FM Plugin Utility documentation

 Last updated 6/24/2015.

Purpose

This utility installs missing plugins, updates outdated plugins, and registers plugins to be used as part of a login script.

How to Use

You can either import the functions / tables / scripts into your own solution or alternately call this file externally.

When to Use

I highly recommend that the installer/activation procedures kick in as early as possible during the log-in routine. You may want to call this utility first before any other log-in steps as the plugins may need to be installed/registered before the user is authenticated, depending on your system set up. If user authentication takes place first, then this should be the second step.

Advantages

1. Plugin installation routine that does not require hosting source plugins on the server. Ideal for non-server based and vertical market run-time solutions.
2. Secure method of storing registration keys without exposing them to the general user using a custom function. Also avoids having to manually enter registration keys manually on the client side.
3. Robust system for error capture and registration status.
4. Minimal overhead:
    a. 1 table (25 fields)
    b. 1 layout
    c. 1 script (< 100 steps)
    d. 6 custom functions, 3 of which can be used for other purposes

How it Works

Following is a brief summary of the steps that would take place to incorporate this utility.

- User launches main program.
- As part of the log-in script, this utility file is called and it runs the Activate Plugins script (see Caveats).
- Script loops through all the plugin records and installs those that are flagged to be installed/updated.
- After the installation loop, a second loop runs which activates the plugins
- Results (successes and errors) are displayed in a system message window. This example uses the following features, which you will have to remove for FM 12. Versions prior to FM 12 cannot use this technique.
    - FM 13 : Pop overs, Refresh Object
    - FM 14 : Button bar w/ icons, Get(ApplicationArchitecture)*, check mark instead of X for check boxes.

* Get(ApplicatoinArchitecture) is used in the sysArchitecture calculation. If you need to downgrade this utility for FM13 or FM12 because of a mixed system environment, you will need to modify the sysArchitecture calc to return "x32". In addition, if you have users that are on FM12, FM13, and FM14, you will need to make sure that the FM14 users are only using the 32 bit version of FM14. See the *32 bit vs. 64 bit FileMaker Applications* section for details.

Caveats for this Utility File

1. This Activate Plugins script has a prompt to proceed or not. This is to allow you as the developer to look under the hood first and make sure that the file is properly prepared before it used in production. For actual production, you will probably want to bypass this step for typical users.
2. Although example plugins are included, the registration keys are invalid. You will need to supply your own.
3. You may wish to consider what to do with the results. Some ideas to think about:
    a. If there are no errors, there probably shouldn't be any messages or interruption in the log-in routine.
    b. If there are errors, you may want to…
        i. Prevent further progress in logging in or…
        ii. Perhaps track which plugins are disabled and have alternative programming in place to compensate or…
        iii. Just disable those features that rely on the disabled plugins.

Under what conditions will a plugin get installed?

Plugins will be installed if user is accessing the database via:

- FileMaker Pro / Advanced via Windows or Mac
- Web-Direct via browser via Windows or Mac AND plugin marked as WPE compatible

Any other operating system (iPad, iPhone, iPod, Android, Linux, Other) will immediately exit the script because they cannot access the plugin features, so installing/registering them is pointless.

Preparing the Utility

Following is a breakdown of each part of a given plugin record, a description of the fields involved, how they work, and how the system implements them.

System Data

sysPlatform	Calculates if the current operating system is Mac, Win, Linux, iPhone, iPad, Android, etc.
sysArchitecture	Calculates whether the FileMaker Application / WebDirect is functioning as 32 bit or 64 bit.
sysErrors	Global field that tracks the number of errors during the registration loop
sysMessage	Global field that displays the details of the results of the script.
sysButton	This is a bell and whistle display calc for the results button label.

sysPlatform/sysArcheticture

When the utility is launched, it determines the system platform and architecture and using unstored calculations. This information is used to determine which steps of the scripts kick in and which plugin files to install. I don't recommend altering these calculations.

sysErrors/sysMessage

These fields provide information regarding the number of errors and results of the script. They are all automatically calculated / populated with the Activate Plugins script.

sysButton

This bell and whistle button is a simple calc that displays "Errors!" or "Success!" or "Results" depending on the number in the "sysErrors" field. "" = "Results", 0 = "Success!", > 0 = "Errors!".

Plugin Data

pName	This is the name of the plugin. **It should match the name that FileMaker sees when it generates the Get (InstalledFMPlugins) list.**
pDescription	Optional description of the Plugin for internal reference.
pInstalled	Flags whether or not the plugin shows up in the Installed Plugins list.
pPosInList	Shows which value position the plugin has within the Get (InstalledFMPlugins) list.
pStatus	Shows the current state of the plugin: Enabled = installed and functioning Disabled = installed, but not functioning Ignored = installed, but there was an error in loading Missing = plugin not detected
pWPE	Check this box if the plugin can be used by the Web Publishing Engine. These plugins can be registered by Web Direct Clients. They may also need to be installed on the server. Consult your plugin documentation and server manual for details.

pName, pDescription, pWPE

These fields are manually entered by you.

pInstalled, pPosInList, pStatus

These fields are automatically calculated based on the results of the Get(InstalledFMPlugins) function.

Version Data

vPlatArch	Uses platform / architecture to determine which repetition number to use for installing plugins: 0 = Operating System not supported, 1 = Win (32 bit), 2 = Win (64 bit), 3 = Mac (32 bit), 4 = Mac (64 bit)
vPluginFile [r4]	The file container fields. Win field must have the .fmx file or .fmx64 file. Mac field must have the .fmplugin file. The .fmplugin file can only be inserted into the container from a Mac computer. Files cannot be installed if using .zip or .tar compression.
vAvailable [r4]	This is the version of the file that has been loaded into the file fields. For some plugin types, there may be different versions for Mac vs. Windows and 32 bit vs.64 bit.
vRequired [r4]	This flag will force installation of the plugin file, even if the currently installed version is a later version than the one in the plugin file. This should only be done if you know for sure that later versions of the plugin will muck things up.
vAvailable2Num	This displays the numeric conversion of the version number, which takes into account versions that have multiple dots or letters. It's the most accurate method for comparing non-standard version numbering systems.
vInstalled	This shows which version of the plugin is currently installed on the computer.
vInstalled2Num	This shows the numeric conversion of the installed plugin.
vUpdate	This flag will be checked if the system has determined that the plugin file stored in the container field needs to be installed.

[r#] = repeating field with the number of repetitions.

vPlatArch

This field is automatically calculated and is displayed in radio button form with a value list of 0-4.

vPluginfile, vAvailable, vRequired

These fields need to be manually entered by you. **Note that you must upload/insert the Windows plugins from a windows machine and the Mac plugins from a Mac machine, otherwise they will not install properly.**

vAvailable2num, vInstalled, vInstalled2num, vUpdate

These fields are automatically calculated.

Version Checking

Although FileMaker provides a basic template for calculating version numbers and comparisons, I've come across three different types of version numbering systems that require some additional programming: Standard decimal, multi-point, and letter versioning. I found that even the same plugin might switch version types within a series. For consistency, I created a custom function that converts all three types into a standard numeric value that allows for accurate version comparisons. See the *VersionToNumber_cf* custom function description in the custom function section for details.

Version Checking Gotchas

In some cases, you may need to make sure that the user doesn't upgrade to a later version of the plugin, either because there is an issue with the later version, or perhaps you don't have an authorization key for the later version. In such cases, you want to make sure that the plugin supplied in the installer will always take precedence. In order to flag that, I added a "vRequired" flag field. If the vRequired box is checked, the system will always install the provided plugin if that's not the version installed. Otherwise it will only install it if the provided plugin is newer than the one installed.

32 bit vs. 64 bit FileMaker Applications

New in FileMaker 14 are variations in application architecture for FileMaker Pro / Advanced: 32 bit vs. 64 bit, and the plugin needs to be in either 32 or 64 bit accordingly to work properly.

For the Mac there is a single installer that is a dual 32/64 bit application. When run from the dock, it defaults to 64 bit. If launched from the finder, it can launch in 32 bit mode. This may be necessary if not all of your plugins are in 64 bit.

For Windows, there are two separate installers, one for 32 bit and one for 64 bit. You can only use one or the other, and if you need to switch, you will have to uninstall the one before replacing it with the other.

For more information, review this FileMaker Knowledge Base article.

32 bit / 64 bit FileMaker Plugins

In order to account for the Mac/Win 32/64 bit combinations, this solution provides a quad-repeating field set for each operating system / bit combo: vFile, vAvaible, and vRequired. The operating system and architecture are auto-calculated when launched. Some potential "gotchas" you should be aware of:

- Some plugins will have one version number for their Mac based plugin and a different version number for their Windows based plugin. And now with the 32 / 64 bit variations, there may be different version numbers for those as well. So you will need to be thorough.
- Note that plugins for the Mac may be distributed as a 32 bit, a 64 bit or a bundle that includes both in the same structure file. If the bundle includes both, you should copy the file to both 32 and 64 bit plugin file containers.
- In addition, plugins cannot be delivered / installed as .zip or .tar files. They have to be uploaded uncompressed to the container field. However, the plugin will be compressed in the container.
- Remember, **Mac plugins need to be uploaded/inserted from a Mac machine and Windows plugins from a Windows machine.** (Yes, I'm repeating myself).

Requirements Gotchas

Some of the new 64 bit plugins may require additional 3rd party software. For example, the 360 Works line of plugins require a 64 bit version of Java installed. This can be downloaded from here:
https://java.com/en/download/manual.jsp

Double check all requirements from your plugin supplier.

Registration Data

rOrder	This number is the order in which the functions are registered. It is used in combination with the Registration function that is described later. See Registration Process below for moreinfo.
rFunction	This is the function that registers the plugin. Each plugin is going to have its own special function (check plugin documentation). Use a "$" to represent the registration key needed to activate the plugin. The actual key is stored in the *PluginActivation_cf* custom function.
rStatus [r20]	Global repeating field for storing registration results using the rFunction. Session / user specific.
rStatus_Exp	This is the expected status if the plugin registration is successful. It has to be manually entered. The plugin documentation should clue you in on what that status should be.
rStatus_Result	This is the registration result called from rStatus, using the rOrder as the repetition number.
rStatus_Disp	This unstored calc displays "Registered" or "Not Registered" depending on if Result = Exp or not.

[r#] = repeating field with the number of repetitions.

rOrder, rFunction, rStatus_Exp

These fields need to be manually entered by you.

rStatus, rStatus_Result, rStatus_Disp

rStatus is populated via the Activate Plugins script. *rStatus_Result* and *rStatus_Disp* are automatically calculated.

Registration Process

Registration uses a custom function (*PluginActivation_cf*) that holds the registration keys and assigns them using a "choose" and the *rOrder* field. A repeating global field (*rStatus*) then stores the results of the registration function based on the order. Then, each plugin record displays the result using and unstored calculation that refers to the global repeating *rStatus* field with *rOrder* repetition number. **Therefore, it is important that for each record, rOrder be unique, in sequence, a positive integer, and not 0.**

If the user is launching via WebDirect, only plugins flagged as rWPE (Web Publishing Engine) compatible will get registered. In some cases, WPE plugins may also have to be loaded on the server. Check with the plugin designer.

Custom Functions

There are six custom functions that are provided with this utility. Three of them are specific to the plugin features. The other three can be used for other purposes. Following is a breakdown of each function, how they work, and how to use them.

Generic Custom Functions

These three custom functions are useful in general, and can be utilized for other purposes beyond this utility.

IsInList_cf (_List ; _Value)

This function returns 1 if the _value is found within _List or 0 if not.

PosInList_cf (_List ; _Value ; _Pos)

This function returns what position/order a _Value has in _List. If the value is not in the list, it returns 0. If the value is in the list multiple times, it only returns the first occurrence.

StripParam_cf (_text ; _param# ; _separator)

This function returns _param# nth item in a _text string using _separator as a delimiter. For example:
StripParam_cf ("Trees^Flowers^Birds,Bees" ; 3 ; "^") = "Birds,Bees"
StripParam_cf ("Trees^Flowers^Birds,Bees" ; 1 ; ",") = "Trees^Flowers^Birds"
StripParam_cf (StripParam_cf ("Trees^Flowers^Birds,Bees" ; 3 ; "^") ; 1 ; ",") = "Birds"

Plugin Oriented Custom Functions

These three custom functions are specifically targeted for plugin installation and registration.

PluginList_cf (_list ; _i)

This function returns a list of all the installed plugins from the *Get(InstalledFMPlugins)* function, but stripped of all data except for the names of the plugins.

VersionToNumber_cf (_version)

This function converts a "version number" to a consistent numbering system, regardless of the version system used. It does this by converting all versions into a multi-point number.

- If there is an "r", it will substitute it with a decimal point.
- Then, if there is only one decimal point, it will add ".0" to it.
- Then it will convert and concatenate the number as follows: x.y.z -> xyyyzzz inserting 0s where needed.
- You may need to tweak the function if your plugin uses some other versioning system.

Here are some examples:

Version Numbering Type	Original Version Number	Converted Number
Standard Decimal	1.972	1972000
	1.97	1970000
	1.9	1900000
Multi-point	4.1.2	4100200
	4.11.35	4110350
Letter versioning using "r"	1.7r1	1700100

PluginActivation_cf (_PluginNum ; _RegFunction)

This function returns the results of the plugin's registration function by substituting $ with the registration key stored in the custom function. This allows you to include the keys as part of the solution but restrict direct access to them by anyone other than an administrator, should you choose to set up the system that way.

_PluginNum in this utility should be the registration order (rOrder) in the plugin record. More info can be found in the custom function itself.

WebDirect/WPE Caveats

The functionality of the installer / registration technique when using WebDirect is my best guess based on the available documentation. I haven't had enough time to really do any significant testing with WebDirect, plugins, WPE and the plugin installer features. So you may need to experiment a bit. If you have any experience or advice on the matter, feedback will be very welcome.

Questions / Feedback

I hope this utility has been helpful. If you have any questions / feedback, please feel free to contact me and I will get back to you as soon as possible. I can be e-mailed at solutions@cordega.com.

Chris