Productive Computing

Cloud Manipulator

Developer's Guide

Revised November 18, 2020

Table of Contents

I. Introduction

The Cloud Manipulator plug-in connects your FileMaker solution with Amazon Web Services (AWS) S3 storage. This plug-in provides FileMaker users with the ability to perform high-level management of AWS S3 buckets, as well as upload and download files with the AWS S3 service, using a convenient, easy-to-understand set of functions. 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 AWS fields.

Product Version History

http://www.productivecomputing.com/aws-filemaker-plugin

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. Review our FileMaker Demo file

Technical Note

All functions return their results as a text string. It is up to the end user to convert this returned value when necessary. If an error occurs within a function call, the function will return the text string “!!ERROR!!”. This my help with capturing errors within your scripts.

II. Integration Steps

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

1. Installing the Plug-in with the Installer

We are happy to announce the introduction of installers for our latest plug-in releases. These installers will not only install the FileMaker plug-in, but will also install the third-party software needed for the plug-in to function, the demo file, and additional resources you may need. We recommend using the installers to ensure that all components necessary for the plug-in to function are properly installed.

Windows Installer:

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 and initialized correctly.
4. Accept the license
5. Select the location to install the plug-in*
6. Confirm the installation
7. If prompted by Windows security, 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 in a valid Extensions folder recognized by the application. By default, the plug-in will be installed to the base FileMaker/Extensions directory and will be accessible 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 location to do so.

Mac Installer:

1. Open the “Install Cloud Manipulator.dmg” file that you downloaded from the website
2. Run the “Install Cloud Manipulator” application 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”
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 License, README, FileMaker Demo file, and plug-ins, along with any optional supporting files that may be included.

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, if you have not already, you will need to run the Visual C++ 2013 Redistributable installers that you can download from the Microsoft website.

1. Open the FileMaker demo file available in the plug-in bundle
2. Select the "Install" button

3. Installing the Plug-in Manually

For those developers who prefer to get their hands dirty, you can also install the plug-in manually instead of using the plug-in installers or the plug-in demo file. As before, any required dependencies of the plug-in will need to be installed separately in order for the plug-in to function.

1. If FileMaker is open, close FileMaker completely
2. Locate the plug-in in your download bundle which will be located in a folder called "Plug-ins". On Windows, the plug-in will have a ".fmx" (for 32-bit FileMaker) or ".fmx64" (for 64-bit FileMaker) extension. For Mac, the plug-in will have a ".fmplugin" extension.
3. Copy the plug-in and paste it into the Extensions folder which is inside the FileMaker Program folder.
    1. On Windows, this is normally located at:
        1. 32Bit: "C:\Program Files (x86)\FileMaker\FileMaker # [Advanced]\Extensions"
        2. 64Bit: "C:\Program Files\FileMaker\FileMaker # [Advanced]\Extensions"
    b. On Mac, this is normally located at:
        1. "(Volume)/Applications/FileMaker #/Extensions" ("Volume" is the name of the mounted volume or drive)

Start FileMaker Pro. Confirm that the plug-in has been successfully installed by navigating to "Preferences" in FileMaker, then select the "Plug-ins" tab. There you should see the plug-in listed with a corresponding check box. This indicates that you have successfully installed the plug-in.

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. 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. 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, it's 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 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. All such plug-ins of ours will require the .NET Framework 3.5, which can be downloaded from the following link: https://www.microsoft.com/en-us/download/details.aspx?id=21

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 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 three troubleshooting steps above do not resolve the issue, please feel free to reach out to our support team for further assistance.

5. Uninstalling the Plug-in

If the user no longer needs the plug-in, or the plug-in is being moved to a different machine or environment, the best result would be to perform an uninstallation of the plug-in to ensure that FileMaker will not load it any further. Uninstallation takes place in one of two ways, depending on the operating system. The steps below assume that the plug-in was originally installed via the installation package.

Windows Uninstallation:

1. Verify that FileMaker is closed; the plug-in will not uninstall properly if it is still in use by FileMaker
2. Open the Control Panel application (usually found in the Start Menu)
3. Navigate to the “Programs or Features” option
    a. This may also be called “Add or Remove Programs”
4. In the list of programs installed on your machine, locate the plug-in to be uninstalled. It may be easier to click on the “Publisher” column header to sort by Publisher; our plug-ins have the publisher listed as “Productive Computing, Inc.”
5. Select the plug-in to uninstall
6. Choose the “Uninstall” button that appears at the top of the window
7. Walk through the setup process to fully uninstall the plug-in

Mac Uninstallation:

1. Verify that FileMaker is closed; the plug-in will not uninstall properly if it is still in use by FileMaker.
2. Open the installation package .DMG file containing the plug-in’s primary installer
    a. If the installation package is no longer available, you can download a fresh copy from our website.
8. Open the “Extras” folder
9. Locate and run the “uninstall.tool” script file
    a. This will bring up a Terminal window, displaying the progress of removing the plug-in and its supporting files that are present on your machine. Any files not installed by the plug-in installer .pkg file will not be affected.

Once uninstallation is complete, the plug-in will be removed from FileMaker’s Extensions folder, along with any files that were placed on the user’s Desktop, if still present. Uninstallation will ONLY affect the files placed by the plug-in’s installer package.

Manual Uninstallation:

In some cases, the installer packages were not used to install the plug-in, and instead the plug-in file was manually inserted into the FileMaker Extensions folder. In these cases, the plug-in will also need to be manually removed.

1. Open up your file system window (“File Explorer” on Windows, “Finder” on Macintosh)
2. Navigate to the Extensions folder that the plug-in was placed into. This is usually the common back-end folder at either of these file paths:
    1. Windows: C:\Users\(user)\AppData\Local\FileMaker\Extensions
    2. Mac: ~/Library/Application Support/FileMaker/Extensions
3. Verify that FileMaker is closed completely
4. Select and delete the plug-in file (with the extensions of .fmx, .fmx64, and/or .fmplugin)

Manually uninstalling the plug-in will only ensure that FileMaker will no longer try to load or use the plug-in. This will leave behind any files that were put in place by the plug-in. The cleanest way to install or uninstall is to use the provided installation packages and uninstallation methods as detailed above, as that will ensure a clean removal of the plug-in on uninstallation.

6. 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. 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 license copy, then simply enter your license number in the “LicenseID” field and select the “Register” button. Ensure you have removed the demo license ID and enter your registration information exactly as it appears in your confirmation email. The plug-in should now be running in “LIVE” mode.

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 seat available and identifies the machine. If there is a seat 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, renamed, 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 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 Register function call. 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 PCCM_GetOperatingMode and PCCM_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 PCCM_Register, PCCM_GetOperatingMode and PCCM_Version functions. This eliminates the need to manually register each machine and ensures the plug-in is installed and properly registered. Below are the basic steps to create a “Plug-in Checker” script.

```
If [ PCCM_Version( “short” ) = “” or PCCM_Version( “short” ) = “?” ]
	Show Custom Dialog [ Title: “Warning”; Message:”Plug-in not installed.”; 
Buttons:”OK” ]
	Exit Script [Result:”Not Installed”]
End If

If [ PCCM_GetOperatingMode <> “LIVE” ]
	 Set Field [Main::gRegResult; Value:PCCM_Register( 
“licensing.productivecomputing.com” ; “80” ; “/PCIReg/pcireg.php” ; “Your License ID” 
]

	If [ Main::gRegResult <> 0 ]
		Show Custom Dialog [ Title: ”Registration Failed”; Message: 
Main::gRegResult; Buttons: “OK” ]
	End If
End If
```

III. Talking to Amazon Simple Storage Service (S3)

Currently, the Cloud Manipulator plug-in is configured to transfer file data and information with Amazon's Simple Storage Service (S3). S3 allows users to upload to and download from an Amazon server in the cloud, storing and retrieving file data as needed from any internet-enable computer.

In order for the Cloud Manipulator plug-in to communicate with S3, FileMaker must first successfully complete a call to the function "PCCM_UseCredentials". When calling PCCM_UseCredentials, the plug-in will use the provided username and password to authenticate with S3, acquiring authentication information that is then used to sign each request.

1. Key Terms

When referring to different aspects of Amazon S3 that the plug-in communicates with, the naming conventions of the plug-in has been generalized in the interest of compatibility with other services. This section will quickly detail several terms that may be used, and how they translate specifically to Amazon S3.

Plug-in Term	S3 Term
Folder	Bucket
File	Object
ID	Key

2. Authentication

As with most third-party integrations, the first step that must take place is to connect to the service and authenticate. This will allow for all communication with the service to possess the authorization key, giving the plug-in "rights" to upload, download, and query information from the server. To authenticate to Amazon S3, the user will need to have their API Key and API Secret, available from their Amazon S3 account security section on the AWS website.

Once the user has their API key and secret (or it has been stored in the solution for use), the information should be passed to the plug-in via the PCCM_UseCredentials function call. Assuming all relevant information is stored in global fields, the call may look like so:

```
Set Variable [ $result; Value: PCCM_UseCredentials( "S3" ; Credentials::gAPIKey ;
Credentials::gAPISecret ; "region=" & Credentials::gRegion )]
```

The function call above tells the plug-in to connect to the S3 service, providing the API key and secret and specifying the connection region to be the chosen region. The authentication value lasts for the remainder of the FileMaker session; after FileMaker closes down, the session will be lost until PCCM_UseCredentials is called again.

It's important to note that the "optParams" parameter requires the key=value format. This is due to the potential for the Cloud Manipulator plug-in to support multiple online file storage locations.

3. Working with Buckets

The primary folders for S3 are known as "buckets". These buckets contain files, or "objects", as well as sub-folders, which are also considered "objects". When setting up buckets for use in the Amazon AWS S3 dashboard, it is important to note that bucket names are unique across the entire space of S3, not just within the scope of any specific user's account. Keep this fact in mind as you seek to roll out S3 storage capability with your client solutions.

The following functions are used when accessing bucket information or contents:

```
PCCM_ListAllFolders

PCCM_FetchFolderContents( FolderID ; Prefix ; Delimiter ; MaxKeys )

PCCM_GetPropertyForFolder( PropertyName ; ContentIndex )

PCCM_CreateFolder( FolderID )

PCCM_DeleteFolder( FolderID )
```

(As stated in the Key Terms section, all reference to "folders" refers to the S3 "buckets".)

After authenticating to S3, the call to PCCM_ListAllFolders will generate a return-separated list of accessible buckets that belong to the authenticated account. Each value in the list corresponds to the unique identifier of the bucket, which is used for passing to the PCCM_FetchFolderContents and PCCM_DeleteFolder functions.

Fetching the folder contents with PCCM_FetchFolderContents will load the bucket properties and contained objects into the plug-in's memory, opening the data up for retrieval. It's important to note that the PCCM_FetchFolderContents function will change its output based on the values provided to the parameters as follows:

- The Prefix parameter will limit the resulting objects to only those that have the starting prefix characters
- The Delimiter parameter will, if set to "/", limit the resulting set of records in memory to just folder-type objects instead of object keys
- The MaxKeys parameter will set the result set to that number of keys to return from S3; the default value is 1000

With the details and files of each bucket retrieved from S3, the plug-in can now use the function PCCM_GetPropertyForFolder, providing the property name and, optionally, the index of the content item to retrieve the property of. Two property values, “Keys” and “Count”, are available as folder properties, and will return information belonging to the folder fetched with PCCM_FetchFolderContents. Other property values, such as “Key”, “OwnerID”, “LastModified”, and so forth, belong to the contents of the fetched folder, and additionally require a 0-based index to indicate for which content object the function should retrieve data.

Folders can be created on Amazon AWS S3 through the PCCM_CreateFolder function. This folder accepts a name for the bucket to be created, and returns either 0 for success or an “!!ERROR!!” result. In order to create a bucket on S3, the authenticated user must have sufficient write permissions, and the bucket must have a globally-unique name; that is, the name of the bucket must be unique across the entire scope of Amazon AWS S3, even among buckets that are owned by other users. The newly-created bucket will be located within the same region that the plug-in has been authenticated to; for example, if the plug-in authenticates to the US Oregon region (us-west-2), then a new bucket will be part of the US Oregon region.

Folders can also be deleted from Amazon AWS S3 by using the PCCM_DeleteFolder function. The act of deleting a folder requires that the authenticated user has sufficient permissions to delete objects and folders. Deleting folders is permanent, and any folder that is deleted via PCCM_DeleteFolder cannot be recovered.

When working with S3 buckets containing large volumes of data, the plug-in will need to use a technique known as “pagination” to process the full set of objects found within a bucket. S3 offers two properties to assist with that: “IsTruncated”, which is a boolean flag of whether there are more objects left on S3 that would work for a given PCCM_GetFolder request; and “ContinuationToken”, which is a unique string that references the next page of objects on S3, and is only set when IsTruncated is true. For an example of pagination, refer to the Sample Scripts section below.

4. Working with Objects

The contents of folders are also known as “objects”. An object can be any kind of file: an image, a document, an archive folder, etc. Objects can also be folders in their own right, although they are not “buckets”, but rather compartments within a bucket for branching where files are stored. A bucket can contain a folder, which also can contain a folder, and so forth.

The following functions can be used to access and manipulate file data on S3:

```
PCCM_GetPropertyForFolder( PropertyName ; ContentIndex )

PCCM_GetObject( FolderID ; ObjectID ; FilePath )

PCCM_PostObject( FolderID ; ObjectID ; FilePath ; optDeleteAfterPost )

PCCM_CopyObject( SrcFolderID ; SrcObjectID ; DestFolderID ; DestObjectID )

PCCM_DeleteObject( FolderID ; ObjectID )

PCCM_GetPresignedURL( FolderID ; ObjectID ; optExpireMinutes )
```

With the exception of PCCM_GetPropertyForFolder, the object functions are folder-agnostic; they work with any folder that is specified by the FolderID parameter, with the object specified by the ObjectID parameter.

Files can be exchanged with the AWS S3 service through the use of the PCCM_GetObject and PCCM_PostObject functions. Both functions accept the ID of the bucket to store the file under, as well as the object ID referencing the file; when posting, the Object ID is the *desired* ID of the file (usually the file’s name), while the Object ID is the *actual* ID of the file when getting or downloading. This object ID value can also include subfolder names, as well. Below are some examples of the specifying the object ID:

TestFile.txt

Subfolder/TestFile2.txt

Subfolder1/Subfolder2/TestFile3.txt

The first example is simply a file named “TestFile.txt”, which will be its primary key. The second references a file named “TestFile2.txt”, stored under the folder called “Subfolder”. And the third references a file named “TestFile3.txt” that is housed within the folder structure “Subfolder1/Subfolder2”. When pulling objects back from S3 after the uploading of these files, the plug-in will list each object individually, and the parent folders will be listed as their own separate entries.

Getting or downloading an object from S3 requires the use of a file path parameter, which is a fully-qualified and properly formatted file path according to the current operating system architecture. The object will be retrieved from S3 from the bucket specified and, if successful, the data of the file will be saved to the provided file path and named the same as the object ID. A successful result of “0” indicates that the file is downloaded and saved at the path specified and can now be imported into FileMaker via the Insert File script step if desired.

Posting or uploading an object to S3 also requires the use of a file path parameter, with the same restrictions as with the GetObject function. A successful result of “0” indicates that the file has been uploaded to S3 and is now located at the provided object ID within the bucket specified. Optionally, the plug-in can be told to delete the object after posting to S3; this will remove the locally-stored file referenced by the file path and is useful for a one-and-done style of uploading an object, because it ensures that the file system is cleared of the original file only after a successful upload. It should be noted that when the system posts an object to a bucket in S3, the destination bucket must be of the same region that the plug-in has authenticated to. If the regions are different, simply re-authenticate to the desired region, and try the object upload again.

Objects can also be copied from one location to another using the PCCM_CopyObject function. Providing the source, the destination bucket, and the object identifiers will allow the plug-in to copy the object located at the source bucket to the destination bucket and provide it with the destination object name. This is useful when the desire is to "backup" a file, such as moving it to a subfolder so that the source object can be cleared out and reuploaded with a fresh copy from the local file system.

As with folders, objects can also be deleted. This action is permanent, and files deleted via the PCCM_DeleteObject function cannot be recovered.

If the developer or user requires the ability to share links of an uploaded file to other users (such as sending out an email containing a link to a FileMaker backup file, for example), the developer can make use of the PCCM_GetPresignedURL function to generate a presigned URL for the desired uploaded object. This URL allows any user who receives it to be able to download the object with the signer's permissions, and lasts for at most 7 days, or one week. After this time has expired, the link will no longer function as per Amazon AWS S3 guidelines.

5. Sample Scripts

The following is a collection of sample scripts that demonstrate some of the uses of the plug-in. These scripts are not complete FileMaker scripts; some sections of the script are “assumed” and will be commented as such. For additional sample scripts, please review the scripting of the Demo file, which is fully unlocked.

Scenario 1: Pull all buckets

This scenario pulls in all buckets for the authenticated user, storing their associated object key list and count of objects as individual records in FileMaker.

```
# Validation checking to ensure plug-in is installed, registered, authentication
# parameters are populated…

# Perform authentication
Set Variable [ $result; Value: PCCM_UseCredentials( “S3” ; Main::gAPIKey ;
Main::gAPISecret ; Main::gRegion ) ]
If [ $result <> 0 ]
       # Error handling…
End If

# List all folders
Set Variable [ $folderList ; Value: PCCM_ListAllFolders ]
If [ $folderList = “!!ERROR!!” ]
       # Error handling…
Else
       Go to Layout [ “Buckets”; Table Occurrence:Buckets ]

       # Initialize the loop iterator and max count variables
       Set Variable [ $iter; Value: 1 ]
       Set Variable [ $max; Value: ValueCount( $folderList ) ]
       Loop
              Exit Loop If [ $iter > $max ]

# Acquire the folder ID
Set Variable [ $folderID; Value: GetValue( $folderList ; $iter ) ]

# Fetch the folder contents; the additional parameters can be omitted,
# which will retrieve data as per the default.
Set Variable [ $result; Value: PCCM_FetchFolderContents( $folderID ) ]
If [ $result <> 0 ]
       # Error handling…
Else
       New Record/Request
       Set Field [ Buckets::Name; Value: $folderID ]
       Set Field [ Buckets::Keys; Value: PCCM_GetPropertyForFolderContent( “Keys” ) ]
       Set Field [ Buckets::Count; Value: PCCM_GetPropertyForFolderContent( “Count” )
]
End If

# Increment the bucket iterator to get the next bucket
Set Variable [ $iter; Value: $iter + 1 ]
       End Loop
```

```
        Go to Layout [ <original layout > ]
End If
```

The script above can be further expanded to also process each key in the Buckets::Keys field (which stores the return-separated list of keys for the bucket) for each bucket. That expansion of scope has been left out for the sake of brevity.

Scenario 2: Upload a file to S3 in a subfolder

This sample script demonstrates the act of uploading a file to S3 into a subfolder of the bucket called "Archive". The file is assumed to be on the local file system, which is a Windows operating system. A duplicate of this script will show the file path on a Mac operating system.

Windows OS:

```
# Validation checking to ensure plug-in is installed, registered, authentication
# parameters are populated…

# Perform authentication
Set Variable [ $result; Value: PCCM_UseCredentials( "S3" ; Main::gAPIKey ;
Main::gAPISecret ; Main::gRegion ) ]
If [ $result <> 0 ]
        # Error handling…
End If

# Prep the file path
Go to Related Record [ Matching Record; Show Only Related ]
Set Variable [ $filePath; Value:"filewin:" & Get(TemporaryPath) &
GetContainerAttribute( Objects::File ; "filename" ) ]
Set Variable [$winPath; Value:Substitute( $filePath ; [ "filewin:/" ; "" ] ; [ "/" ;
"\\" ] )]

# Set the new key name to be the subfolder and the key of the file
Set Variable [$newKey; "Archive/" & Objects::Key]

# Export the file to the file system
Export Field Contents [Objects::File; "$filePath"; Create directories:Off]

If [Get(LastError) <> 0]
        # Error handling…
Else
        Set Variable [ $result ; Value: PCCM_PostObject( Objects::BucketID ; $newKey ;
        $winPath ; True )]

        If [$result = "!!ERROR!!"]
                # Error handling…
        Else
                # Upload was successful
        End If
End If
```

Mac OS:

```
# Validation checking to ensure plug-in is installed, registered, authentication
# parameters are populated…

# Perform authentication
Set Variable [ $result; Value: PCCM_UseCredentials( “S3” ; Main::gAPIKey ;
Main::gAPISecret ; Main::gRegion ) ]
If [ $result <> 0 ]
       # Error handling…
End If

# Prep the file
Go to Related Record [ Matching Record; Show Only Related ]
Set Variable [ $filePath; Value:“filemac:” & Get(TemporaryPath) &
GetContainerAttribute( Objects::File ; “filename” ) ]
Set Variable [$macPath; Value:Substitute( $filePath ; “filemac:/” ; “” )]

# Set the new key name to be the subfolder and the key of the file
Set Variable [$newKey; “Archive/” & Objects::Key]

# Export the file to the file system
Export Field Contents [Objects::File; “$filePath”; Create directories:Off]

If [Get(LastError) <> 0]
       # Error handling…
Else
       Set Variable [ $result ; Value: PCCM_PostObject( Objects::BucketID ;
       Objects::Key ; $macPath ; True )]

       If [$result = “!!ERROR!!”]
              # Error handling…
       Else
              # Upload was successful
       End If
End If
```

Scenario 3: Archive a copy of a file on S3 and then clear it

In this scenario, we will be archiving a file from S3 using the PCCM_CopyObject function, then clearing it on success.

```
# Validation checking to ensure plug-in is installed, registered, authentication
# parameters are populated…

# Perform authentication
Set Variable [ $result; Value: PCCM_UseCredentials( “S3” ; Main::gAPIKey ;
Main::gAPISecret ; Main::gRegion ) ]
If [ $result <> 0 ]
       # Error handling…
End If

# Set the new key name to be the Archive subfolder and the key of the file
Set Variable [$newKey; “Archive/” & Objects::Key]

If [Get(LastError) <> 0]
       # Error handling…
Else
       # Copy the object to the Archive folder
```

```
        Set Variable [ $result ; Value: PCCM_CopyObject( Objects::BucketID ;
        Objects::Key ; Objects::BucketID ; $newKey )]

        If [$result = "!!ERROR!!"]
                # Error handling…
        Else
                # Copy was successful, so now delete the old object
                Set Variable [ $result ; Value: PCCM_DeleteObject( Objects::BucketID ;
                Objects::Key )]

                If [$result = "!!ERROR!!"]
                        # Error handling…
                Else
                        # Delete successful; file is now archived.
                End If
        End If
End If
```

Scenario 4: Get every object in a bucket belonging in a subfolder

In this scenario, we will be pulling every object belonging within a subfolder of a bucket, and we expect that the bucket contains more than 1000 objects. We assume that the script is in proper context.

```
# Validation checking to ensure plug-in is installed, registered, authentication
# parameters are populated…

# Perform authentication
Set Variable [ $result; Value: PCCM_UseCredentials( "S3" ; Main::gAPIKey ;
Main::gAPISecret ; Main::gRegion ) ]
If [ $result <> 0 ]
   # Error handling…
End If

# Set the prefix to be "2018". When used in PCCM_GetFolder, this will tell S3 to
# return the objects belonging to the subfolder "2018".
Set Variable [$prefix; "2018"]

# The first request is just sending along the bucket ID and the prefix
Set Variable [ $result; Value: PCCM_FetchFolderContents( Objects::BucketID ; $prefix
)]

If [$result = "!!ERROR!!"]
    # Error handling…
Else
    # Process the bucket properties

    Set Variable [ $keyList; Value: PCCM_GetPropertyForFolderContent( "Keys" ; "0" )]
    Set Variable [ $keyCount; Value: PCCM_GetPropertyForFolderContent( "Count" ; "0" )]

    # Process each object…
    # ...
    # ...

    # Now check to see if we have more objects to process
    Set Variable [ $truncated; Value: PCCM_GetPropertyForFolderContent( "IsTruncated" ;
    "0" )]
    If [$truncated = “true”]
       Loop
       Set Variable [ $token; Value: PCCM_GetPropertyForFolderContent(
       “ContinuationToken” ; “0” )]

       # Issue the same request again, with the continuation token
       Set Variable [ $result; Value: PCCM_FetchFolderContents( Objects::BucketID ;
       $prefix ; “” ; “” ; $token )]
       If [$result = “!!ERROR!!”]
          # Error handling…
       Else
          # Process the bucket properties

          Set Variable [ $keyList; Value: PCCM_GetPropertyForFolderContent( “Keys” ;
          “0” )]
          Set Variable [ $keyCount; Value: PCCM_GetPropertyForFolderContent( “Count” ;
          “0” )]

          # Process each object…
          # ...
          # ...

          # Now check to see if we have more objects to process
          Set Variable [ $truncated; Value: PCCM_GetPropertyForFolderContent(
          “IsTruncated” ; “0” )]

          # Exit the loop if there are no more objects to process or the token is
          # empty.
          Exit Loop If [ IsEmpty( $token ) or $truncated = “false” ]
       End If

       End Loop
    End If
End If
```

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
Help Center:	http://help.productivecomputing.com/help_center
Forum:	https://fmforums.com/forum/297-filemaker-utility-plug-ins/

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!