

Design Pop-Up Interfaces

They're not just cool; they're intuitive.

by Geoff Ryle (geoff_ryle@excelisys.com)

I have always been a fan of those cute little pop-up routines. Take a popular application like Microsoft Outlook, for instance: when you need to enter a date, a little button in the form of a calendar icon appears next to the field which, when clicked on, brings up a small "floating" window of a calendar. You can move the window around, navigate from month to month, then click on a date and return to the original window with the date you selected automatically entered into the date field. How cool is that?

This sort of calendar navigation is nothing new, but presenting it as a pop-up interface is seldom done in FileMaker Pro solutions. Typically, I've seen calendar navigators integrated into the same layout in which the date field appears. There's nothing wrong with this, of course, unless you're desperately trying to free up some valuable screen real estate. And sometimes the calendar navigation interface has nothing to do with the other parts of the layout in which it resides but serves only to confuse the user. If your goal is to require less knowledge on the part of the user regarding the inner-workings of FileMaker and wish to make their experience as simple and smooth as possible, this article will teach you a technique that will do just that and give your solution a more professional look.

Scripter's List

Another good example of where pop-up interfaces are useful is when the user needs to choose from a list of names, or products, etc., during a data entry or search setup process. You won't necessarily want such a list to appear as a portal in the same layout as the data entry fields. This can tend to slow the system down if you're going to display a large number of entries and/or the data in the list is being shared over a network. There is also an issue regarding record locking. If you are editing a record in a layout with a portal showing records in another file, FileMaker automatically locks the first related record in the portal, thus making it impossible to edit that record, even though it's in a different file.

It's much more elegant and intuitive to give the user an option to bring up the list if desired. When they do, it can appear as a smaller window that "floats" on top of the main layout by bringing up the file that contains the actual data, rather than using a portal of related records.

The trick is all in how you script your pop-up interface, so the user understands how to proceed and doesn't accidentally stumble in or out of it. Specifically, you will be using the "Pause/Resume Script" step, but one must be very cautious in its implementation. There are few things more embarrassing for a programmer than to have a script indefinitely paused while the user tries to stumble through your solution.

Before I continue, let me point out that the following example is just one of many possible ways of incorporating a pop-up interface. As always, FileMaker provides you with many ways of doing any one thing. So before you FileMaker aficionados out there start writing me flame email (although I do love the attention), I will admit in

advance there may be a better way of accomplishing what I propose. Nevertheless, it's the result that matters most, which is to provide a more elegant and intuitive interface for the user, heaven help him or her.

Fields of Dreams

Our two-file solution is a very simple one. You can choose to build them yourself, or you can download the sample files I've pre-constructed and learn by example. The sample files are available for downloading at the FileMaker Pro Advisor Magazine website, located at <<http://www.advisor.com/www/FileMakerProAdvisor>>.

File #1 in our example has only two fields, and file #2 has three. Table 1 shows how you should define all the fields.

Table 1: A field day—The definitions for the fields in File #1 and #2.

Field Name	Type	Options
* File #1: Entry *		
tName	Text	
cnConstant	Calculation	Number Result, Indexed, = 1
* File #2: List *		
tName	Text	
gtName	Global	Text
cnConstant	Calculation	Number Result, Indexed, = 1

The file names shown here incorporate a naming convention that indicates the field type. For example, the field "tName" begins with "t" to indicate it is a text field, and the field "gtName" begins with "gt" to show it is a global text field. And "cnConstant" begins with "cn" to let you know that it is a calculation field with a number result. It's a good idea to integrate such a naming convention into all your solutions, as it can become very helpful when working on files that sport dozens if not hundreds of fields that may share similar names.

As I said before, there are a few different ways of implementing this technique. One such "fork in the road" is how to move data selected in file #2's list over to the data entry layout of file #1. I personally like to avoid copying and pasting data using the system's clipboard. This is because doing so will blow out anything the user may have previously copied to the clipboard, which in some cases is something they wanted to keep there. By creating a field called "cnConstant" in each file, and defining both as a calculation with a number result equal to 1, you can create a universal relationship between the two files. Then, you can use this universal relationship to move the selected data over from one file to the other by the "Set field" step in a script, thereby bypassing the clipboard altogether.

Relationships to Layout

The relationship you need in order to move the selected data will be created in file #1. Similar to using a field-naming scheme, you should name the relationship in a manner that will indicate both the local and foreign key field names as well as the related file's name.

Relationship Name:	cnConstant cnConstant [List]
Relationship:	cnConstant = ::cnConstant
Related File:	List.fp5

This is your universal relationship, meaning it will always relate the first file to the second regardless of which record is selected. The only time the relationship is invalid is when 0 records are present in the file, there is a found set of 0 records, or if the user enters Find mode. If you need the relationship to be valid throughout each of these scenarios, you should define the local key field in the relationship not as a calculation but as a global field with a number result (in which case you would give it the name "gnConstant"). However, you would have to make sure this field was always set to a value of 1. This is done by writing a script (set to execute upon opening the file) that would use the "Set Field" step to enter a "1" into this global field.

Next, you need to create a few basic layouts. File #1 needs only one layout for data entry, and should include the "Name" field and a button sporting a clever icon (something that hints as to what's in store for the user should they click here) that will allow the user to bring up the pop-up interface.

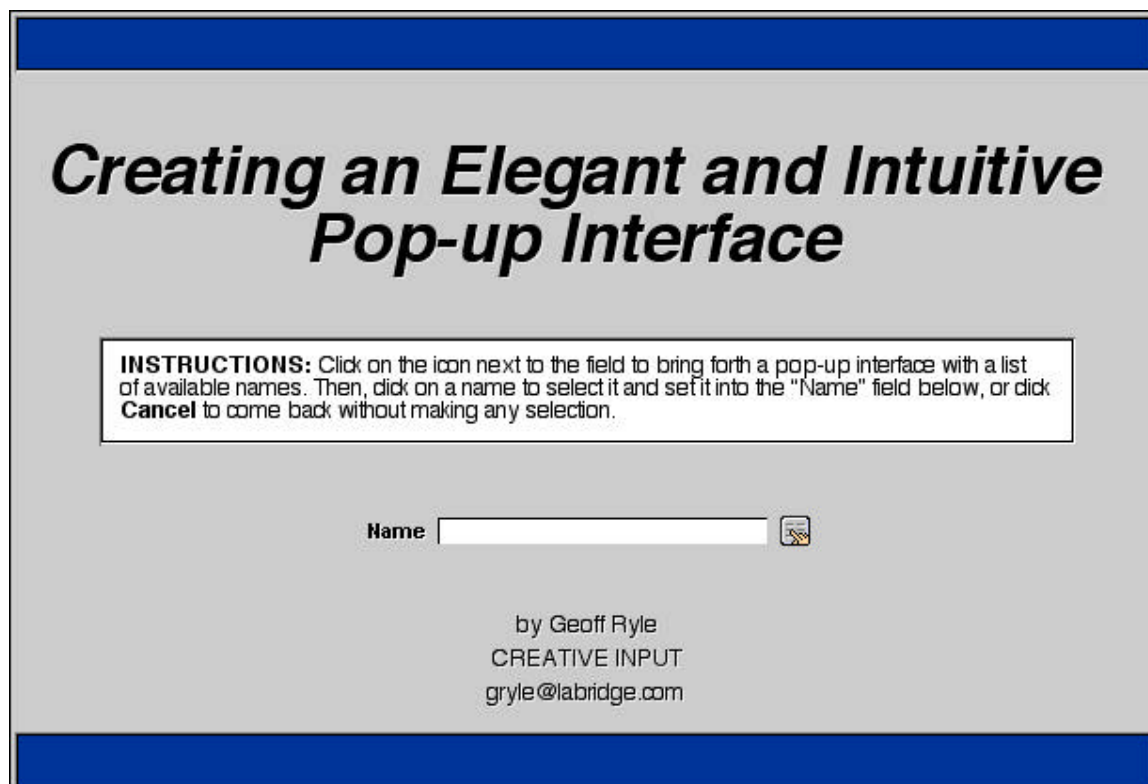


Figure 1: Basic layout—A basic layout is all that's necessary for data entry in file #1, with a button for selecting the pop-up interface. Be sure to choose a button that clearly indicates its purpose to the user.

File #2 needs two layouts: one blank layout to default to upon opening, and a list layout to display the records in this file. The blank layout should have some text explaining there's nothing the user can do with the file except hide it, should they happen to bring it forward manually via the Window menu.

The list layout in file #2 is at the heart of our pop-up interface. It should have a header, where you can type instructions on what they should do to proceed. ("Click on a name in the list to continue.") Place the "tName" field in the body of the layout so that it completely fills the vertical space within the part, and then format it to not allow user entry. Finally, place a button in the footer section that the user can use to cancel out of the list without making a selection. Once you have the basic list layout ready, add a few records with some names to use for testing purposes.

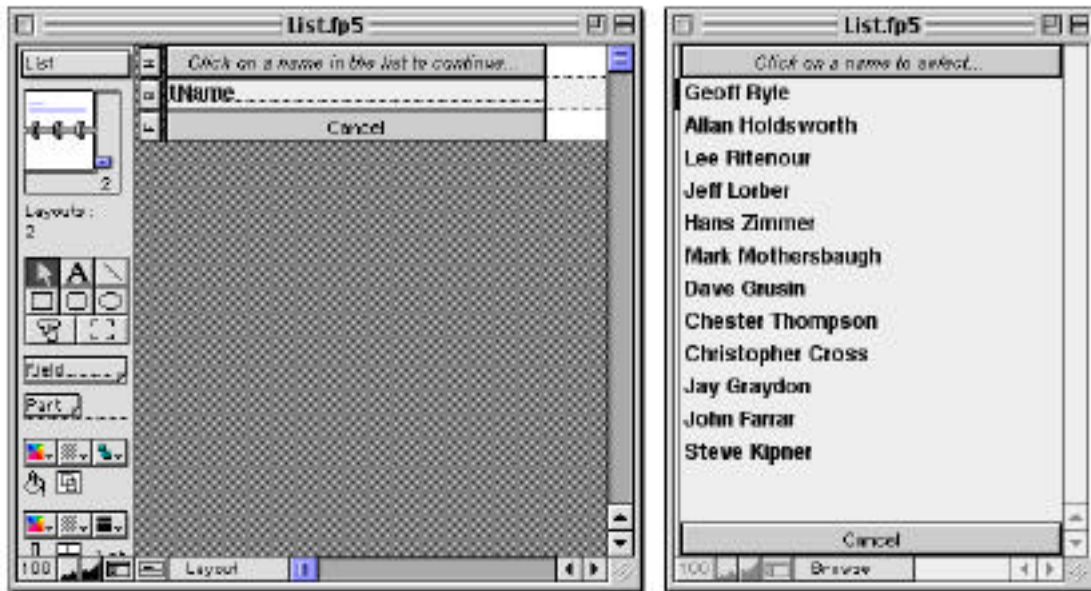


Figure 2: **List layout**—Your list layout should provide instructions to the user on its use, with an option for canceling out without selecting from the list.

Scripts Institute

The first file will require two scripts: one for calling up the pop-up interface, and one to come back to after a valid choice has been made from the pop-up interface. The second file requires three scripts: one for coming to the list of names, one for returning to the first file after selecting a name, and one for resetting the window to the "blank" layout. This last script is important, because we don't want users who manually navigate to the second file via the Windows menu to be able to choose from the list.

The two scripts in file #1 will go like this:

Script Name: Go to Pop-up List

Perform Script [Sub-scripts, External: "List.fp5", "Come to List"]

Script Name: Return

Set Field ["tName", "cConstant|cConstant [List]::gtName"]

The three scripts you need to write in file #2 will go like this:

Script Name: Come to List

Allow User Abort [Off]

Go to Layout ["List"]

Pause/Resume Script

(Note: Leave the "Allow User Abort" step set to "On" until you have fully implemented this technique. Otherwise, you could find yourself caught in an script with no way out.)

Script Name: Choose Name

```
Set Field ["gtName", "Name"]  
Perform Script [Sub-Scripts, "Reset Window"]  
Perform Script [Sub-Scripts, External: "Entry.fp5", "Return"]
```

Script Name: Reset Window

```
Go to Layout ["Blank"]  
Toggle Status Area [Hide, Lock]  
Set Zoom Level [100%, Lock]  
Toggle Window [Zoom]  
Toggle Window [Hide]
```

(Note: It is important that you enable the "Lock" option for the "Toggle Status Area" step to prevent the user from manually navigating to the other layout.)

Take Your Pause Off Me

Now you need to go back to your layouts and define your buttons. Obviously, the lone button in file #1's data entry layout will be assigned to the script "Go to Pop-up List". In file #2, define the name field in the List layout as a button, and assign it to the script "Choose Name". Then, assign the Cancel button in the footer to the script "Reset Window". You should also consider adding a button to the blank layout in file #2 and assigning it to the "Reset Window" script.

The "Pause/Resume Script" step (in the "Come To List" script in file #2) is important in this technique because it ensures that the user makes a valid choice (choose a name or cancel), which always results in file #2 reverting to its "blank" layout and hiding itself. Later, after you've tested your work and are confident it performs properly, you will set the "Allow User Abort" step (in the same script) to "On" so the user won't be able to manually cancel out of the script. They won't be able to access any other visible FileMaker file (by clicking on it or selecting it from the Window menu) until they continue the script in the only two ways you've provided.

Since you are using the "Pause/Resume Script" step to get to your pop-up interface, it is important to cover your tracks so the user has no chance of unwittingly being stuck in an eternal pause. To accomplish this, be sure to set your buttons in file #2's list layout to "Exit" the current script.

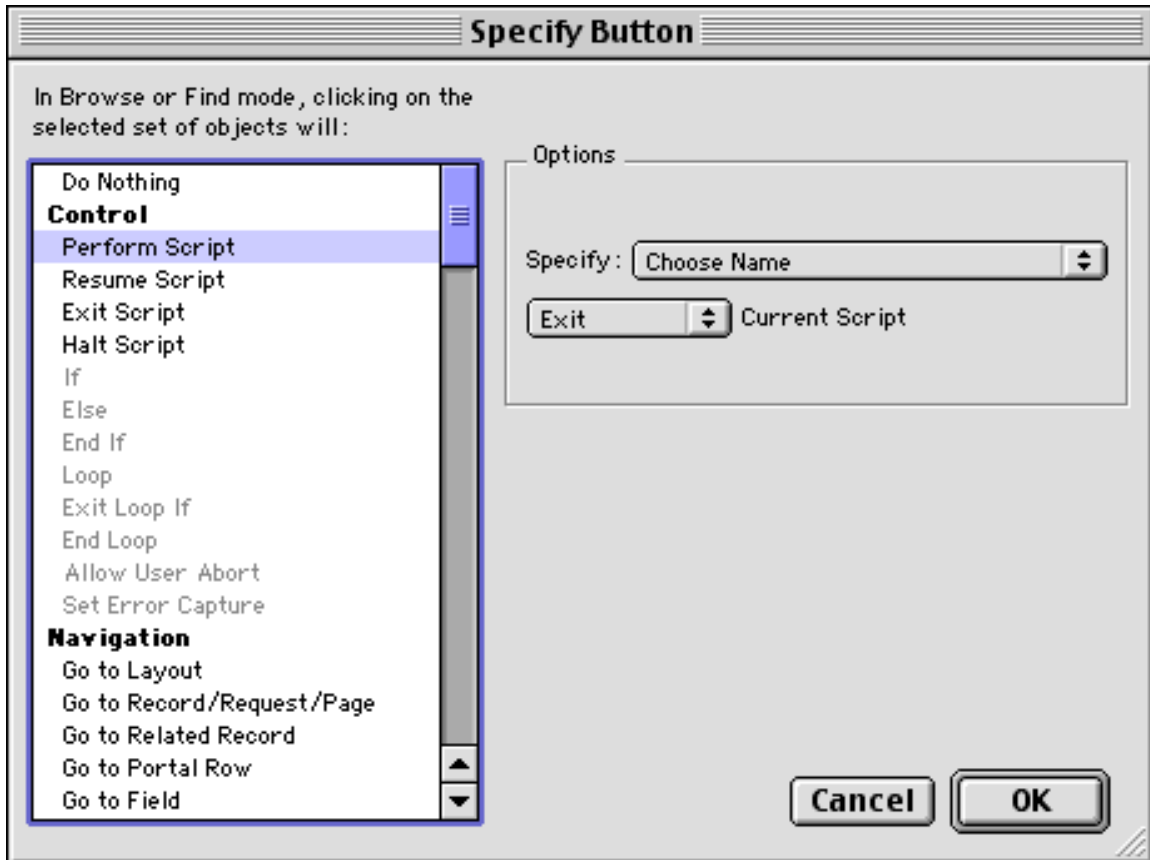


Figure 3: **Free the user**—Be sure your buttons are set to "Exit" the current script, or your users will be stuck in an eternal pause.

All Together Now

Once all of the above steps have been completed, you can test out your version of this technique. If everything works right, go back into the "Come to List" script in file #2 and set the "Allow User Abort" step to "On". Otherwise, the user will be able to abort the script and possibly move out of the pop-up interface in an improper way. If they do, file #2 will not reset as it should to prevent unauthorized access. When setup correctly, the user will have no choice after accessing your pop-up interface but to either select an item from the list or click the "Cancel" button.

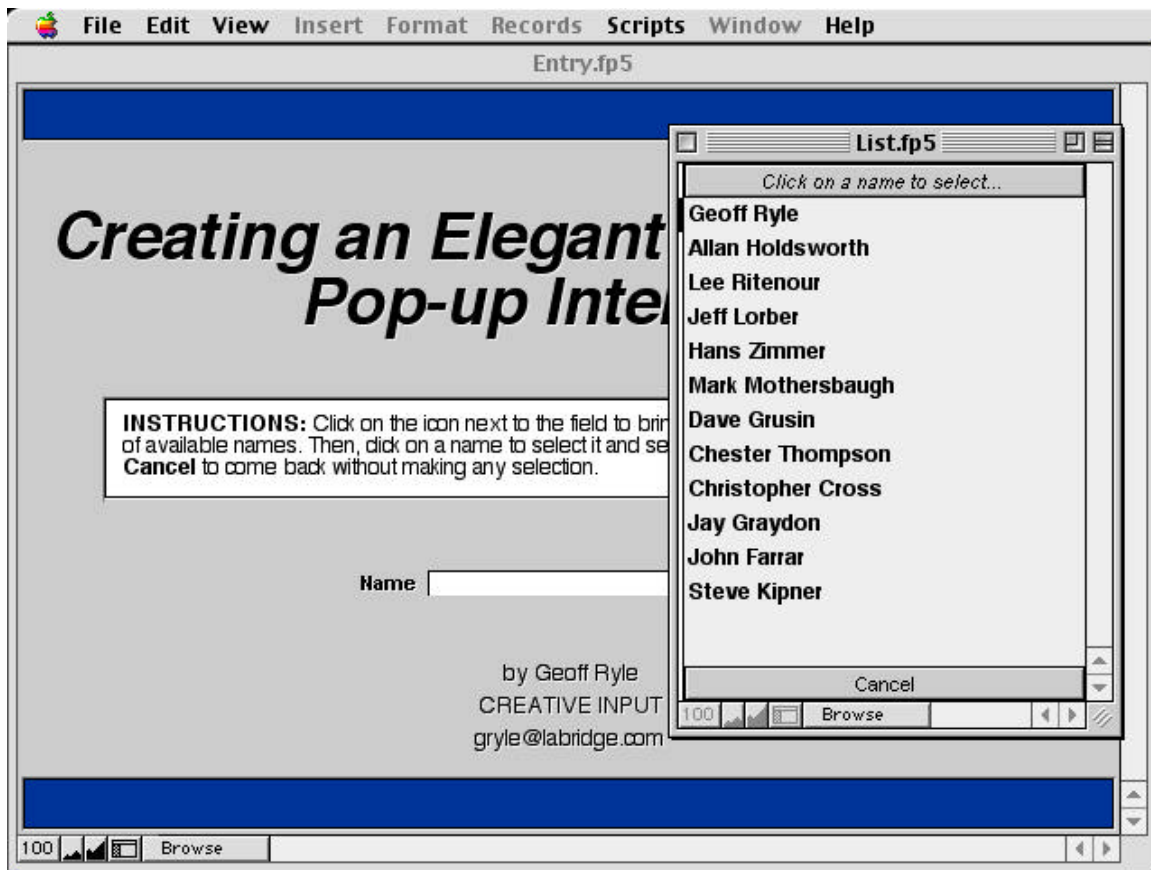


Figure 4: **User choices**—With its window sized properly, the list will appear to temporarily "float" above the first layout. Notice that several menus at the top are in gray indicating you are paused in a script, leaving no other options but to select a name or click Cancel.

This technique can be expanded in many ways. The best part is that, if the user prefers to move the window to a different position on the screen, it will reappear in the same place they moved it to the next time they bring it up.

That's a Wrap

Using a pop-up interface saves valuable space in your layout, but also lets the user move the window around if they need to see what's "behind" it. There are other less complex methods for letting the user select from a list, such as a portal. But by using the technique I've described here, you will provide an interface that is less cluttered, more intuitive, and adds style to your solution.

Using the "Pause/Resume Script" step, you'll ensure that the user can only respond in a manner of your choosing and the pop-up interface window will reset to its benign state as it should, thus preventing accidental or intended misuse of your solution. And best of all, the user will (hopefully) be impressed with how simple and intuitive your solution is. If they aren't, you can send them to me and I'll slap 'em around until they see just how talented a programmer you truly are.

###