Productive Computing

gManipulator

Developer's Guide

Revised July 27, 2020

Table of Contents

I. Introduction

The gManipulator plug-in connects your FileMaker solution with Google. With this plug-in, FileMaker Pro users are able to bidirectionally exchange data between FileMaker and Google. These operations are accomplished using FileMaker function calls from within FileMaker calculations. These calculations are generally determined from within FileMaker "SetField" or "If" script steps. This document describes the basic integration steps, features, and error handling. Please see the accompanying Functions Guide for a list of available plug-in functions and Google fields.

Product Version History:
https://www.productivecomputing.com/products/google-filemaker-integration

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 Function's Guide
3) Watch our video tutorials:
   http://help.productivecomputing.com/knowledge_base/categories/gmanipulator
4) Review our FileMaker demo file
5) Familiarize yourself with the Google Suite

II. Integration Steps

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

1) Installing the Plug-in

We have introduced installers to make installation of our plug-ins even easier. These installers will not only install the FileMaker plug-in file, but will also install any third-party software or dependencies needed for the plug-in to function, as well as install the demo file and any 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.

Once you download the gManipulator installation zip package, simply extract the package and open the resulting folder. Install the gManipulator with the following steps.

Windows Installation:

1. Run the "setup.exe" file that is included in the bundle downloaded from our website
2. If prompted, install the Visual C++ 2013 Runtime Libraries
3. If you are currently running FileMaker, please close FileMaker so that the plug-in will be installed correctly.
4. Accept the End User License Agreement ("EULA")
5. Select the location to install the plug-in*
6. Confirm the installation
7. If prompted by Windows user account control, allow the installer to run
8. Your installation is complete!

*In order for FileMaker to properly recognize the plug-in, we suggest you do not change this default location. The FileMaker plug-in needs to be installed to the base FileMaker/Extensions folder to be available across multiple versions of FileMaker. However, if you wish to install the plug-in at a version specific location like FileMaker Pro Advanced 16/Extensions, you may browse to the folder to do so.

Mac Installation:

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

Note: Both installers come with an application (.exe or .pkg) to install the plug-in and an Extras folder. In the Extras folder, you will find additional resources such as the License, README, 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 bundle from our website. Note for Windows Users: If you have not already, you will need to run the Visual C++ installers that are in the Extras folder for the plug-in to function properly.

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 an error comes up, please refer to the troubleshooting steps below involving the most common problems that may cause those errors.

1. Invalid Bitness of FileMaker
    a. In some cases, FileMaker Pro may be attempting to install a plug-in with a different bitness than the FileMaker Pro application. This is most common with Windows plug-ins. The general rule is that the plug-in and FileMaker Pro must be the same bitness.
    b. To resolve this, ensure that the container field holding the plug-in contains the correct bitness of the plug-in. You can verify the plug-in’s bitness by checking the file extension: if the extension is .fmx, the plug-in is a 32-bit plug-in; if the extension is .fmx64, the plug-in is a 64-bit plug-in. Mac plug-ins are built to function as both 32-bit and 64-bit plug-ins. You can verify the bitness of FileMaker Pro itself by viewing the “About FileMaker Pro” menu option in the Help menu, and clicking the “Info” button to see more information; bitness is found under “Architecture”.

2. Missing Dependencies
    a. Every plug-in has dependencies, which are system files present in the machine’s operating system that the plug-in requires in order to function. If a plug-in is “installed” into an Extensions folder, but the plug-in does not load or is not visible in the Preferences > Plug-ins panel in FileMaker Pro’s preferences, is it likely that there are files missing.
    b. To ensure that the appropriate dependencies are installed, please verify that the Visual Studio 2013 C++ redistributable package is installed. This can be located by opening the Control Panel and checking the Installed Programs list (usually found under “Add/Remove Programs”. Older plug-ins may require the Visual C++ 2008 Redistributable Package, instead of the 2013 version.
    c. Some plug-ins also have a .NET Framework component that is also required. The gManipulator plug-in requires .NET Framework 4.5 which can be downloaded at the following link: https://www.microsoft.com/en-us/download/details.aspx?id=30653

3. Duplicate Plug-in Files
    a. When installing plug-ins, it is possible to have the plug-in located in different folders that are considered “valid” when FileMaker 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 to fail to load a newly-installed plug-in during the installation process.
    b. To resolve this, navigate to the different folder locations listed below 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.

        Installation Paths (Windows):
        C:\Users\(current user)\AppData\Local\FileMaker\Extensions

C:\Users\(current user)\AppData\Local\FileMaker\FileMaker Pro [Advanced]\##\Extensions
C:\Program Files (x86)\FileMaker\FileMaker Pro ## [Advanced]\Extensions

Installation Paths (Mac):
~/Library/Application Support/FileMaker/Extensions
~/Library/Application Support/FileMaker/FileMaker Pro [Advanced]/##/Extensions
~/Applications/FileMaker Pro ## [Advanced]/Extensions

If the troubleshooting steps above do 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, 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 number 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.

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 acknowledgement from the server and installs a certificate on the machine. This certificate never expires. If the certificate is ever moved, modified, or deleted, then the client will be required to register again. On Windows, this certificate is in the form of a “.pci” file. On Mac, this certificate is a “.plist” preferences 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 PCGM_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 PCGM_GetOperatingMode and PCGM_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 PCGM_Register, PCGM_GetOperatingMode, and PCGM_Version functions. This eliminates the need to manually register each machine and ensures that the plug-in is installed and properly registered. Below are some basic steps to create a “Plug-in Checker” script.

```
If [ PCGM_Version( “Short” ) = “” or PCGM_Version( “Short” ) = ? ]
Show Custom Dialog [ “Warning”; Message: “Plug-in not installed.”; Buttons: “OK” ]
Exit Script [ Text Result:”Plug-in not installed” ]
Else If [ PCGM_GetOperatingMode <> “LIVE” ]
Set Field [ Main::gRegResult; Value:PCGM_Register( “licensing.productivecomputing.com” ; “80”
; “/PCIReg/pcireg.php” ; $$LicenseID )]
If [ Main::gRegResult <> 0 ]
Show Custom Dialog [ “Registration Error” ; Message: “Plug-in registration failed”; Buttons:”OK”
]
End If
End If
```

5) FileMaker 16 Plug-in Script Steps

Newly introduced with 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 gManipulator plug-in's script steps to demonstrate the difference. The same scripting differences can 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 gManipulator Demo File: EVENT__Pull.

Script 1 - EVENT__Pull with calculation ("traditional") plug-in scripting:

```
Set Error Capture [On]
Allow User Abort [Off]

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

# Make sure the user has specified a calendar folder
If [IsEmpty ( Main::gFolder Events )]
    Show Custom Dialog ["Warning"; "Please select a Calendar folder first."]
    Exit Script [Text Result:"No Calendar folder specified."]
End If

# Clear any records from Events to refresh the list
Freeze Window
Go to Layout ["SU__Events" (Events); Animation:None]
Show All Records
Delete All Records [With dialog:Off]

# Open the selected folder
Set Variable [$result; Value: PCGM_OpenFolder( Main::gFolder Events ; "Calendar" )]

# Get the record count to see if any records exist for the current folder
Set Variable [$count; Value: PCGM_GetRecordCount]
If [$count > 0]
    Set Variable [$result; Value: PCGM_GetFirstRecord]
    Loop
        Exit Loop If [$result = "!!ERROR!!" or $result = "END"]

        New Record/Request
        Set Field [Events::Google_ID; PCGM_GetFieldValue( "ID" )]

        # Pull the editable fields
        Set Field [Events::Attendees; PCGM_GetFieldValue( "Attendees" )]
```

```
            Set Field [Events::Description; PCGM_GetFieldValue( “Description” )]
            Set Field [Events::Location; PCGM_GetFieldValue( “Location” )]
            Set Field [Events::Start; PCGM_GetFieldValue( “Start” )]
            Set Field [Events::End; PCGM_GetFieldValue( “End” )]
            Set Field [Events::Summary; PCGM_GetFieldValue( “Summary” )]
            Set Field [Events::Visibility; PCGM_GetFieldValue( “Visibility” )]

            # Pull the read-only fields
            Set Field [Events::Created; PCGM_GetFieldValue( “Created” )]
            Set Field [Events::Creator; PCGM_GetFieldValue( “Creator” )]
            Set Field [Events::Kind; PCGM_GetFieldValue( “Kind” )]
            Set Field [Events::Updated; PCGM_GetFieldValue( “Updated” )]

            Commit Records/Requests [With dialog:Off]
            Set Variable [$result; Value: PCGM_GetNextRecord]
        End Loop
        Go to Layout [original layout; Animation:None]
        Go to Object [Object Name: “calendar”]
    Else
        Go to Layout [original layout; Animation:None]
        Go to Object [Object Name: “calendar”]
        Show Custom Dialog [“No records”; “No records to pull from Google.”]
    End If
```

Script 2 - EVENT__Pull with plug-in script steps:

```
    Set Error Capture [On]
    Allow User Abort [Off]

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

    # Make sure the user has specified a calendar folder
    If [IsEmpty ( Main::gFolder Events )]
        Show Custom Dialog [“Warning”; “Please select a Calendar folder first.”]
        Exit Script [Text Result:”No Calendar folder specified.”]
    End If

    # Clear any records from Events to refresh the list
    Freeze Window
    Go to Layout [“SU__Events” (Events); Animation:None]
    Show All Records
    Delete All Records [With dialog:Off]

    # Open the selected folder
    PCGM_OpenFolder [Select; Results:$result; Folder ID:Main::gFolder Events; Folder Type:”Calendar”]

    # Get the record count to see if any records exist for the current folder
```

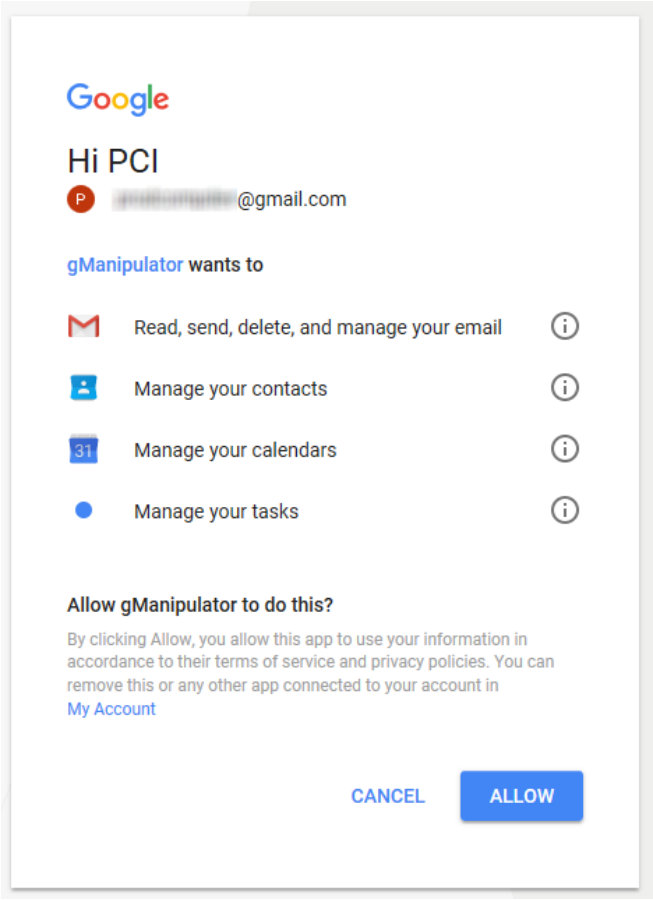
```
PCGM_GetRecordCount [Select; Results:$count]
If [$count > 0]
    PCGM_GetFirstRecord [Select; Results:$result]
    Loop
        Exit Loop If [$result = "!!ERROR!!" or $result = "END"]

        New Record/Request
        PCGM_GetFieldValue [Select; Results:Events::Google_ID; Field Name:"ID"]

        # Pull the editable fields
        PCGM_GetFieldValue [Select; Results:Events::Attendees; Field Name:"Attendees"]
        PCGM_GetFieldValue [Select; Results:Events::Description; Field Name:"Description"]
        PCGM_GetFieldValue [Select; Results:Events::Location; Field Name:"Location"]
        PCGM_GetFieldValue [Select; Results:Events::Start; Field Name:"Start"]
        PCGM_GetFieldValue [Select; Results:Events::End; Field Name:"End"]
        PCGM_GetFieldValue [Select; Results:Events::Visibility; Field Name:"Visibility"]

        # Pull the read-only fields
        PCGM_GetFieldValue [Select; Results:Events::Created; Field Name:"Created"]
        PCGM_GetFieldValue [Select; Results:Events::Creator; Field Name:"Creator"]
        PCGM_GetFieldValue [Select; Results:Events::Kind; Field Name:"Kind"]
        PCGM_GetFieldValue [Select; Results:Events::Updated; Field Name:"Updated"]

        Commit Records/Requests [With dialog:Off]
        PCGM_GetNextRecord [Select; Results:$result]
    End Loop
    Go to Layout [original layout; Animation:None]
    Go to Object [Object Name: "calendar"]
Else
    Go to Layout [original layout; Animation:None]
    Go to Object [Object Name: "calendar"]
    Show Custom Dialog ["No records"; "No records to pull from Google."]
End If
```

6) Talking to Google

Talking to Google typically requires that you follow these steps:

A. Authenticate to Google

In order to exchange any information with Google's API services, the plug-in must be authenticated as a user. Performing the plug-in function "PCGM_Authenticate" prompts the user to sign into their Google account and allow the plug-in to communicate with different areas of the Google API.



Once the plug-in has been granted permission to communicate with Google as the authenticated user, all incoming and outgoing messages from the plug-in will be treated as belonging to the authenticated user.

Important Developer's Note: The authentication process has an internal timer of 60 seconds after the call to PCGM_Authenticate is made. During this time, the user is prompted to sign into Google and authorize the plug-in to connect. If the user does not authorize or cancel the authentication process within the 60 second time limit, or closes the web browser prematurely, the PCGM_Authenticate function will return "!!ERROR!!", and PCGM_GetLastError will report an error that the authentication process timed out.

1. Session Management

Once the plug-in has been authenticated successfully with Google, the authenticated session is stored locally for the plug-in on the user's machine. This will allow the plug-in to reauthenticate at a later time without requiring the need for the user to allow access to the Google APIs again, and streamlines the authentication process. On Windows, this session is stored in the user's Application Data folder, alongside other required dependencies installed for the plug-in at runtime. On Mac, the authentication data is saved in the user's keychain utility for the plug-in to access.

Session information can be stored and loaded via the plug-in through the PCGM_LoadSession and PCGM_SaveSession functions. It is recommended that when the plug-in authenticates successfully, the FileMaker solution should save a copy of the session information to a field in the solution, such as a "Globals" or "Preferences" table where appropriate. This session info can then be loaded on another machine, allowing that second machine to communicate as the authenticated session without needing to authenticate via PCGM_Authenticate.

The session information is also required when deploying the plug-in in a server-side capacity. For more information on server-side development and deployment, please refer to the "Server-Side Deployment" section later in this document.

2. Disconnecting a Session

The plug-in can also disconnect the currently-authenticated session by calling PCGM_Disconnect. This function will terminate the active session, revoking access for the authenticated account, and additionally will remove any saved session information from the user's machine as noted above. Note that any machine using this disconnected session will need to reauthenticate in order to continue communicating with Google.

B. Select a Folder

After authentication, the next step is to navigate to the desired Google folder. Each folder in Google represents a collection of records of a specific module type, and can have different names: Mail folders are called "Labels"; Contact folders are called "Contact Groups"; Calendar folders are called "Calendars"; and Task folders are called "Task Lists". For ease of understanding, however, all folders can be referred to as "Folders".

Opening a folder requires both the folder's Google ID and the type of the folder to open (Mail, Contact, Calendar, or Task). If the Google ID of a folder is not immediately apparent, the developer can also navigate folders via the PCGM_GetFirstFolder and PCGM_GetNextFolder functions.

Example: opening the Inbox folder of a user's Mail
PCGM_OpenFolder("INBOX" ; "Mail")

Example: opening a Contact folder with an ID stored in a field
PCGM_OpenFolder(Contacts::FolderID ; "Contact")

Example: Opening the first Calendar folder indirectly

PCGM_GetFirstFolder(“Calendar”)

C. Create/Open a Record

Once the folder has been opened, you can now create new records to be placed within the folder, or access records that are contained by the folder.

Create New Record:

There is one method for creating a new record which is by calling the PCGM_NewRecord(optModule) function. Upon being called, a new record of the specified type will be created and stored in memory. The record will not be pushed to Google until the record is properly saved. Note that this function call will close any currently-opened record without committing any changes; it is up to the developer to properly save the current record before creating a new record.

Open an Existing Record:

When the record has already been created, the plug-in can open it in one of two methods: directly or indirectly.

If the user knows the Google ID of the desired record, they can open the record directly by calling PCGM_OpenRecord(RecordID ; Type), providing the Google ID and the type of the record to open. The record ID is the record’s unique identifier, which is the identifying code that references the record in Google, while the Type parameter is the type of record (Mail, Contact, Event, or Task). Note that this function call will close any currently-opened record without committing any changes; it is up to the developer to properly save the current record before opening a record.

If the record’s ID is not known, or the goal of a script is to pull a set of records from within the currently-opened folder, the script can iterate through the set of records with the use of the PCGM_GetFirstRecord and PCGM_GetNextRecord functions. PCGM_GetFirstRecord will open the first record in the currently-opened folder, and PCGM_GetNextRecord will move the internal record pointer to the next record, if it exists, loading that record into memory. Note that this function call will close any currently-opened record without committing any changes; it is up to the developer to properly save the current record before opening a record.

Example: Storing the Google IDs of all records within the Inbox

```
Set Variable [ $result ; PCGM_OpenFolder( “INBOX” )]
// Error capture…

Set Variable [ $id ; PCGM_GetFirstRecord]
Loop
    Exit Loop If [$id = “END” or $id = “!!ERROR!!” or $id = “?”]
    New Record/Request
    Set Field [ Mail::GoogleID ; $id ]
    Set Variable [ $id; PCGM_GetNextRecord]
End Loop
```

If the record's ID is unknown and the use case determines that all records of a given type should be read from Google into FileMaker, the function PCGM_GetAllRecords(Type) will serve that task. The function combines the essentials of the PCGM_OpenFolder and PCGM_GetFirstRecord functions, but instead of opening a specific folder, it will open the "base" folder and retrieve all records for the provided type, even records that may be hidden by special folder identifiers. The return value of PCGM_GetAllRecords will be the Google ID of the first record in the set, and subsequent calls to PCGM_GetNextRecord will iterate over the set as normal.

Additionally, the PCGM_GetAllRecords function will respect any filters that are previously declared in the workflow, so the developer can fine-tune what set of "all records" to pull back. Examples include all calendar event records within the past week, all mail records coming from the email address "j.smith@testemail.com", and so forth.

D. Manipulate the Fields

Once a record is created or opened, the fields will be accessible within that record. Using the PCGM_SetFieldValue function, fields within the opened record can be edited, changing their values. Conversely, PCGM_GetFieldValue will extract data from the fields within the opened record, returning that data to FileMaker. All available field names are in the Functions Guide.

The PCGM_SetFieldValue(FieldName ; FieldValue ; optType) function is used to populate fields in the new or currently opened record. Optionally a field type can be specified, which is used to differentiate between multi-value field types, such as home address or work email.

The PCGM_GetFieldValue(FieldName ; optType) function is used to extract data from the currently-opened record. Like SetFieldValue, a field type can be specified, which is used to differentiate between multi-value field types, such as home address or work email.

E. Saving the Record

After creating or modifying a record and populating it with data, use the PCGM_SaveRecord(optParam) function to save the record, submitting it to Google and returning the record's Google ID. The record will be saved as a child member of the currently-opened folder.

If the record is a Mail or Event type record, the "optParam" parameter can be specified to allow further control of how the record behaves when saving. By default, new or edited mail and event records will send out an email or notification to any added recipients. By specifying "DontSend" as the optional parameter, the email message will be saved as a draft, or the event record will simply be saved without sending notifications to linked attendees.

The following examples show saving records both with and without the optional parameter specified.

Example 1: Saving a new Mail record as a "Draft"

```
Set Variable [ $result ; PCGM_NewRecord( "Mail" )]
// Error capture…
```

```
Set Variable [ $result; Value: PCGM_SetFieldValue( “To” ; Mail::To )]
Set Variable [ $result; Value: PCGM_SetFieldValue( “Cc” ; Mail::Cc )]
Set Variable [ $result; Value: PCGM_SetFieldValue( “Bcc” ; Mail::Bcc )]
Set Variable [ $result; Value: PCGM_SetFieldValue( “Subject” ; Mail::Subject )]
Set Variable [ $result; Value: PCGM_SetFieldValue( “Body” ; Mail::Body )]

Set Variable [ $ID; Value: PCGM_SaveRecord( “DontSend” )]
If [ $ID = “!!ERROR!!” or $ID = “?” ]
        // Handle error
Else
        # Save the Google ID so the mail record can be edited later on.
        Set Field [ Mail::Google_ID; $ID ]
End If
```

Example 2: Saving an existing Contact record

```
Set Variable [ $result ; PCGM_OpenRecord( “Contact” )]
// Error capture…

Set Variable [ $result; Value: PCGM_SetFieldValue( “FirstName” ; Contacts::FirstName )]
Set Variable [ $result; Value: PCGM_SetFieldValue( “LastName” ; Contacts::LastName )]
Set Variable [ $result; Value: PCGM_SetFieldValue( “OrganizationTitle” ; Contacts::JobTitle ;
“Work” )]
Set Variable [ $result; Value: PCGM_SetFieldValue( “EmailAddress” ; Contacts::WorkEmail ;
“Work” )]
Set Variable [ $result; Value: PCGM_SetFieldValue( “PhoneNumber” ; Contacts::WorkPhone ;
“Work” )]

Set Variable [ $ID; Value: PCGM_SaveRecord]
If [ $ID = “!!ERROR!!” or $ID = “?” ]
        // Handle error
Else
        # Save the Google ID.
        Set Field [ Contacts::Google_ID; $ID ]
End If
```

F. Shortcut Functions

In addition to the traditional methods of creating/opening a record, setting field values, and then saving the record, the gManipulator plug-in adds in four one-off "shortcut" functions that allow for quick creation of records within a single function call. There is one such shortcut function for each record type:

PCGM_SendMail(To ; Cc ; Bcc ; Subject ; Body ; Attachments ; Format)

PCGM_CreateContact(FirstName ; LastName ; Company ; Phone ; Email ; Website ; AddressBlock)

PCGM_CreateEvent(Start ; End ; Summary ; Description ; Location ; bSetReminder)

PCGM_CreateTask(Due ; Title ; Notes)

Calling the PCGM_SendMail function will return a value of 0 for success or "!!ERROR!!" for failure. Calling PCGM_CreateContact, PCGM_CreateEvent, or PCGM_CreateTask will return a Google ID of the created record if successful, or "!!ERROR!!" if an error is encountered. Newly created records will be generated within the default folder for the given record type.

Please review the Functions Guide for specifics about the shortcut functions above.

G. Deleting Records

When a record is no longer needed, the PCGM_DeleteRecord(RecordID ; bPermanent) function can be called to remove that record from Google. This function also accepts a boolean (true/false) parameter indicating whether to delete the record permanently, that applies to Mail records. If specified as false, the record in question will not be permanently deleted, and instead flagged as "TRASH". All other record types will be permanently deleted regardless of the value of the bPermanent parameter.

The developer should call PCGM_ClearDeletedRecords to clear out any records marked as "TRASH". This function will locate any records flagged as trash and purge them from Google, permanently deleting them.

As a note, records that are permanently deleted cannot be restored; the system will need to recreate the record in order to "recover" the record in Google.

H. Filtering Records

By default, when pulling down records from Google, all records in the opened folder will be retrieved and available for insertion into FileMaker. However, there are times when pulling all records is not desired, and only a subset of records is needed. One method is to use Google's online interface to relocate desired records to subfolders, and to pull the records in from that subfolder so that you only get the contents of that folder. Another method is to use filtering by the two functions, PCGM_FilterByLastModified and PCGM_FilterByField.

Filtering by Modification Timestamp

The simplest form of filtering is to filter by the modification timestamp. After opening a folder, making a call to PCGM_FilterByLastModified and providing a standard FileMaker timestamp will ensure that when the record list is retrieved in the function call to PCGM_GetFirstRecord, only records that have been modified on or after that timestamp will be pulled in. There are some important caveats, however:

1) Contact records do not have a modification metadata field available to them, so you cannot filter by last modified for Contact records
2) Mail records have a creation metadata field, but not a modification metadata field, so the filter will filter by the creation date instead

Let's take a look at the use of PCGM_FilterByLastModified with a scenario and matching script.

Scenario 1: Pulling in all mail messages in the Inbox after the last batch process run

In this scenario, we assume that there is an automated process, running on FileMaker Server, that runs regularly in the background, pulling new information in form Google and adding or updating in FileMaker as appropriate. This example focuses on the Mail section of the database; similar logic could be made for Calendar and Task sections, as well. Let's see the script below for the part that pulls in the Mail records:

```
Allow User Abort [Off]
Set Error Capture [On]

# Verify the plug-in is ready to function
Perform Script ["Plug-in Checker"; Parameter:"NoDialog=True"]

# Verify the plug-in is authenticated
If [ IsEmpty( Main::bConnected ) or Main::bConnected = 0 ]
	Perform Script ["Load Authenticated Session"]
	If [ Get(ScriptResult) <> 0 ]
		Exit Script [ "Unable to load authentication. Please re-authenticate." ]
	End If
End If

# Get into the proper context
Set Variable [$mailFolderID; Value: Main::MailFolderID]
Set Variable [$lastRun; Value: Main::LastPullRun]

Go to Layout ["Script_Use__MAIL" (Mail); Animation:None]
Show All Records

# Open the Inbox Mail folder
Set Variable [$result; Value: PCGM_OpenFolder( $mailFolderID ; "Mail" )]
If [ $result <> 0 ]
```

```
	# Error capture: failed to open mail folder
	Exit Script [“Unable to open Mail folder: ” & PCGM_GetLastError( “Text” )]
End If

Set Variable [$result; Value: PCGM_FilterByLastModified( $lastRun )]
If [ $result <> 0 ]
	# Error capture: failed to apply filter
	Exit Script [“Unable to apply Last Modified filter: ” & PCGM_GetLastError(
“Text” )]
End If

# Get the first record, which applies the stored filter
Set Variable [$mailID; Value: PCGM_GetFirstRecord]
If [ $mailID = “!!ERROR!!” ]
	# Error capture: failed to pull records
	Exit Script [“Unable to pull records: ” & PCGM_GetLastError( “Text” )]
End If

# Filter has been applied; pull our records in and handle accordingly.
Loop
	Exit Loop If [ $mailID = “!!ERROR!!” or $mailID = “END” or $mailID = “?” ]
	Perform Script [“Pull Mail Message”; Parameter:$mailID]

	Set Variable [$mailID; Value: PCGM_GetNextRecord]
End Loop

# Update the batch timestamp
Go to Layout [“Script_Use__MAIN” (Main); Animation:None]
Set Field [Main::LastPullRun; Get(CurrentTimestamp)]

Exit Script [0]
```

The script above is a simple serverside driver script that applies the previous run’s timestamp to the found set of records in the mail folder, getting only the records that were added to the system after the timestamp, and pulls them into FileMaker via the subscript “Pull Mail Message”. Since this is server-side scripting, we make sure to include no calls to Show Custom Dialog. It is important to ensure that the call to apply the filter is made between the call to open a folder and the call to get the first record; this will validate and store the filter, and the filter will then be applied to the list of records being retrieved from Google, resulting in a potentially-smaller found set.

Filtering by Fields

For more complex record access requirements, the use of filter fields may be just what a developer needs in order to pull records based not just on modification/creation timestamps, but also on field content and folder participation. The PCGM_FilterByField(FieldName ; Value ; optAppendToFilter) function does just this, by letting the developer specify the field to filter on, the value to filter for, and an optional parameter to append the corresponding filter value to whatever filter is already in place. To see a simple instance of field filtering in action, let’s take a look at the following scenario and its script.

Scenario 2: Pulling in Calendar records due next month

In this scenario, we assume that the current month is July, 2018, and we want to get all upcoming calendar records that are in the month of August.

```
Allow User Abort [Off]
Set Error Capture [On]

# Verify the plug-in is ready to function
Perform Script [“Plug-in Checker”]

# Verify the plug-in is authenticated
If [ IsEmpty( Main::bConnected ) or Main::bConnected = 0 ]
	Perform Script [“Connect to Google”]
	If [ Get(ScriptResult) <> 0 ]
		Show Custom Dialog [“Authentication Error”; "Unable to authenticate.
Please try again.”]
		Exit Script [ -1 ]
	End If
End If

# Get into the proper context
Set Variable [$calFolderID; Value: Main::CalendarFolderID]

Go to Layout [“Script_Use__CAL” (Calendar); Animation:None]
Show All Records

# Open the Inbox Mail folder
Set Variable [$result; Value: PCGM_OpenFolder( $calFolderID ; “Calendar” )]
If [ $result <> 0 ]
	# Error capture: failed to open mail folder
	Show Custom Dialog [“Folder Error”; "Unable to open the folder: “ &
PCGM_GetLastError( “Text” )]
	Exit Script [ -1 ]
End If

Set Variable [$currentTS; Value: Get(CurrentTimestamp)]
Set Variable [$dateAugustStart; Value: Date( Month( $currentTS ) + 1 ; 1 ; Year(
$currentTS ) )]
Set Variable [$dateAugustEnd; Value: Date( Month( $currentTS ) + 1 ; 31 ; Year(
$currentTS ) )]

# Set the filter to get events whose time minimum is the start of August
Set Variable [$result; Value: PCGM_FilterByField( “timeMin” ; $dateAugustStart ; False
)]
# Set the filter to get events whose time maximum is the end of August, and append
# it to the first filter
Set Variable [$result; Value: PCGM_FilterByField( “timeMax” ; $dateAugustEnd ; True )]

# Get the first record, which applies the stored filters
Set Variable [$calID; Value: PCGM_GetFirstRecord]
If [ $calID = “!!ERROR!!” ]
	# Error capture: failed to pull records
	Show Custom Dialog [“Record  Error”; "Unable to get the first record: “ &
PCGM_GetLastError( “Text” )]
	Exit Script [ -1 ]
End If

# Filters have been applied; pull our records in and handle accordingly.
Loop
	Exit Loop If [ $calID = “!!ERROR!!” or $calID = “END” or $calID = “?” ]
	Perform Script [“Pull Calendar Event”; Parameter:$calID]
```

```
      Set Variable [$calID; Value: PCGM_GetNextRecord]
End Loop

# Return to the original context
Go to Layout [original layout; Animation:None]
```

Unsurprisingly, the simplistic structure of the script is very similar to that of Scenario 1: first the script checks to ensure it's registered and authenticated, then it navigates to the proper context, applies the filters in question, and finally processes the resulting records. The key difference here is the use of multiple calls to PCGM_FilterByField; the first function call applies the filter for the minimum time (dates or timestamps are acceptable), and the second functioncall appends the filter for the maximum time. If represented by a FileMaker find, this would be the equivalent of setting an Event record's "Start" field to be "8/1/2018…8/31/2018".

Advanced Filtering

Most advanced filtering is available for the Mail module; Calendar, Task, and Contacts support simple filtering. The use of multiple calls to PCGM_FilterByField allows for the developer to set up complex filter rules, allowing the solution to restrict the set of records in novel and creative ways. There are some key points to consider when working with multiple filters:

1) Multiple function calls to both PCGM_FilterByLastModified and PCGM_FilterByField are incompatible; calling PCGM_FilterByLastModified will clear any filters set by PCGM_FilterByField, and vice versa
    a. It still can be done, however; the way to apply a "last modified" filter while also providing Field filters is explained below.
2) Multiple calls to PCGM_FilterByLastModified will not set multiple last modified filters; only the last call will be the filter value used.
3) Multiple fields specified by PCGM_FilterByField are passed to Google with logical "AND" operators. That is to say, when retrieving the set of records, records will be pulled when Condition1 AND Condition2 are both true. This is the case for Mail, Calendar, and Task filters.
    a. The Contacts module, on the other hand, performs as a logical "OR" operator; records will be pulled when Condition1 OR Condition2 are true.

When filtering for mail records, all filter fields can be found in both the Functions Guide and the following website: https://support.google.com/mail/answer/7190?hl=en. This site refers to the filter fields that are used for searching for messages in Google's interface; the plug-in handles filters in the same way.

Example 1: Searching for mail records that are unread and from a specific email address

By default, when multiple filter functions are called, the filter criteria are combined with an AND operator and passed to Google to get results. As a reminder, make sure the second call is set with a third parameter of "True".

```
PCGM_FilterByField( "Sender" ; "test.person@email.com" ; False )
PCGM_FilterByField( "Is" ; "unread" ; True )
```

The combination of the two filters will return all email records that are both unread and from the sender with the email address test.person@gmail.com.

Example 2: Searching for mail records with a specific label that have an attachment, using a single function call

By making use of the "QueryString" or "Q" filter field, developers can set up a custom query using the filter fields in the linked website above all within the set of a single function call.

```
PCGM_FilterByField( "QueryString" ; "label:Friends is:unread has:attachment"
)
```

The above function call will filter with a query string so that only records with the label "Friends", that are unread messages, and have at least one attachment will be returned.

Example 3: Getting mail records in the "Work" label that are either modified after July 4th, 2018, or from a contact named "Dave"

In the use of the QueryString filter field, developers can also apply logical "OR" operations instead of the default logical "AND". This example also explores the use of providing a "last modified"-type filter field.

```
PCGM_FilterByField( "QueryString" ; "label:Work (from:Dave OR
after:7/4/2018)" )
```

This filter dictates that a record must be assigned to the "Work" label and be either from "Dave", or be created after July 4th, 2018.

I. Server-Side Deployment

The gManipulator plug-in is compatible with FileMaker Server, allowing server-side scripts, scheduled scripts, FileMaker WebDirect, and FileMaker Go to utilize the plug-in via the FileMaker Server scripting and web publishing engines. This section will go into detail on how to install the server-side plug-in, as well as how the plug-in manages registration, session control, and functionality.

1. Installation

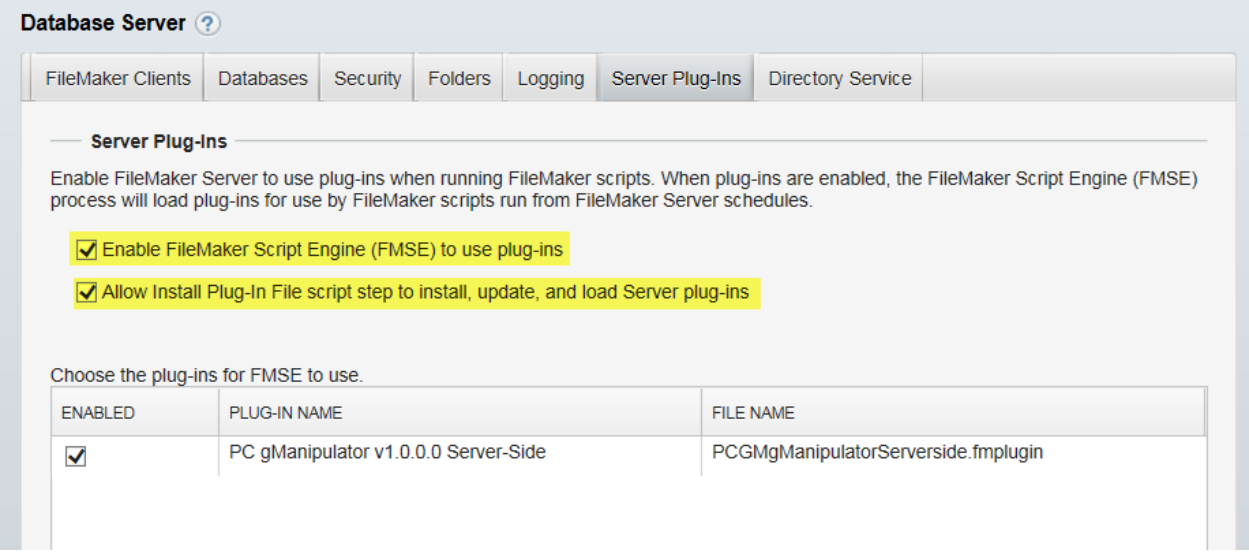
The server plug-in will need to be installed on the server machine within the FileMaker Server's "Extensions" folder. If the desire is to use the plug-in for Perform Script on Server or in server-side scheduled scripts, the plug-in must be installed in the scripting engine's Extensions folder, located within the "Database Server" folder in the installation directory. For WebDirect, this Extensions folder would be in the "cwpc" folder (Custom Web Publishing Core) in the Web Publishing Engine. See the paths below for the plug-in installations on a Windows server (assuming a default installation path):

Windows:
Database Server:
C:\Program Files\FileMaker\FileMaker Server\Database Server\Extensions
Web Direct and Custom Web Publishing:
C:\Program Files\FileMaker\FileMaker Server\Web Publishing\publishingengine\cwpc\Plugins

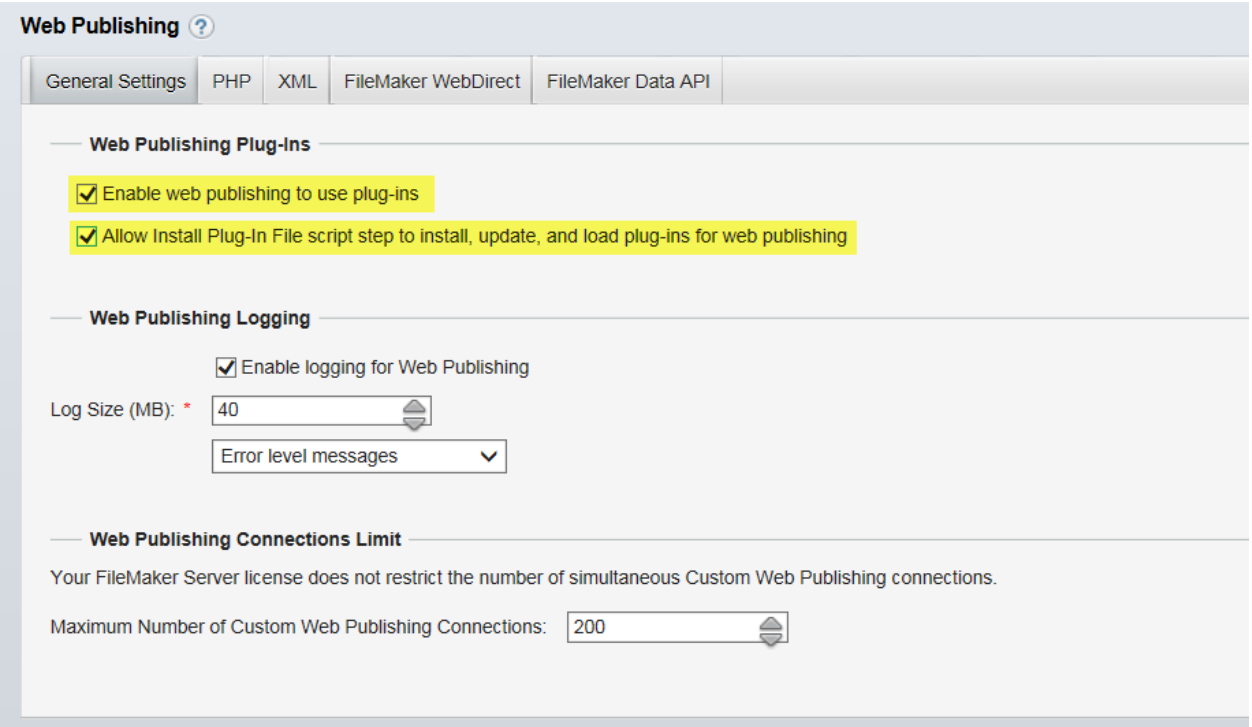
The demo file available from our website provides a one-click installation method for installing the gManipulator server-side plug-in into a host FileMaker Server's Extensions folder; however, it will only install into the Database Server location above. The demo file must be hosted on FileMaker Server in order to install the plug-in into the Server's Extensions folder. In order to install the plug-in for use in WebDirect or custom web publishing, it can be installed manually, or the demo file can be altered to allow access via WebDirect, where the "Install Plug-in" script step can install into the WebDirect Extensions folder.

Please note:
Any manual installation of the plug-in will require the server or web publishing engine to be restarted in order for the plug-in to load.

In order to allow FileMaker Server to install the server-side plug-ins via the demo file, both options in the Database Server > Server Plug-ins section in the Admin Console must be checked. See screenshot below.



For Web Publishing, the same options need to be checked in the Web Publishing > General Settings section of the Admin Console. See screenshot below.



Please refer to the link below for more information on managing server-side plug-ins on FileMaker Server.

https://fmhelp.filemaker.com/help/16/fms/en/index.html#page/fms%2Fplugins_manage.html%23wwconnect_header

2. Deployment Requirements

In order to utilize the gManipulator server-side plug-in, the plug-in must be installed at the location above on the FileMaker Server machine, and the solution that has scripts designed to use the plug-in must be hosted on that FileMaker Server machine. This is easily done by following the instructions for uploading a solution to FileMaker Server, which can be reviewed in the FileMaker Server User's Guide provided with each version of FileMaker Server.

3. Registration

Like the client version, the server-side version of gManipulator must be registered with the Productive Computing, Inc., licensing server in order to function. A call to PCGM_Register with valid license information must be made at the beginning of any scripted process, whether running from a scheduled script or from a call to Perform Script on Server. This process should be hard-coded and reference either passed script parameters from a calling script or pre-populated fields within the FileMaker solution. Please see the "Recommended Script Structure" section below for an example of how registration can be utilized.

Depending on the intended deployment of the gManipulator plug-in, scripts utilizing the plug-in may need to be changed.

4. Recommended Script Structure

When a client calls Perform Script on Server, the action reaches out to FileMaker Server with the desired script to allow the server's Scripting Engine to perform the heavy lifting of calculation and data manipulation, and return a result back to the caller. For handling plug-in usage, the following is a handy checklist of what is needed to ensure proper plug-in functionality:

1. Ensure the script space starts in the right table occurrence context with the appropriate record(s) in the found set and available to be accessed
2. Perform a call to PCGM_Register with valid registration information to ensure the plug-in is in the "LIVE" operating mode
3. Perform a call to PCGM_LoadSession with an active saved session string to ensure the plug-in is communicating with the correct Google account and is ready to transfer information

See below for an example of a usage of Perform Script on Server, using scripts from the PCGM_Demo_Serverside demo file:

Script 1: Push Mail [Server] ("Driver Script")

```
Set Error Capture [On]
Allow User Abort [Off]

# Check the solution status. This file should be hosted.
Perform Script ["Check Solution Status"]

# Get the ID of the mail record to send
Set Variable [$mailID; Value: Mail::ID]
```

```
    # For demo purposes, we don't want to modify an email that has already been sent
    If [not IsEmpty ( Mail::Google_ID )]
        Show Custom Dialog ["Message"; "This email has already been sent. Do you wish to resend it?"]
        If [Get(LastMessageChoice) <> 1]
            Halt Script
        End If
    End If

    # Perform the script on the server
    Perform Script on Server [Wait for completion:On; "PSOS - MAIL__Push"; Parameter: $mailID]

    # Get the result and check for errors.
    Set Variable [$result; Value: Get(ScriptResult)]
    If [$result = 0]
        Show Custom Dialog ["Success"; "Email sent successfully."]
    Else
        Show Custom Dialog ["Failure"; "Failed to send email message: " & $result]
    End If
```

Script 2: PSOS - MAIL__Push ("Worker Script")

```
    Set Error Capture [On]
    Allow User Abort [Off]

    # Verify the plug-in is registered
    Perform Script ["PSOS - Register Plug-in"]
    Set Variable [$result; Value: Get(ScriptResult)]
    If [$result <> 0]
        # Log the error result into an internal logging table
        Perform Script ["Record Log ( LogText )"; Parameter: "Error encountered registering the plug-
    in." & ¶]
        Exit Script [Text Result:$result]
    End If

    # Verify the plug-in is authorized
    Perform Script ["PSOS - Authenticate"]
    Set Variable [$result; Value: Get(ScriptResult)]
    If [$result <> 0]
        # Log the error result into an internal logging table
        Perform Script ["Record Log ( LogText )"; Parameter: "Error encountered with plug-in
    authentication." & ¶]
        Exit Script [Text Result:$result]
    End If

    Set Variable [$mailID; Value: Get(ScriptParameter)]
    If [$mailID = ""]
        # Log the error result into an internal logging table
```

```
    Perform Script [“Record Log ( LogText )”; Parameter: “Invalid mail record chosen. Please verify the mail record has a valid FileMaker ID.” & ¶]
    Exit Script [Text Result:”Invalid mail record chosen. Please verify the mail record has a valid FileMaker ID.”]
End If

# Make sure there is data to push
Commit Records/Requests [With dialog:Off]
Go to Layout [“SU__Mail” (Mail); Animation:None]
# Perform Find: Mail::ID == $mailID
Perform Find [Restore]
If [Get(FoundCount) = 0]
    # Log the error result into an internal logging table
    Perform Script [“Record Log ( LogText )”; Parameter: “No records to send out.” & ¶]
    Exit Script [Text Result:”No records to send out.”]
End If

# Push the record
Set Variable [$result; Value: PCGM_OpenFolder( Main::gFolder Mail ; “Mail” )]
Perform Script [“Record Log ( LogText )”; Parameter: “Adding record...” & ¶]
Set Variable [$result; Value: PCGM_NewRecord( “Mail” )]

# Push the editable fields
Perform Script [“Record Log ( LogText )”; Parameter: “Setting fields...” & ¶]
Set Variable [$result; Value: PCGM_SetFieldValue( “From” ; Mail::From )]
Set Variable [$result; Value: PCGM_SetFieldValue( “To” ; Mail::To)]
Set Variable [$result; Value: PCGM_SetFieldValue( “Cc” ; Mail::Cc )]
Set Variable [$result; Value: PCGM_SetFieldValue( “Bcc” ; Mail::Bcc )]
Set Variable [$result; Value: PCGM_SetFieldValue( “Subject” ; Mail::Subject )]
Set Variable [$result; Value: PCGM_SetFieldValue( “Body” ; Mail::Body )]
Set Variable [$result; Value: PCGM_SetFieldValue( “isBodyHTML” ; If( Mail::IsBodyHTML = 1 ; “true” ; “false” ) )]
Set Variable [$result; Value: PCGM_SetFieldValue( “Priority” ; Mail::Priority )]

# Handle attachments
Perform Script [“Record Log ( LogText )”; Parameter: “Adding attachments...” & ¶]
Commit Records/Requests [With dialog:Off]
Go to Related Record [Show only related records; From table: “Mail_Attachments”; Using layout: “SU__Mail_Attachments” (Mail_Attachments)]
If [not Get(LastError)]
    Perform Script [“Record Log ( LogText )”; Parameter: “Attachment found. Inserting...” & ¶]
    Commit Records/Requests [With dialog:Off]
    Go to Record/Request/Page [First]
    # Loop through all attachments
    Loop
        If [not IsEmpty ( Mail_Attachments::File )]
            # Insert attachment as binary container data
```

```
            Set Variable [$result; Value: PCGM_AddAttachment( Mail_Attachments::File ;
        Mail_Attachments::FileName ; “true”)]
            Perform Script [“Record Log ( LogText )”; Parameter: “Result: ” & $result & ¶]
            Commit Records/Requests [With dialog:Off]
        End If
        Go to Record/Request/Page [Next; Exit after last:On]
    End Loop
    Go to Layout [original layout; Animation:None]
End If

# Save the record
Perform Script [“Record Log ( LogText )”; Parameter: “Saving the record...” & $result & ¶]
Set Variable [$result; Value: PCGM_SaveRecord]

If [$result = “!!ERROR!!”]
    Set Variable [$message; Value: “Failed to save the record: “ & PCGM_GetLastError( “Text” )]
Else
    Set Field [Mail::Google_ID; $result]

    # Pull some of the meta data
    Set Variable [$result; Value: PCGM_OpenRecord( Mail::Google_ID )]

    Set Field [Mail::From; PCGM_GetFieldValue( “From” )]
    Set Field [Mail::Date; PCGM_GetFieldValue( “Date” )]
    Set Field [Mail::internalDate; PCGM_GetFieldValue( “internalDate” )]
End If

# Developer’s Note: When returning from a PSOS session, adding a small pause before exiting script
tends to improve performance.
Pause/Resume Script [Duration (seconds): .1]
Exit Script [Text Result:$message]

Script 3: PSOS - Register Plug-in (Subscript)
    Set Error Capture [On]
    Allow User Abort [Off]

    Go to layout [“Main” (Main); Animation:None]

    # The registration process requires access to the internet. Once registered
    # on our server, the plug-in will either be running in “DEMO” mode or fully
    # registered and unlocked. You are no longer required to register our
    # products via the configuration screen. This script step is required to be
    # called at least once before running any function in the plug-in.

    Set Field [Main::gRegistration Result; Value: PCGM_Register( Main::gRegistration Server ;
    Main::gRegistration Port ; Main::gRegistration Page ; Main::gRegistration License ID]

    If [Main::gRegistration Result = 0]
```

```
        Exit Script [Text Result:0]
    Else
        Exit Script [Text Result:Main::gRegistration Result]
    End If
```

Script 4: PSOS - Authenticate (Subscript)

```
    Set Error Capture [On]
    Allow User Abort [Off]

    Go to layout [“Main” (Main); Animation:None]

    # Check the session info parameter.
    If [Main::SessionInfo = “”]
        Exit Script [Text Result:”No session information is saved. Please connect to Google using the
    Client plug-in and save the session.”]
    End If

    # Attempt to authenticate
    Set Field [Main::gAuthentication Result; Value: PCGM_LoadSession( Main::SessionInfo )]

    If [Main::gAuthentication Result = 0]
        Exit Script [Text Result:0]
    Else
        Exit Script [Text Result:PCGM_GetLastError( “Text” )]
    End If
```

Script Explanations:

When performing scripts on the server, it is helpful to split the script workflows into two groups: the Client-Side scripts, which are primarily for gathering found sets, performing any necessary work that requires user interaction, calling any server-side scripts, and handling results; and the Server-Side scripts, which perform the work and require no user interaction. For ease of naming, we refer to these as “Driver” and “Worker” scripts, respectively.

The Driver script in the example above, “Push Mail [Server]”, is intended to gather any information from the user’s perspective that will be passed to the worker scripts in order to accomplish the workflow’s goal. It also provides the user with interface cues, such as messages or layout changes, as necessary. Finally, the Driver script is responsible for issuing the call to Perform Script on Server, referencing the desired Worker script that will carry out the workflow’s task using the FileMaker Server scripting engine.

The Worker script, by comparison, has no user interface cues (no message dialogs, all other dialogs are “off” or disabled) and its purpose is to perform any and all work related to the workflow. In this case, the job of the Worker script is to compose and send the email out. An important fact to note about the Worker script is that when the script starts up, the virtual FileMaker session created by FileMaker Server will run through the solution’s “Open” script (if defined), and will start in the default context of the solution as if the user had opened it up in FileMaker Pro for the first time. Therefore, it is important to ensure that the Worker script first sets itself into the correct context using Go to Layout, as well as

performing any finds using information passed to it from the Driver script, to ensure the process will be able to work without issue.

In addition to the primary workflow executed by the Worker script, the subscripts for registration and authentication make sure that the plug-in is in a proper operating status on the FileMaker Server machine. The calls to register and authenticate should be made at least once per call to Perform Script on Server; the recommended location for this would be at the beginning of the Worker script called directly by the Driver.

5. Server-Side Scheduled Scripts

Scheduled scripts would follow similar adjustments to the order of scripting as scripts called by Perform Script on Server. The “Driver” script, in this case, could be largely ignored, but the script space still needs to be in the correct context, PCGM_Register must still be called, and PCGM_LoadSession must still use valid Google session information before any work can commence.

6. WebDirect Scripting

At a glance, scripting in WebDirect and scripting in FileMaker Pro are identical when it comes to plug-in calls and how they’re structured. The only critical difference between WebDirect scripting and standard FileMaker Pro scripting is the requirement of authenticating with a Google account and registering the plug-in. All other features are the same.

When working within a WebDirect session, it’s safe to assume that the session will be similar to that of a FileMaker Pro session. This means that the call to PCGM_LoadSession must take place at least once each time the user is connected to the solution via WebDirect. A developer may wish to include a call to PCGM_LoadSession with each “workflow” similar to server-side scheduled scripts or in a Perform Script on Server session. Similarly, a WebDirect implementation can also make use of Perform Script on Server instead of using script steps and function calls as a client application; in this case, the plug-in used by the Database Engine is the one performing the work and not the one used by the Web Publishing Engine. It is up to the developer to determine the best setup for their solution when it comes to where to host the server-side version of the plug-in.

7) Custom Contact Fields

When working with user contacts, there is a possibility that not all fields available to the contact will fit with a developer's specific solution. Google's People API allows for a subset of "user-defined" properties to be set for a given contact record. These user-defined properties are specified by a particular key, which is the "name" of the field to be added, edited, or retrieved for the specified contact.

For an example of handling contact fields, the two scripts below show the setting and getting of a custom field called "FileMaker ID", which houses the primary key field of the FileMaker contact record.

Script 1: Setting the Custom Field for a New Contact

```
Set Variable [ $result ; PCGM_OpenRecord( "Contact" )]
// Error capture…

Set Variable [ $result; Value: PCGM_SetFieldValue( "FirstName" ; Contacts::FirstName )]
Set Variable [ $result; Value: PCGM_SetFieldValue( "LastName" ; Contacts::LastName )]
Set Variable [ $result; Value: PCGM_SetFieldValue( "OrganizationTitle" ; Contacts::JobTitle ;
"Work" )]
Set Variable [ $result; Value: PCGM_SetFieldValue( "EmailAddress" ; Contacts::WorkEmail ;
"Work" )]
Set Variable [ $result; Value: PCGM_SetFieldValue( "PhoneNumber" ; Contacts::WorkPhone ;
"Work" )]

# Set the Custom Field value for the "FileMaker ID"
Set Variable [ $result; Value: PCGM_SetFieldValue( "UserDefined" ; Contacts::ID ; "FileMaker ID"
)]

Set Variable [ $ID; Value: PCGM_SaveRecord]
If [ $ID = "!!ERROR!!" or $ID = "?" ]
    // Handle error
Else
    # Save the Google ID.
    Set Field [ Contacts::Google_ID; $ID ]
End If
```

Script 2: Getting the Custom Field from an Existing Contact

```
Set Variable [ $result ; PCGM_OpenRecord( Contacts::Google_ID ; "Contact" )]
// Error capture…

# Set the Custom Field value for the "FileMaker ID"
Set Variable [ $customField; Value: PCGM_GetFieldValue( "UserDefined" ; "FileMaker ID" )]
```

In the examples, if we assume that the FileMaker ID for the contact in question is "CNTCT0001", the screenshot below shows the appearance of the contact record when viewing it on Google Contacts in the web browser:

Dr. Johnny C Doel Sr.

"Johnny "

Renamed Title

My Contacts Test

Home	John@gmail.com
Work	John@company.com
	Add email
Home	111-111-1111
Work	222-222-2222
	Add phone
Home	123 Home Apt 1 City Home, AB
Work	123 Work Apt 2 City Work, AB
	Add address
Birthday	December 28, 1990
Anniversary	May 15, 2012
Anniversary	May 15, 2012
	Add date
Work	www.work.com
Other	www.home.com
	Add URL
Profession	Developer
FileMaker ID	CNTCT0001
	Add custom field

Add

8) Error Handling

When something unexpected happens while calling a plug-in function, the function will return a result of “!!ERROR!!”. This makes it simple to check for errors. If the value of “!!ERROR!!” is returned, then immediately call PCGM_GetLastError for a detailed description of the exact error.

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 solution. Here are a few samples of how you can check for errors.

```
Set Variable [ $result ; PCGM_SomeFunction( “Param1” ; “Param2” ) ]
If [ $result = “!!ERROR!!” ]
        Show Custom Dialog [ “Error” ; PCGM_GetLastError( “Text” ) ]
End If
```

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

The list below includes return codes that the plug-in directly reports on. For some error codes, such as -16003, -998, and -400, the error text can vary due to different causes. Please refer to the result of PCGM_GetLastError(“Text”) for more information when encountering those particular error codes.

Code	Description
0	Success
-1	Plug-in Not Registered
-2	Registration Failed
-3	Invalid Number of Parameters
-4	Invalid Parameter Value(s)
-10	Expired Registration / Registration Error
-200	Library Failed to Load
-300	The specified file cannot be found
-301	The specified file cannot be found
-302	The specified folder cannot be accessed

-400	System Error
-998	Unknown Exception
-999	COM Exception
-16000	Google Log-on Failed
-16001	Authentication information missing; please authenticate to Google first
-16002	This Version is not supported
-16003	Error encountered from Google
-16004	Authenticated user has insufficient Google permissions
-16005	Google service has not yet been authenticated
-16006	This function is not supported in the current version of gManipulator
-16007	Operation called without opening/creating a record
-16008	Missing parameter value
-16009	Error encountered during Authentication
-16011	Invalid record or folder type
-16012	Invalid path or file name
-16013	Property is read-only
-16014	Invalid field name specified
-16015	Invalid field value provided
-16016	Field is unsupported

III. Known Issues

As the plug-in is used in integration, issues may arise that have been determined to have no immediate resolution, or have a workaround available. This section is used to document these known issues; if a full resolution is later determined, the issue entry in this section will be updated to reflect the solution.

1550 – Plug-in Installation Error Specific Case

When performing an installation of the plug-in, there are certain situations in which a “1550” error can come up. The most common reason is that the plug-in is missing its dependencies, which can be resolved by following the steps specified in the Plug-in Installation section above.

A more uncommon reason that FileMaker returns a “1550” error, however, is due to a system setting integrated into the plug-in. This setting determines which system libraries the plug-in utilizes in order to communicate with the required dependencies it uses to perform its various tasks. This scenario comes up when performing the Install Plug-in script step to install the plug-in into FileMaker after the plug-in is already installed, in two subsequent calls; the first call is successful, but the second call results in a -1550 error. After receiving the error, the plug-in will be disconnected from FileMaker, and FileMaker will behave as if the plug-in is not installed.

There are two resolutions for this specific scenario:

1. Implement scripting to prevent the plug-in from being installed more than once using the Install Plug-in script step per FileMaker session, as a single call should be sufficient to update the existing plug-in if a new version is desired.
2. Close and re-open FileMaker, which causes the plug-in to properly initialize and be considered “installed”.

Error 403 - “Quota Exceeded”

The Google API sets a limitation on the number of requests that can be submitted to its services on a per-day and per-minute basis. Quotas are split between the different services; there are separate quotas for Google Contacts, Gmail, Calendar, and Tasks. This limitation is fairly generous, however these quotas are linked to all communication that is performed by the gManipulator plug-in, which affects any and all customers utilizing the plug-in.

If this error occurs, there can be multiple reasons; one reason is that there have been too many requests made to a given service within the span of a set number of seconds, and the other reason is that too many requests have been made to a given service within the span of a day. The table below outlines the default quota for each of the four services that gManipulator communicates with.

Service Name	Request Quota #1	Request Quota #2	Request Quota #3
People API	30 read req’s / min / user	60 write req’s / min / user	90 crit. write / min / user
Gmail API	1,000,000,000 queries / day	25,000 queries / 100sec / user	2,000,000 queries / 100 sec
Calendar API	1,000,000 queries / day	500 queries / 100sec / user	N/A
Tasks API	50,000 / day	500 / 100 sec / user	N/A

For example, for the Calendar API, the gManipulator plug-in is able to submit up to one million requests (push or pull) per day, with no more than five hundred requests submitted per 100 seconds per authenticated user account.

If you encounter a Quota Exceeded error, determine it to be a daily quota instead of a per-100-seconds quota, and require more requests to be submitted, please reach out to our support line via the Contact Us section down below and inform us which service you experienced the error message for (e.g. if you were sending an email, or pulling calendar events, etc.). We will request for Google to expand the quota limit and will inform you via email as soon as the quota has been increased.

Minimum Modification Time Lies too Far in the Past

When making a request to Google using the "UpdatedMin" field for filtering on Events, the function "PCGM_GetFirstRecord" may return this error if the value of "UpdatedMin" is too far back. The tested "window" of time that the UpdatedMin field will be successful for is up to 30 days prior to the current date of making the request. This is a known issue, and a resolution is currently under development. If using UpdatedMin or Last Modification filtering for Calendar Events in the interim, please ensure that the date or timestamp value passed to the filter function is within this 30-day window.

IV. Contact Us

Successful integration of a FileMaker plug-in requires the creation of integration scripts within your FileMaker solution. A working knowledge of FileMaker Pro, especially in the areas of scripting and calculation, 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!