListID]
PCQB_RqExecute [Select; Results:$$Result]
If [$$Result <> 0]
      Exit Script [Text Result:PCQB_SGetStatus]
End If

# Pull customer info into FileMaker (Response)
PCQB_RsOpenFirstRecord [Select; Results:$$Result]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::ListID; Field
Name:"ListID"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::FullName;
FieldName:"FullName"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::First Name; Field
Name:" FirstName"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Last Name; Field
Name:" LastName"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Company; Field
Name:" CompanyName"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 1;
Field Name:" BillAddress::Addr1"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 2;
Field Name:" BillAddress::Addr2"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 3;
Field Name:" BillAddress::Addr3"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 4;
Field Name:" BillAddress::Addr4"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_City; Field
Name:" BillAddress::City"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_State; Field
Name:" BillAddress::State"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Postal Code;
Field Name:" BillAddress::PostalCode"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Phone; Field
Name:"Phone"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Email; Field
Name:"Email"]

Exit Script [Text Result:0]
```

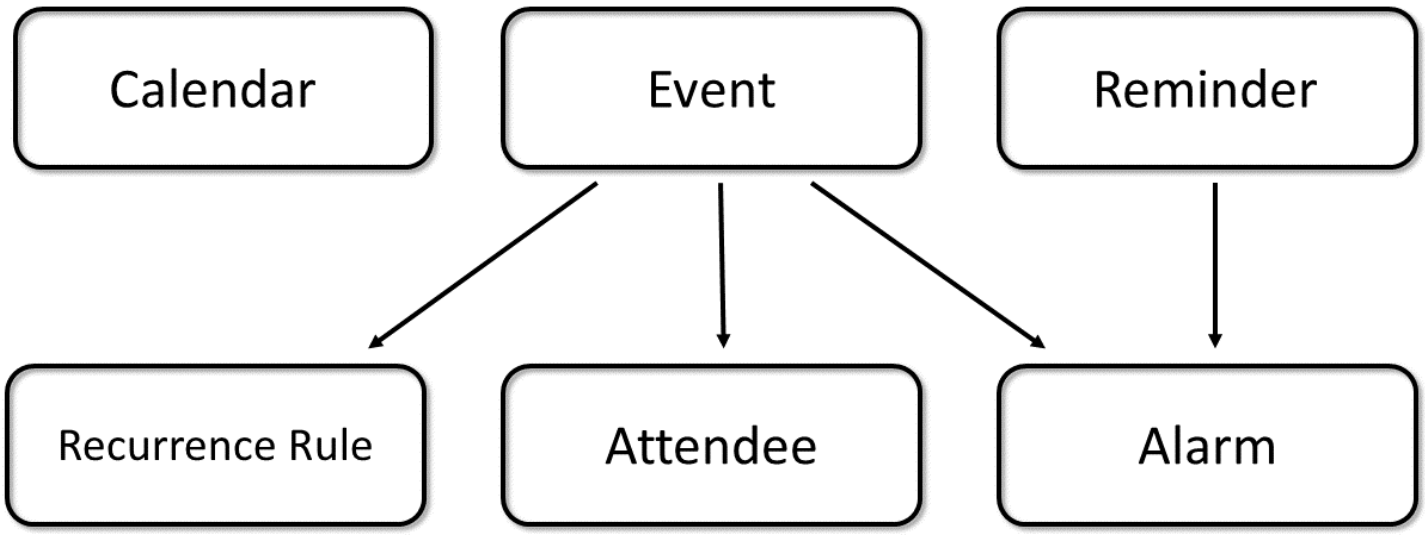
Using script steps instead of the more traditional methods can make scripting within a solution more direct, as well as help with data entry validation. Some functions accept calculation-style input, while others accept a Boolean "true" or "false" option, and others employ a drop-down list for the developer to choose an option from. As stated earlier, the functionality of the plug-in script step is identical to its functionality as a calculation function; PCQB_RsOpenFirstRecord as a script step will still open the first record in the response, and store the value in the $$Result global variable (as seen in Script 2), just the same as the Set Variable script step calls PCQB_RsOpenFirstRecord (which opens the first response record) and stores the result in the $$Result variable.

For all Productive Computing, Inc., plug-ins that provide plug-in script step functionality, calculation functions will still be provided for use in development. This is to ensure that scripts already integrated with any of our plug-ins will still be viable and functional, and the developer now has the option to utilize the plug-in script steps at their discretion.

7) Talking to Apple Calendar

Moving information between FileMaker and Apple Calendar/Apple Reminders involves pushing or pulling records from one application into the other. Using the plug-in you are able to push information from FileMaker to Apple Calendar or pull information from Apple Calendar to FileMaker.

First it is necessary to understand the basic Apple Calendar schema as shown below.



The three basic item types in Apple Calendar are calendar, event, and reminder.
A calendar item represents a group of events and reminders grouped together on one calendar.
An event item represents a scheduled event that can be displayed on a calendar.
A reminder item represents something that you need to do or accomplish.

Events can have recurrences, attendees, and alarms. Reminders can have alarms.
A recurrence rules represent the details of the frequency of which an event may occur.
An attendee is someone who attends an event.
An alarm alerts the user of a pending event or reminder by performing an action.

A. Pushing a new item to Apple Calendar

Adding a new item to Apple Calendar is accomplished by creating the new item using PCIM_NewItem(type), setting the desired property values using PCIM_SetProperty(type ; property ; value) and saving the item using PCIM_SaveItem(type ; optSpan). The specific property names can be found in the Functions Guide. Let's have a look at three examples below to understand this basic concept.

Example 1 - Add New Calendar

#Create a new Apple Calendar Item
PCIM_NewItem("calendar")
#Populate Apple Calendar Record with data from fields in FIleMaker
PCIM_SetProperty("calendar" ; "title" ; SOMETITLEVALUE)
PCIM_SetProperty("calendar" ; "notes" ; SOMENOTESVALUE)
PCIM_SetProperty("calendar" ; "color" ; SOMECOLORVALUE)
#Save Record in Calendar
PCIM_SaveItem("calendar")

Example 2 - Add New Event

#Create a new Calendar Event Item
PCIM_NewItem("event")
#Populate Calendar Event Record with data from fields in FIleMaker
PCIM_SetProperty("event" ; "title" ; SOMETITLEVALUE)
PCIM_SetProperty("event" ; "isallday" ; SOMEISALLDAYVALUE)
PCIM_SetProperty("event" ; "enddate" ; SOMEENDDATEVALUE)
PCIM_SetProperty("event" ; "startdate" ; SOMESTARTDATEVALUE)
PCIM_SetProperty("event" ; "notes" ; SOMENOTESVALUE)
PCIM_SetProperty("event" ; "location" ; SOMELOCATIONVALUE)
PCIM_SetProperty("event" ; "calendartitle" ; SOMECALENDARTITLEVALUE)
#Save Record in Apple Calendar
PCIM_SaveItem("event" ; "this")

Example 3 - Add New Reminder

#Create a new Reminder Item
PCIM_NewItem("reminder")
#Populate Reminder item with data from fields in FIleMaker
PCIM_SetProperty("reminder" ; "calendartitle" ; SOMECALENDARTITLEVALUE)
PCIM_SetProperty("reminder" ; "title" ; SOMETITLEVALUE)
PCIM_SetProperty("reminder" ; "duedate" ; SOMEDUEDATEVALUE)
PCIM_SetProperty("reminder" ; "iscompleted" ; SOMEISCOMPLETEDVALUE)
PCIM_SetProperty("reminder" ; "completeddate" ; SOMECOMPLETEDDATEVALUE)
PCIM_SetProperty("reminder ; "notes" ; SOMENOTESVALUE)
PCIM_SetProperty("reminder" ; "priority" ; SOMEPRIORITYVALUE)
#Save Record in Apple Calendar
PCIM_SaveItem("reminder" ; "this")

B. Pulling from Apple Calendar to FileMaker

When pulling event and reminder records from Apple Calendar the developer will need to fetch specific records. Fetching allows you to create a "found set" of events or reminders in Apple Calendar which match the parameters. The PCIM_EventsFetch and PCIM_RemindersFetch functions are used by the plug-in to create this "found set" of records. There is no need to fetch calendars as the plug-in automatically finds all calendars. Once we have a "found" our records, then PCIM_GetItemCountForType will return the number of available items of a specified record type. PCIM_OpenItemAt or PCIM_OpenItemWithUID will open the item and PCIM_GetProperty retrieves the specified property for the currently opened item of the specified type. Let's look at the steps necessary to import various types of items.

Import Calendars

These functions used to import calendars.

- PCIM_GetItemCountForType("calendar")
- PCIM_OpenItemAt("calendar" ; index) or PCIM_OpenItemWithUID("calendar" ; uid ; optOccurrence)
- PCIM_GetProperty("calendar" ; property)

First we will get an item count using PCIM_GetItemCountForType to ensure that there are calendar records to import. Next we enter a loop and open each item using PCIM_OpenItemAt, then set all the desired calendar fields using PCIM_GetProperty and end the loop.

Example 1 - Import Calendar

```
#Import all calendar items..
$index = 1
loop
   #Exit the loop if the Index is greater than the number of calendars available...
   exit loop if $index > PCIM_GetItemCountForType( "calendar")
   #open the calendar at the Index
   PCIM_OpenItemAt( "calendar" ; $index)
   #with the calendar opened populate the filemaker calendar record with data from Apple Calendar
   PCIM_GetProperty( "calendar" ; "title" )
   PCIM_GetProperty( "calendar" ; "notes" )
   PCIM_GetProperty( "calendar" ; "color" )
   PCIM_GetProperty( "calendar" ; "uid" )
   $index = $index + 1
end loop
```

Importing calendars is quite simple and straight forward. Now let's use the PCIM_EventsFetch and PCIM_RemindersFetch functions to import events and reminders.

Import Events

These functions used to import events.

- PCIM_EventsFetch(startDate ; endDate ; calendars ; uid)
- PCIM_GetItemCountForType("event")
- PCIM_OpenItemAt("event" ; index) or PCIM_OpenItemWithUID("event" ; uid ; optOccurrence)
- PCIM_GetProperty("event" ; property)

Example 2 - Import Events

```
#Use PCIM_EventsFetch to obtain a found set of events to import.
#startDate will restrict events with a start date LATER than startDate
#endDate will restrict events with an end date EARLIER than endDate
#calendars will restrict events belonging to calendars named here
#uid is the uid for a specific event. This parameter is used if the event is a repeating event.
PCIM_EventsFetch( startDate ; endDate ; calendars ; uid )
#now import found Event Items...
$index = 1
loop
   #Exit the loop if the index is greater than the number of event available...
   exit loop if $index > PCIM_GetItemCountForType( "event")
   #open the event at the index
   PCIM_OpenItemAt( "event" ; $index)
   #with the event opened populate the filemaker event record with data from Calendar
   PCIM_GetProperty( "event" ; "title" )
   PCIM_GetProperty( "event" ; "startDate" )
   PCIM_GetProperty( "event" ; "endDate" )
   PCIM_GetProperty( "event" ; "notes" )
   PCIM_GetProperty( "event" ; "location" )
   PCIM_GetProperty( "event" ; "uid" )
   PCIM_GetProperty( "event" ; "datestamp" )
    ...
   $index = $index + 1
end loop
```

Import Reminders

These functions used to import events.

- PCIM_RemindersFetch(calendars ; status ; date)
- PCIM_GetItemCountForType("reminder")
- PCIM_OpenItemAt("reminder" ; index) or PCIM_OpenItemWithUID("reminder" ; uid ; optOccurrence)
- PCIM_GetProperty("reminder" ; property)

Example 3 - Import Reminders

```
#Use PCIM_RemindersFetch to obtain a found set of reminders to import.
#calendars is a list of calendars to be searched
#status is a string indicating the type of reminder to be found (complete, incomplete or blank)
#date indicates reminders with either a due date before or after this date depending on status
PCIM_RemindersFetch( calendars ; status ; date )
#now import found Reminder Items...
$index = 1
loop
   #Exit the loop if the index is greater than the number of Reminders available...
   exit loop if $index > PCIM_GetItemCountForType( "reminder")
   #open the Reminder at the index
   PCIM_OpenItemAt( "reminder" ; $index)
   #with the Reminder opened populate the filemaker Reminder record with data from Apple Calendar
   PCIM_GetProperty( "reminder" ; "title" )
   PCIM_GetProperty( "reminder" ; "notes" )
   PCIM_GetProperty( "reminder" ; "dueDate" )
   PCIM_GetProperty( "reminder" ; "priority" )
   PCIM_GetProperty( "reminder" ; "uid" )
   PCIM_GetProperty( "reminder" ; "datestamp" )
    ...
   $index = $index + 1
end loop
```

Now that you understand the basic steps involved in importing calendars, events, and reminders, let's explore one level deeper. Typically events will also have attendees, recurrences, and alarms. Here are the example steps involved for importing alarms and attendees. Handling recurrences will be discussed later in this document.

Import Alarms

The function used to import alarms is PCIM_GetProperty("alarm" ; property).

Example 4 - Import Alarms

```
#with an event or reminder opened we can import the alarms for that item.
#this is the same method for importing the event or reminder
$alarm_index = 1
loop
   #Exit the loop if the index is greater than the number of alarm available...
   exit loop if $alarm_index > PCIM_GetItemCountForType( "alarm")
   #open the alarm at the index
   PCIM_OpenItemAt( "alarm" ; $alarm_index)
   #with the alarm opened populate the filemaker alarm record with data from Apple Calendar
   PCIM_GetProperty( "alarm" ; "action" )
   PCIM_GetProperty( "alarm" ; "sound" )
    ...
   $alarm_index = $alarm_index + 1
end loop
```

Import Attendees

Attendees are read only. The function used to import attendees is PCIM_GetProperty("attendee" ; property).

Example 5 - Import Attendees

```
#with an event opened we can import the attendees for that item.
$attendee_index = 1
loop
   #Exit the loop if the index is greater than the number of attendees available...
   exit loop if $attendee_index > PCIM_GetItemCountForType( "attendee")
   #open the attendee at the index
   PCIM_OpenItemAt(  "attendee" ; $attendee_index)
   #with the attendee opened populate the filemaker attendee record with data from Calendar
   PCIM_GetProperty( "attendee" ; "address" )
   PCIM_GetProperty( "attendee" ; "commonName" )
   PCIM_GetProperty( "attendee" ; "status" )
    ...
   $attendee_index = $attendee_index + 1
end loop
```

C. Updating an existing record in Apple Calendar

Updating an existing item from FileMaker to Apple Calendar is accomplished by opening an existing item using PCIM_CalOpenCalendarWithTitle(title), PCIM_OpenItemWithUID(Type ; uid ; optOccurrence) or PCIM_OpenItemAt(type ; index), setting the desired property values using PCIM_SetProperty(type ; property ; value) and saving the item using PCIM_SaveItem(type ; optSpan).

The Apple Calendar UID is used to identify if the item already exists. This can be useful in determining if the item is a new item or existing item to be edited or deleted.

The examples below illustrate these various methods. Please note that PCIM_CalOpenCalendarWithTitle can only be used with calendar items to open the calendar with the calendar's name. PCIM_OpenItemWithUID can be used with calendar, event and reminder items to open the desired item with the item's UID. PCIM_OpenItemAt can be used with calendar, event, reminder, attendee and alarm items to open the item at the specified index. Once you open the desired item, then the item may be read or written to. We provide various examples below to illustrate the concept of each function. You can use your creativity and development skills to apply these concepts to your solution as needed.

Example 1 - Open Calendar with Title

```
#Open the Apple Calendar with Title
PCIM_CalOpenCalendarWithTitle( SOMECALENDARTITLE )
#Populate Apple Calendar record with data from fields in FIleMaker
PCIM_SetProperty( "calendar" ; "title" ; SOMETITLEVALUE )
PCIM_SetProperty( "calendar" ; "notes" ; SOMENOTESVALUE )
PCIM_SetProperty( "calendar" ; "color" ; SOMECOLOERVALUE )
#Save Record in Apple Calendar
PCIM_SaveItem( "calendar" )
```

Example 2 - Open Event with UID

```
#Open the Calendar Event Item with UID
PCIM_OpenItemWithUID( "event" ; SOMEUID )
#Populate Calendar Event with data from fields in FIleMaker
PCIM_SetProperty( "event" ; "enddate" ; SOMEENDDATEVALUE )
PCIM_SetProperty( "event" ; "location" ; SOMELOCATIONVALUE )
PCIM_SetProperty( "event" ; "startdate" ; SOMESTARTDATEVALUE )
PCIM_SetProperty( "event" ; "calendartitle" ; SOMECALENDARTITLEVALUE )
PCIM_SetProperty( "event" ; "notes" ; SOMENOTESVALUE )
PCIM_SetProperty( "event" ; "title" ; SOMETITLEVALUE )
PCIM_SetProperty( "event" ; "url" ; SOMEURLVALUE )
#Save Record in Apple Calendar
PCIM_SaveItem( "event" )
```

Example 3 - Open Reminder at Index:

#Open the Reminder Item at Index

PCIM_OpenItemAt(“reminder” ; $index)

#Populate Reminder record with data from fields in FIleMaker

PCIM_SetProperty(“reminder” ; “calendartitle” ; REMINDERS:: calendartitle)

PCIM_SetProperty(“reminder” ; “completeddate” ; REMINDERS:: completeddate)

PCIM_SetProperty(“reminder” ; “dueDate” ; REMINDERS:: dueDate)

PCIM_SetProperty(“reminder” ; “notes” ; REMINDERS:: notes)

PCIM_SetProperty(“reminder” ; “priority” ; REMINDERS:: priority)

PCIM_SetProperty(“reminder” ; “title” ; REMINDERS:: title)

PCIM_SetProperty(“reminder” ; “url” ; REMINDERS:: url)

#Save Record in Apple Calendar

PCIM_SaveItem(“reminder”)

D. Deleting an existing record in Apple Calendar

Deleting an existing item in Apple Calendar can be accomplished by using PCIM_RemoveItem(type ; uid ; occurrenceDate ; span) function.

Example 1 - Delete an Entire Calendar

Deleting a calendar is simple and should be used with caution. In the example below we use the PCIM_RemoveItem("calendar" ; CALENDARS::uid) function to delete the specified calendar. The type parameter is set to "calendar" and the uid parameter is set to CALENDARS::uid referencing the uid field on the CALENDARS table in the demo file.

```
# ------------------------------------------------------------------------------------------------------------
# This script deletes the FileMaker record from the Calendars application
# ------------------------------------------------------------------------------------------------------------

# Check the plug-in first
Perform Script [ "Plug-in Checker" ]

# Check for iCal UID
If [ IsEmpty ( CALENDARS::uid ) ]
    Show Custom Dialog [ "Warning" ; "You cannot delete a calendar without an Apple Calendar UID" ]
    Exit Script [ ]
End If

# Delete Individual Calendar from iCal.  Requires iCal UID as the parameter.
Show Custom Dialog [ "Message" ; "Are you sure you want to permanently delete this calendar in Apple Calendar?"  & ¶ &"WARNING: Dele…" ]
If [ Get (LastMessageChoice) = 1 ]
    Exit Script [ ]
Else
    If [ PCiM_RemoveItem( "calendar" ; CALENDARS::uid ) ≠ 0 ]
        Show Custom Dialog [ "Warning" ; PCiM_GetLastError( "text" ) ]
    Else
        Set Field [ CALENDARS::uid ; "" ]
        Commit Records/Requests [ No dialog ]
        Show Custom Dialog [ "Message" ; "Calendar successfully deleted from Apple Calendar." ]
        Delete Record/Request [ No dialog ]
        Close Window [ Current Window ]
    End If
End If
```

Example 2 - Delete an Event

Deleting an event requires use a few more parameters. In the example below we call PCIM_RemoveItem(“event” ; EVENTS::uid ; EVENTS::occurrence ; $span) to delete the specified event. The type parameter is set to “event” and the uid parameter is set to EVENTS::uid referencing the uid field on the EVENTS table in the demo file. The occurrenceDate is set to EVENTS::occurrence referencing the occurrence field on the EVENTS table. The occurrenceDate is a required parameter if deleting an event and indicates the date of the specific occurrence of the event to delete. The span parameter is also required when deleting an event and indicates which events should be deleted. Valid span values are “this,” “future” or “all.” “This” will affect only the current item in the recurrence pattern. “Future” will affect all future items in the recurrence pattern. “All” will affect all items in the recurrence pattern. Below the span parameter is set to a variable referenced as $span.

```
# ------------------------------------------------------------------------------------------------------------
#  This script will delete the event record from the calendar
# ------------------------------------------------------------------------------------------------------------

# Check the plug-in first:
Perform Script [ “Plug-in Checker” ]

# Check for iCal UID
If [ IsEmpty ( EVENTS::uid ) ]
    Show Custom Dialog [ "Warning" ; "You cannot delete a record without an iCal UID. Would you like to delete the FileMaker Record inst…" ]
    If [ Get(LastMessageChoice) = 2 ]
        # Delete the FileMaker record
        Delete Record/Request [ No dialog ]
        Perform Script [ “Close Window Detail ( Type )” ; Parameter: "event" ]
        Perform Script [ “Import Events” ; Parameter: "event" ]
        Exit Script [ ]
    Else
        Exit Script [ ]
    End If
End If

# Delete Individual Contact from iCal.  Requires UID as the parameter.
Show Custom Dialog [ "Message" ; "Are you sure you want to permanently delete this event in iCal?" ]
If [ Get (LastMessageChoice) = 1 ]
    Exit Script [ ]
Else
    Set Variable [ $span ; Value: "this" ]
    If [ EVENTS::isRecurring = "True" ]
        Show Custom Dialog [ "Warning" ; "Delete this one event or all future events also?" ]
        If [ Get( LastMessageChoice ) = 3 ]
            Exit Script [ ]
        Else If [ Get( LastMessageChoice ) = 2 ]
            Set Variable [ $span ; Value: "all" ]
        End If
    End If
    If [ PCiM_RemoveItem( "event" ; EVENTS::uid ; EVENTS::occurrence  ; $span  ) ≠ 0 ]
        Show Custom Dialog [ "Warning" ; PCiM_GetLastError ]
    Else
        Show Custom Dialog [ "Message" ; "Record successfully deleted in iCal." ]
        Perform Script [ “Close Window Detail ( Type )” ; Parameter: "event" ]
        Perform Script [ “Import Events” ; Parameter: "event" ]
    End If
End If
```

Example 3 - Delete a Reminder

When deleting a reminder we call PCIM_RemoveItem(“reminder” ; REMINDERS::uid). The type parameter is set to “reminder” and the uid parameter is set to REMINDERS::uid referencing the uid field on the REMINDERS table in the demo file.

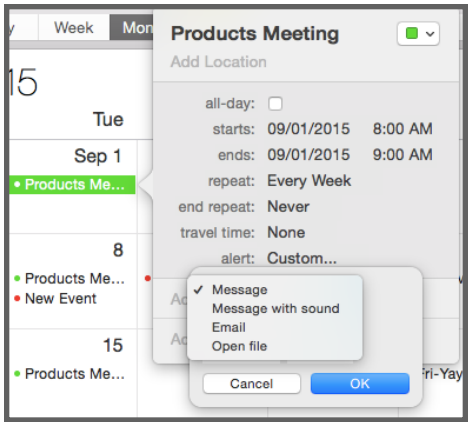
```
# ------------------------------------------------------------------------------------------------------------
#  This script deletes a task from the Reminders
# ------------------------------------------------------------------------------------------------------------

# Check the plug-in first:
Perform Script [ “Plug-in Checker” ]

# Check for iCal UID
If [ IsEmpty ( REMINDERS::uid ) ]
    Show Custom Dialog [ "Warning" ; "You cannot delete a record without an Reminders UID. Would you like to delete the FileMaker Record…" ]
    If [ Get(LastMessageChoice) = 2 ]
        # Delete the FileMaker record
        Delete Record/Request [ No dialog ]
        Perform Script [ “Close Window Detail ( Type )” ; Parameter: "reminder" ]
        Perform Script [ “Import All Reminders” ]
        Exit Script [ ]
    Else
        Exit Script [ ]
    End If
End If
# Delete Individual Task from iCal.
Show Custom Dialog [ "Message" ; "Are you sure you want to permanently delete this reminder from Reminders?" ]
If [ Get (LastMessageChoice) = 1 ]
    Exit Script [ ]
Else
    If [ PCiM_RemoveItem( "reminder" ; REMINDERS::uid ) ≠ 0 ]
        Show Custom Dialog [ "Warning" ; PCiM_GetLastError ]
    Else
        Show Custom Dialog [ "Message" ; "Reminder successfully deleted from Reminders." ]
        Delete Record/Request [ No dialog ]
        Perform Script [ “Close Window Detail ( Type )” ; Parameter: "reminder" ]
        Perform Script [ “Import All Reminders” ]
    End If
End If
```

8) Adding and Removing Alarms

You can add and remove alarms from any reminder or event using the PCIM_AddAlarm(type ; abstrigger ; action ; eaddr ; reltrigger ; sound ; url), PCIM_RemoveAlarm(type ; index) or PCIM_RemoveAllAlarmsFrom(type).

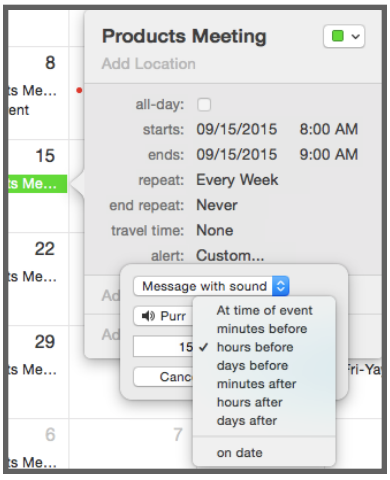
Adding Alarms

Alarms in Apple Calendar can be set to display a message, display a message with sound, send an email, open a file, or run a script as shown below.



You can set as many alarms as you want to a single event or reminder. However, if your calendar is administered through a Microsoft Exchange Server, then you can only set one alarm per event or reminder.

Once you set the alarm action, then you have the option to set a relative or absolute trigger as shown below. The relative trigger is relative to the number of seconds before or after the event or reminder date. The absolute trigger is a timestamp indicating the exact date and time to trigger the alarm.



Let's have a look at how to set the various Apple Calendar alarm actions in FileMaker.

Example 1 - Add "Message" Alarm to Event

```
#Open the Calendar Event Item at Index
PCIM_OpenItemWithUID( "event" ; SOMEUID )
#
if( Using an absolute date to trigger the alarm )
  #Populate Apple Calendar open event with Alarm and absolute trigger
  PCIM_AddAlarm( "event" ; SOMETIMESTAMP ; "display" ; "" ; "" ; "" ; "" )
else
  #Populate Apple Calendar open event with Alarm and relative trigger (15 minutes prior to event)
  PCIM_AddAlarm( "event" ; "" ; "display" ; "" ; "-900" ; "" ; "" )
end if
#
#Save Record in Apple Calendar
PCIM_SaveItem( "event" )
```

Example 2 - Add "Message with Sound" Alarm to Event

```
#Open the Apple Calendar Event Item at Index
PCIM_OpenItemWithUID( "event" ; SOMEUID )
#Populate Apple Calendar open event with Alarm and absolute trigger
PCIM_AddAlarm( "event" ; SOMETIMESTAMP ; "sound" ; "" ; "" ; SOMESOUNDNAME ; "" )
#Save Record in Apple Calendar
PCIM_SaveItem( "event" )
```

Example 3 - Add "Send an Email" Alarm to Event

```
#Open the Apple Calendar Event Item at Index
PCIM_OpenItemAt( "event" ; $index )
#Populate Apple Calendar open event with Alarm and absolute trigger
PCIM_AddAlarm( "event" ; SOMETIMESTAMP ; "email" ; SOMEADDRESS@SOMEDOMAIN.COM ; "" ; "" ; "" )
#Save Record in Apple Calendar
PCIM_SaveItem( "event" )
```

Removing Alarms

There are two functions used for removing alarms. PCIM_RemoveAlarm(type ; index) will remove the specified alarm from the currently opened item. To obtain a number of alarms for an item, open the item and then use the PCIM_GetItemCountForType function.

Example 1 - Remove Alarm from Event

```
#assumes that an event is currently opened
$count = PCIM_GetItemCountForType( “alarm” )
loop
 exit loop if $count < 1
 #Remove Alarm
 PCIM_RemoveAlarm( “event” ; $count)
 $count = $count -1
end loop
#Save Record in Apple Calendar
PCIM_SaveItem( “event” )
```

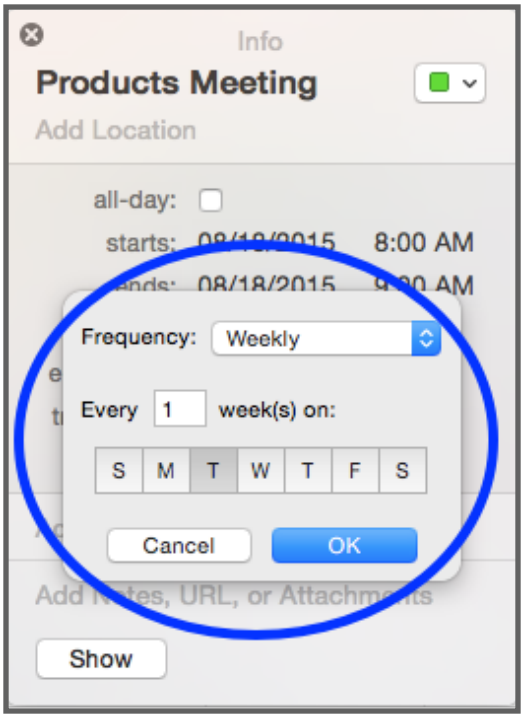
The second function to remove alarms is PCIM_RemoveAllAlarms. This function removes all alarms for the item type specified. This function requires a call to PCIM_openItemWithUID or PCIM_OpenItemAt as you must know what event or reminder you are working with before you can delete all the alarms.

Example 2 - Remove All Alarms from Reminder

```
#Open the Reminder Item at Index
PCIM_OpenItemWithUID( “reminder” ; SOMEUID )
#Remove Alarms
PCIM_RemoveAllAlarms( “reminder” )
#Save Record in Apple Calendar
PCIM_SaveItem( “reminder” )
```

8) Recurrence / Repeating Events

Every event in Apple Calendar has at least one occurrence. Some events may repeat and those that do have several occurrences. Each occurrence of an event is identified by the date on which the specific occurrence starts. From within Apple Calendar the user can set how often and at what interval an event repeats with the "repeat" popup menu located in the event editor as shown in the image below.



Creating the interval and number of occurrences with the plug-in is not as simple as using a popup menu, but with some explanation the FileMaker developer will be able to create and delete the "repeat" feature on the event.

In order to learn how to work with events that repeat the FileMaker developer needs to understand the objects and processes that Apple Calendar uses to create the Event occurrences. Basically, Apple Calendar uses a set of 10 properties to determine when and how often an event will repeat. These properties are contained by two objects: the Recurrence Rule and the Recurrence End. The Recurrence Rule object contains six properties and the Recurrence End object contains two properties.

Recurrence Rules

A Recurrence Rule object contains a set of properties that Apple Calendar uses in to determine on which day(s) to repeat an event. The Recurrence Rule object also contains a reference to a Recurrence End object which is used to determine the date of the last occurrence of the repeating event.

Types of Recurrence Rules and Their Properties:

There are four basic types of Recurrence Rules: daily, weekly, monthly, and yearly. The recurrenceType property indicates the type of Recurrence Rule. It also indicates the base unit between occurrences. For example, a recurrenceType value of "daily" indicates that the base unit between occurrences is a day and a type of monthly indicates a base unit of a month. The recurrenceInterval indicates how many base units between each occurrence. For example a recurrenceType of "weekly" and a recurrenceInterval of 3 means that the event will repeat every third week. There are other properties in a Recurrence Rule that are specific to the specific type of rule. These properties are used to further modify which dates occurrences will fall on.

Weekly types

For weekly types the days of the week may also be specified with the daysOfTheWeek property.

- daysOfTheWeek -

Monthly Types

There are two additional properties for monthly Recurrence Rules: daysOfTheMonth and nthWeekDaysOfTheMonth. These properties are mutually exclusive. This means that if one has a value, then the other does not. When creating Recurrence Rules the nthWeekDaysOfTheMonth parameter is evaluated before the daysOfTheMonth.

- daysOfTheMonth-
- nthWeekDaysOfTheMonth -

Yearly Types

There are two additional properties for yearly Recurrence Rules: monthsOfTheYear and nthWeekDaysOdTheMonth. If the nthWeekDaysOfTheMonth is specified then the monthsOfTheYear MUST also be specified.

- monthsOfTheYear -
- weekOfTheYear -

Property Descriptions:

- daysOfTheWeek -

This property is a number list (return separated list of numbers) that indicates which days of the week the occurrences should occur. Each number may be a value between 1 and 7 (1 for the first day of the week and 7 for the last). For example the value

1

3

5

would indicate that the occurrences should occur on the 1st, 3rd, and 5th days of the week. If Sunday is the first day of the week then the event occurs on Sunday, Tuesday, and Thursday.

- daysOfTheMonth-

This property is a number list (return separated list of numbers) that indicates which days of the month the occurrences should occur. Each number may be a value between 1 and 31. For example the value

1

10

20

30

would indicate that the occurrences should occur on the 1st, 10th, 20th, and 30th days of the month.

- weekOfTheYear -

This property is a number list (return separated list of numbers) that indicates which weeks of the year the occurrences should occur. Each number can be from 1 to 53 and from -1 to -53, where negative values count backwards from the end of the year. Example values are:

1

-1

would indicate the 1st week in January and the last week in December.

- monthsOfTheYear -

This property is a number list (return separated list of numbers) that indicates which months of the year the occurrences should occur. Each number may be a value between 1 and 12. For example the value

1

3

12

would indicate that the occurrences should occur in January, March, and December.

Recurrence End

A Recurrence End object contains one indicator (usesEndDate) and a set of two properties (endDate and occurrenceCount) that Apple Calendar uses to determine if and when an event will stop repeating. Not every Recurrence Rule has a Recurrence End. If the event is to repeat forever the Recurrence Rule for that event will not have a Recurrence End and the Recurrence Rule's hasEnd property will be false.

- usesEndDate -

Indicates whether the endDate property is used to end the Recurrence Rule, or if the occurrenceCount is used to end the Recurrence Rule. Returns true if there is an endDate for the recurrence end.

- endDate

Indicates after a specific date the Recurrence Rule should end. The end date if the recurrence uses an end date value.

- occurrenceCount -

Indicates after how many occurrences a Recurrence Rule should end. Number of times an event repeats, if the recurrence uses a number value. 0 if it uses an end date.

When creating a recurrence rule for an event if the occurrenceCount and endDate parameters are empty, then the Recurrence Rule will not have an end.

Please see demo file for examples of how to add, import and delete recurrences.

9) Error Handling

Typically a 0 is returned for a success and if 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 does not return a 0 OR returns !!ERROR!!, then immediately after call PCIM_GetLastError function for a detailed description of what the exact error was.

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

Example 1:

```
Set Variable [ $result = PCIM_SomePluginFunction( "a" ; "b" ) ]
If [ $result ≠ 0 ]
Show Custom Dialog [ "An error occurred: " & PCIM_GetLastError ]
End If
```

Example 2:

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

The PCIM_GetLastError(format) function gives you the option to display the error description or error number. If format parameter is empty such as PCIM_GetLastError or PCIM_GetLastError(""), then a description in English of the error will be returned. Since displaying an error number can be more user friendly in international environments, you also have the option to display an error number rather than an English text description. If the format parameter is set to "Number" such as PCIM_GetLastError("Number"), then an error number will be returned.

Please find a list of error codes and descriptions below.

Error Code	Description
0	SUCCESS
-1	Plug-in not registered
-2	Plug-in expired
-3	Invalid Number of Parameters
-4	Invalid value for parameter
-10	Registration failed
9000	Invalid call order. Must access other item prior to call.
9001	Item not found
9002	Item index out of bounds
9003	Unable to create event predicate
9004	Cannot create calendar array from names
9005	Missing argument in function.
9006	Invalid argument in function.
9007	Unable to create reminder predicate.
9008	No Alarms available.
9009	Unable to create item,
9010	Unable to create recurrence end.
9011	Unable to create recurrence rule.
9012	Unable to create item.
9013	Unable to locate calendar by uid.
9014	Unable to open calendar.
9015	This operating system is not supported.
9016	Parameter is not supported.
9017	Unable to save calendar. Either a parameter is invalid or the calendar is read-only.
9018	Unable to save event. Either a parameter is invalid or the event is read-only.
9019	Unable to save reminder.
9020	Invalid parameters passed for adding a recurrence rule.
9021	Invalid recurrence type. Must be ‘Daily’, ‘Weekly’, ‘Monthly’, or ‘Yearly’
9022	Unable to delete calendar. Calendar could be non-deletable or the only calendar existing.
9023	This calendar can’t be modified.
9024	An interval greater than 0 is required for adding recurrence to an event.
9025	Due date is invalid. Please verify there is a valid due date.
9026	Alarm was unable o be added.
9027	Alarm was unable to be deleted.
9028	An email must be specified when setting an alarm type to ‘email’.
9029	URLs and an alarm type of ‘procedure’ are not supported after 0S X 10.9
9030	A sound name must be specified when setting up an alarm type to ‘sound’.
9031	A relative offset or absolute date must be specified for the alarm.
9032	An action for the alarm must be specified.
9033	Invalid date format for the alarm.
9034	Event was not found
9035	Reminder was not found.

Error Code	Description
9036	Calendar was not found.
9037	A span must be specified.
9038	Unable to create calendar.
9039	Unable to create reminder list.
9040	Unable to create event.
9041	Unable to create reminder.
9042	Invalid type specified in parameter.
9043	An event must be opened before a recurrence rule can be added.
9044	An alarm action was not specified.
9045	An alarm type must be specified.
9046	A reminder must be opened before adding an alarm.
9047	An event must be opened before adding an alarm.
9048	An event must be opened before a recurrence rule can be removed.
9049	An event must be opened before setting a property.
9050	A property must be specified.
9051	A calendar must be opened before setting a property.
9052	A reminder must be opened before setting a property.
9053	An invalid span value was passed. Valid values are ‘this’, ‘future’, or ‘all’.
9054	You must fetch events before getting an object count of events.
9055	You must fetch reminders before getting an object count of reminders.
9056	An event must be opened before getting an object count.
9057	A reminder or event must be opened before getting an alarm count.
9058	Unable to find calendar by title.
9059	Unable to find reminder by uid.
9060	Unable to open reminder at the specified index.
9061	Unable to open alarm at the specified index.
9062	An attendee must be opened before getting a property.
9063	Invalid property specified in parameter.
9064	An alarm must be opened before getting a property.
9065	A calendar must be opened before getting a property.
9066	An event must be opened before getting a property.
9067	A reminder must be opened before getting a property.
9068	An event must be opened before getting a recurrence rule property.
9069	Invalid Recurrence Rule End property specified in parameter. Valid sub properties are 'usesEndDate' , 'occurenceCount' , or 'endDate'.
9070	Event could not be saved. Events containing attendees can’t be modified externally from the calendar application.
9071	Calendar could not be saved. Verify that it allows content modification.
9072	Unable to open reminder list.
9073	The Recurrence Rule Property ‘WeeksOfTheYear’ is currently not supported.
9074	No Relative Offset specified when adding the Relative Offset alarm property.
9075	No Absolute Offset specified when adding the Absolute Offset alarm property.
9076	Start Date is greater than End Date for this event.
9077	An Event Predicate has not been created. Get an Event Predicate before opening an event.
9078	Unable to delete reminder.
9079	Unable to delete event.

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!