.com Solutions Inc.

LiveCode Converted - Contacts.fmp12 ReadMe

LiveCode Converted - Contacts.fmp12 ReadMe

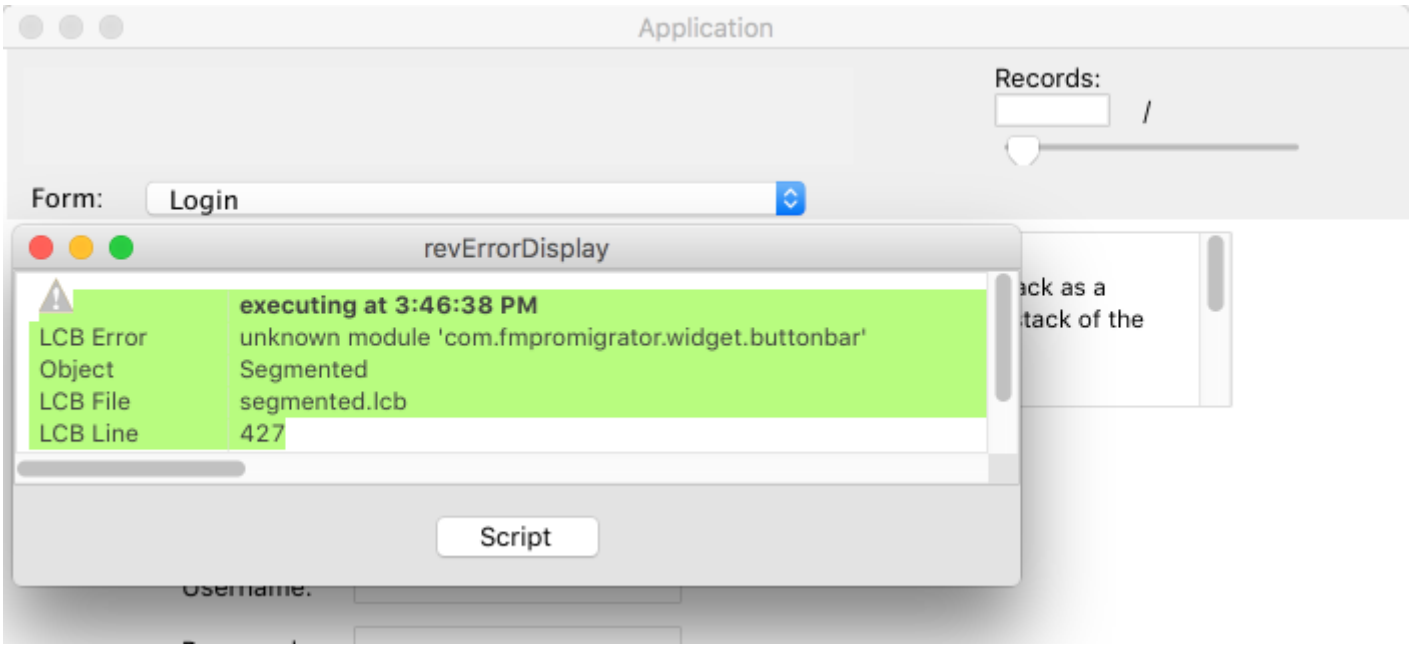
1 Contacts.livecode Conversion Details

Contacts.livecode
Conversion Details

Adding the Button Bar Widget

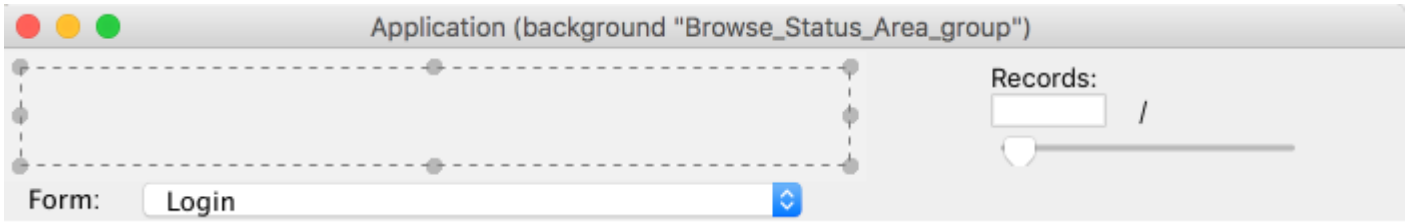
FmPro Migrator will add the Button Bar widget to your new stack file automatically - and it will be displayed during the conversion process. But this widget needs added to the LiveCode IDE before it will be displayed within the stack from within the LiveCode IDE. Follow these instructions to install the Button Bar widget.

Button Bar Widget - Unknown Module Error



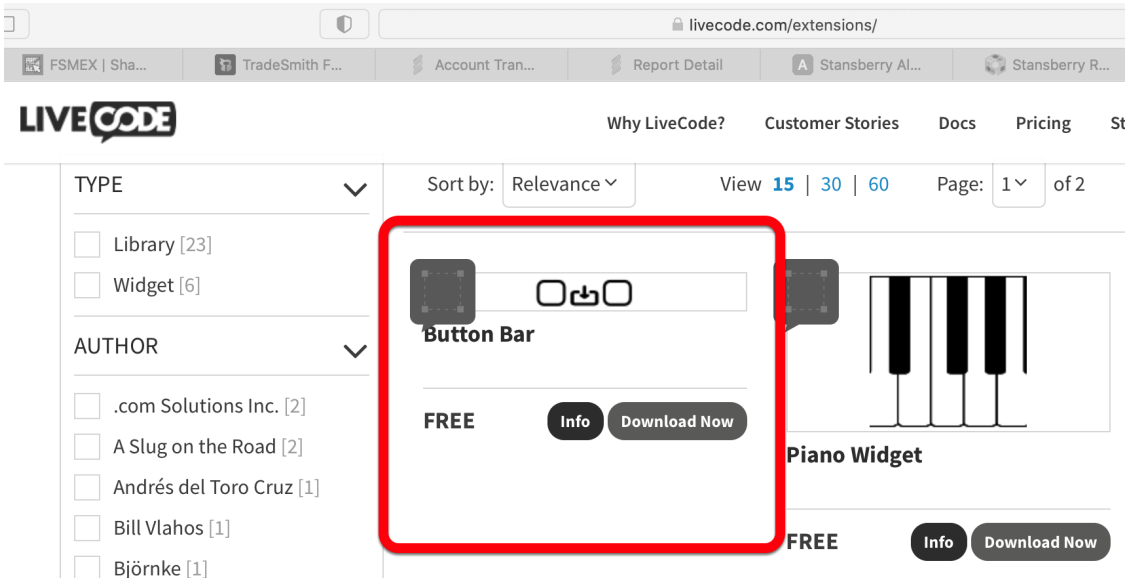
If you open the newly created Application.livecode stack before installing the Button Bar widget, you will see the Unknown Module error dialog.
This error will not occur during the actual conversion process because the Button Bar widget is built into FmPro Migrator and is displayed on screen during the conversion process.

Button Bar Widget - Uninstalled Widget Outline Displayed in Browse Status Area Group



If you close the error dialog and edit the Browse Status Area/Find Status Area groups, you will see the Button Bar widget outline when clicking on it.

Download Button Bar Widget - LiveCode Store Web Site



The Button Bar widget can be downloaded from the LiveCode store at
https://livecode.com/extensions/
Download and unzip the Button Bar widget folder.

The Button Bar widget is not yet available within the store link in the Extensions Manager of the LiveCode IDE.

Download Button Bar Widget - FmPro Migrator Web Site

FmPro Migrator Developer Edition for macOS - FmProMigrator9.14DEmacOS.dmg 38 Mb
This version of FmPro Migrator is supported for use with macOS version 10.9 and higher.

FmPro Migrator Developer Edition for Windows: 64-bit - FmProMigrator9.13DEWindows_64bit.zip 38 Mb
FmPro Migrator Developer Edition for Windows: 32-bit - FmProMigrator9.13DEWindows_32bit.zip 36 Mb
This version of FmPro Migrator is supported for use with Windows Vista/Windows 7/Windows 8/Windows 10.

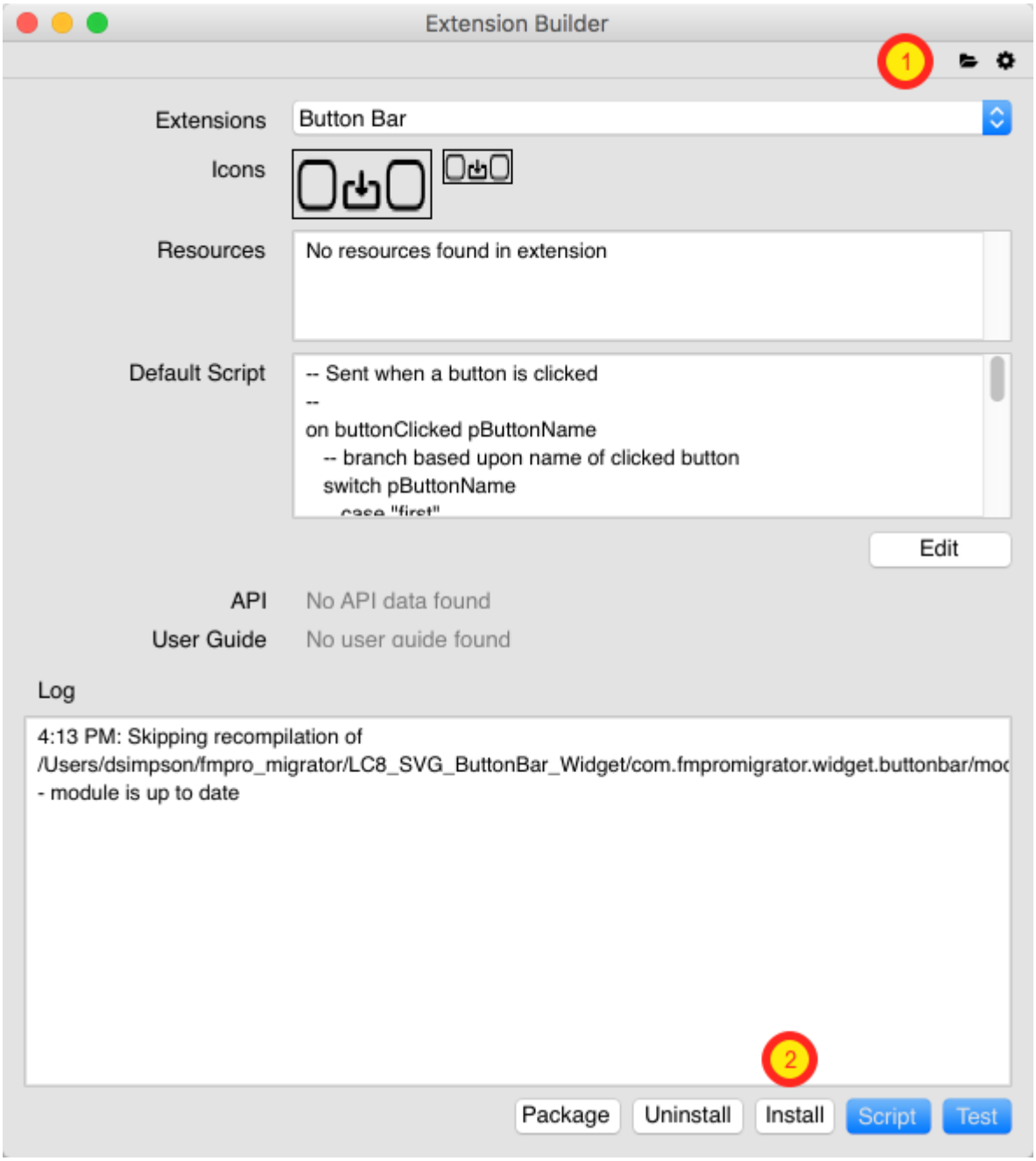
Table Consolidation Layout Troubleshooter [Used for FileMaker Table Consolidation Projects] [PDF Manual]
macOS - TableConsolidationLayoutTroubleshooter1.05macOS.dmg
Windows - TableCosolidationLayoutTroubleshooter105Windows.zip

LiveCode 9 Button Bar Widget [Used for LiveCode Conversion Projects]
com.dotcomsolutionsinc.widget.buttonbar_101_LC9.zip

The Button Bar widget can be downloaded from the same web page where fully functional versionFmPro Migrator was downloaded (not the Free Trial version). Download and unzip the Button Bar widget folder.

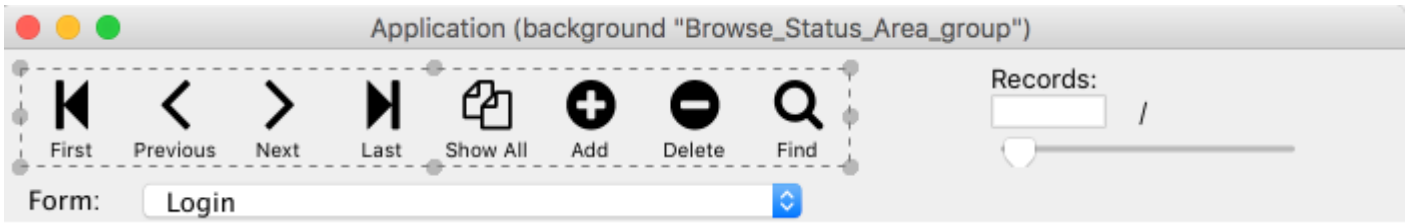
The Button Bar widget is not yet available within the store link within the Extensions Manager of the LiveCode IDE - but it is available from the LiveCode store website.

Extension Builder Window



From within the LiveCode IDE, select the Tools -> Extension Builder menu item. (1) Click the folder icon in the upper right corner of the Extension Builder window, select the buttonbar.lcb file from the Button Bar widget folder. (2) Click the Install button.
Quit and re-open the LiveCode IDE so that the dictionary entries are displayed for the new widget you have just installed.

Button Bar Widget - Displayed in Browse Status Area Group



Once the Button Bar widget has been installed, open the newly created Application.livecode LiveCode stack to see it displayed.

Button Bar Widget - Displayed in Find Status Area Group

Clicking on the Find button in the Browse Status Area group, will display the Find Status Area group.

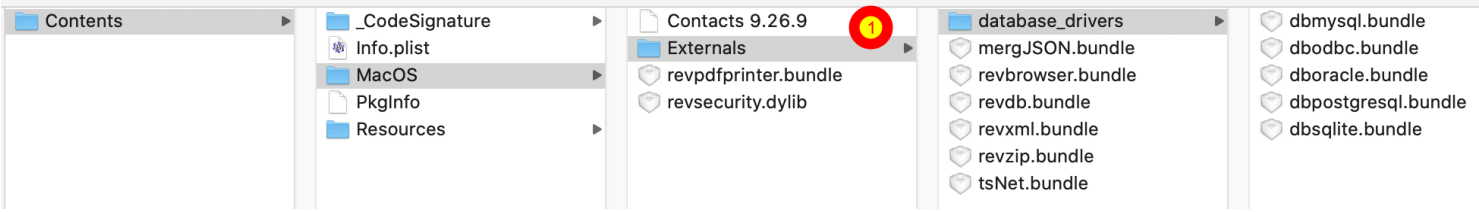
Overview - Converted Contacts.fmp12 Database

Included Files

The files in this package include:
Contacts <version>.livecode
Contacts.EXE Windows 64-bit application
Contacts.app macOS application.
LiveCode Converted - Contacts ReadMe.pdf (this manual)
FmPro Converted Scripts.txt
FmPro Original Scripts.txt
MigrationProcess.db3
Missing Relationships Report.xls
Relationships.JSON
SQLColumnTypes.JSON
SQLiteDB.db3
Stored_Calc_Library.livecodescript
Unstored_Calc_Library.livecodescript

FmPro Migrator 9.26
2/11/2021

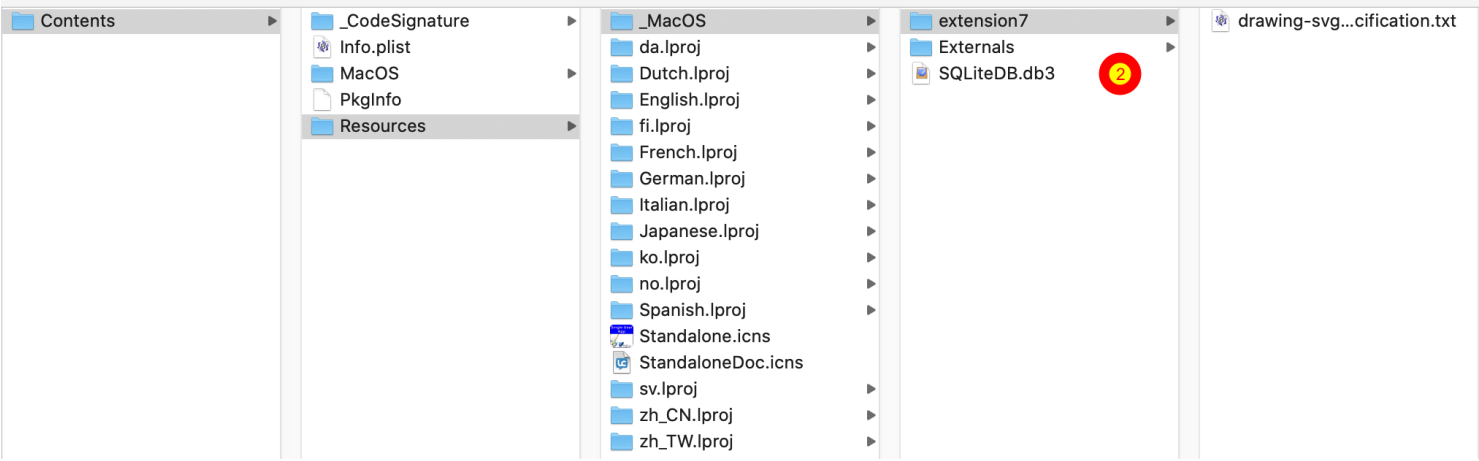
Structure of Standalone App - macOS



Inside the macOS app bundle, the (1) standalone app is located within the /Contents/MacOS path shown above.
This path also includes PDF printing and security externals.
The Externals folder contains additional externals for JSON, XML, ZIP and tsNET networking.
Database drivers are included in the database_drivers subfolder.

Note: Your own application will contain a different selection of Externals and libraries based upon the features you selected in the standalone builder.

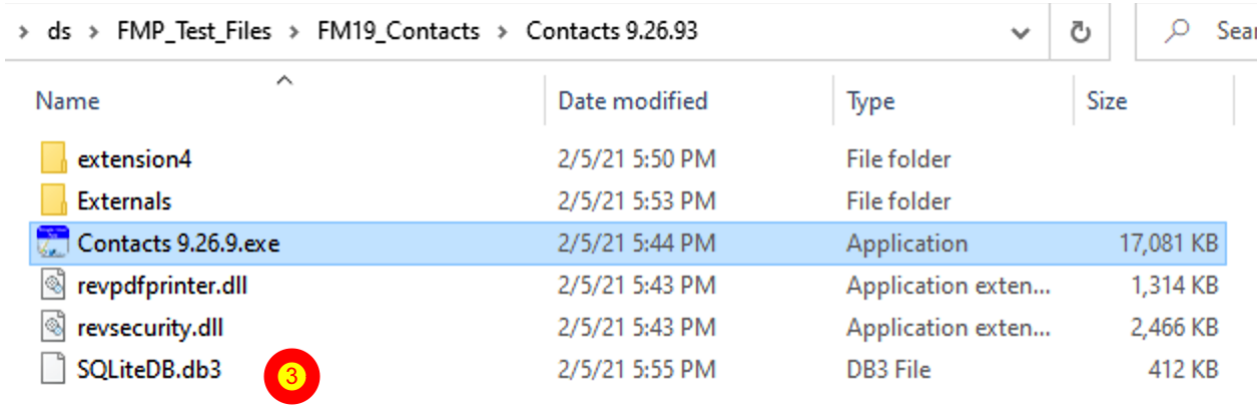
Structure of Standalone App - macOS - SQLite



For this demo app, the (2) SQLite database is stored within the Contents/Resources/_MacOS folder as required by Apple guidelines.

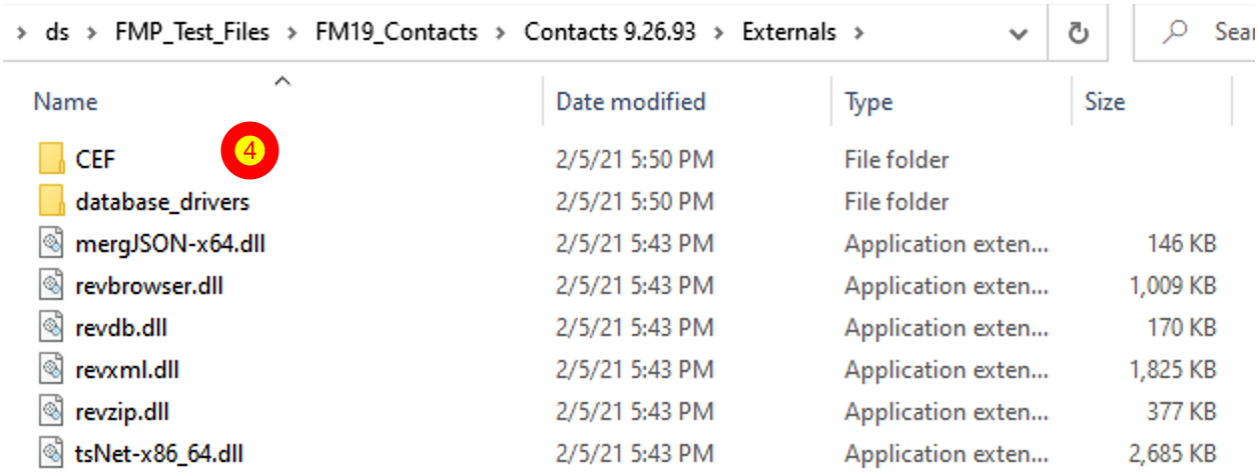
Note: For a single-user demo app which isn't going to get updated often, it is Ok to store the SQLite database inside the app bundle.
But for apps where the application may change on a regular basis - it would be better to store the SQLite database somewhere else, such as a folder named for the application within the Documents directory. Keeping the database file separate from the application will insure that the user's data doesn't get deleted when updating the application.

Structure of Standalone App - Windows



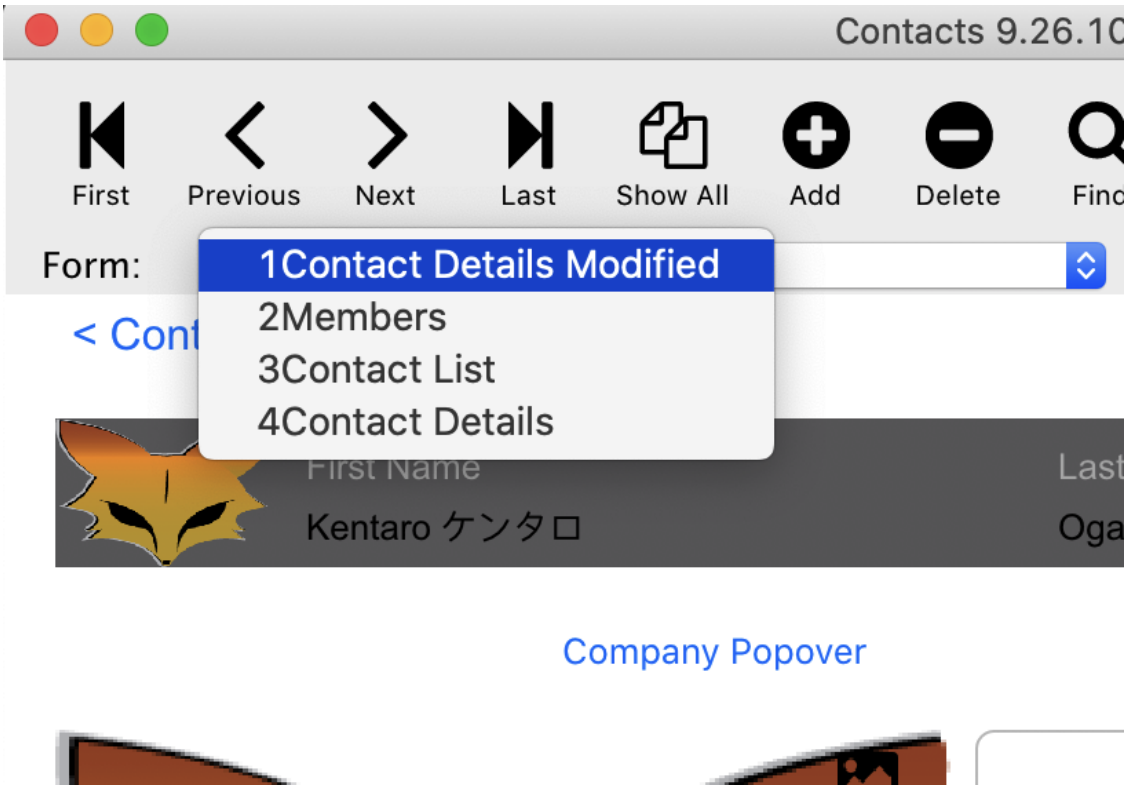
On Windows, the standalone EXE is located directly inside the folder created by the standalone builder. The (3) SQLite database is at the top-level of the folder along with the EXE.

Structure of Standalone App - Windows - Externals



The Windows app has database drivers, externals and a (4) CEF folder inside the Externals folder. Inside this sample app, there is a browser object which requires the addition of the 148mb sized CEF (Chromium Embedded Framework) folder.

Layouts Converted



1Contact Details Modified - this is a version of the Contact Details layout which has been modified in FileMaker prior to conversion.
2Members - The unmodified version of the Members layout.

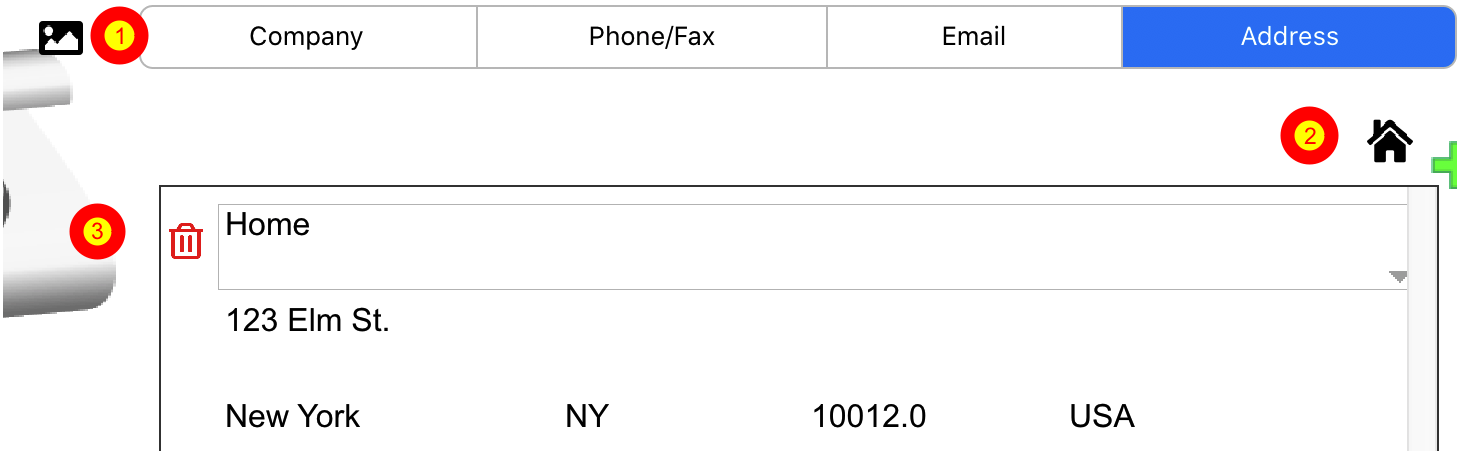
3Contact List - The unmodified Contact List layout.
4Contact Details - The unmodified Contact Details layout.

Note: The numbers at the left of each layout aren't significant other than for showing these converted layouts in alphabetical order in the list of layouts. FmPro Migrator processes the layouts in alphabetical order and also sorts them in the same order when creating the drop down Form menu for the status area. This list of converted layouts (which are now cards in the LiveCode stack) can easily be modified after conversion.

SVG to PNG Button Images Changes

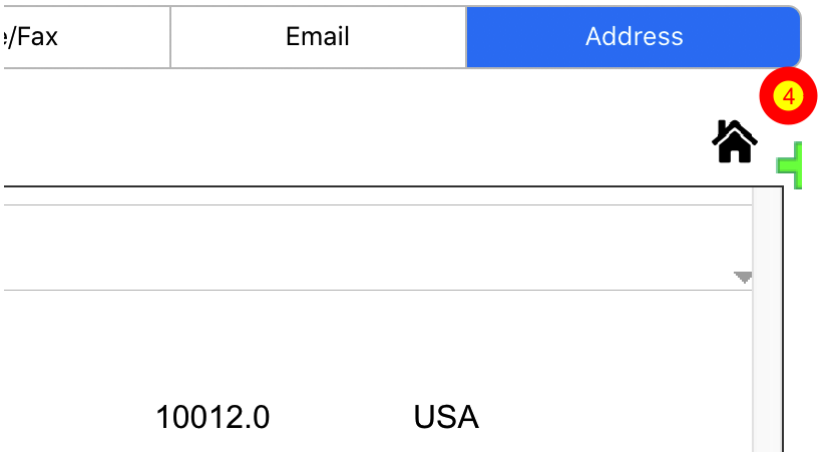
This layout has been modified in FileMaker to improve the conversion process.

SVG Buttons Replaced with PNG Files



In this screenshot of the converted layout shown in the LiveCode app, the SVG button bar type buttons have been replaced with PNG files in the FileMaker database, prior to conversion. At this time FmPro Migrator isn't able to convert the SVG based images, though the outline of an invisible button will get created.

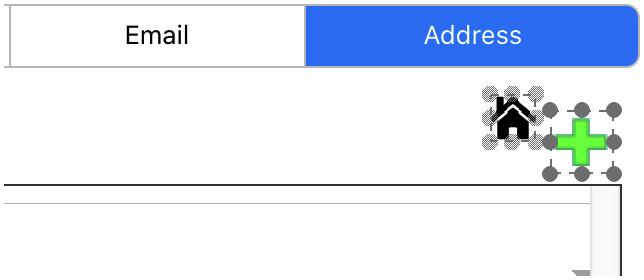
DataGrid Add Button



The green + button at the top right of the DataGrid adds a new blank record to the grid. Auto-Enter fields are calculated and inserted into a new record.

(4) Notice that the green + button is clipped at the right side. This is due to the object being located within the segmented group containing the objects within the Address tab of the original tab control.

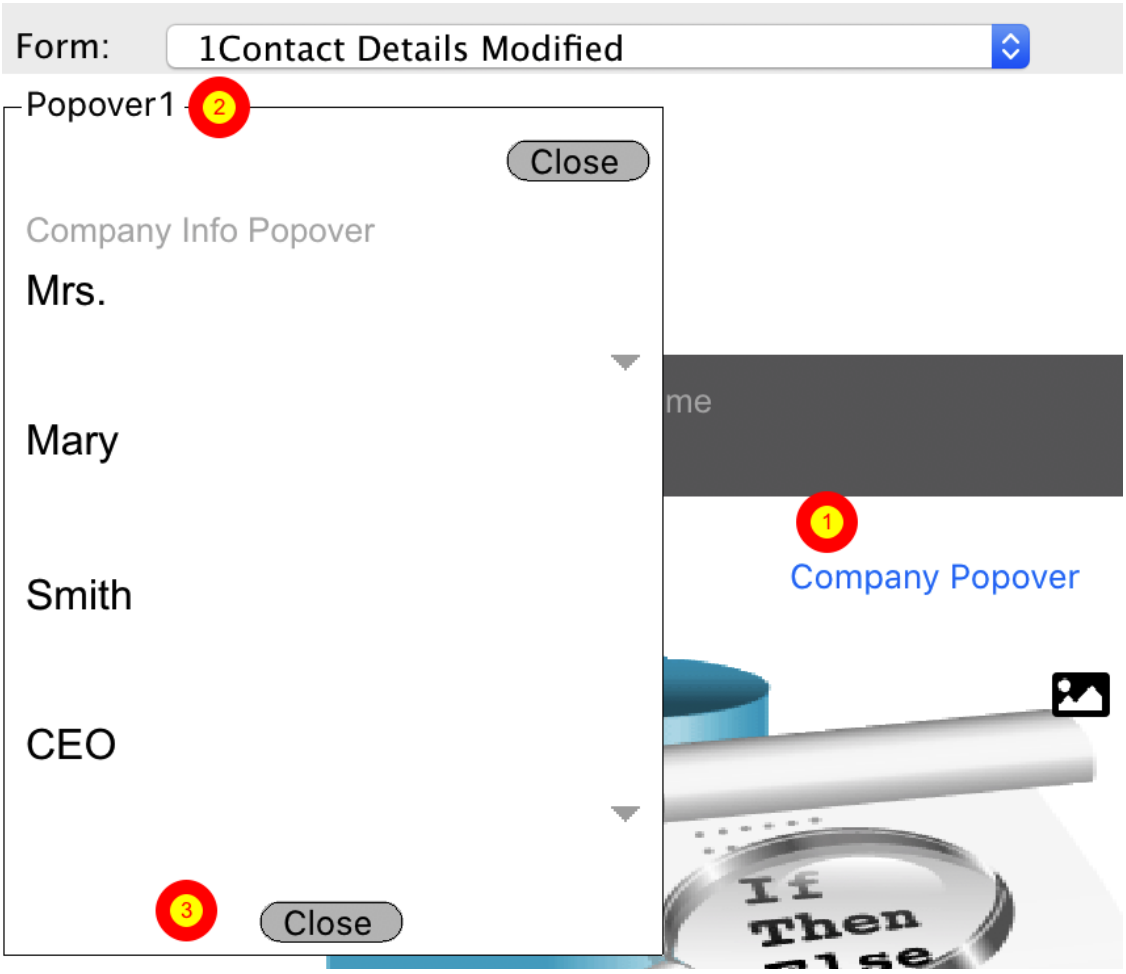
Moving DataGrid Add Record Button



To solve this problem, it is easy to edit the contents of the group in the LiveCode IDE to move these two objects up by 10 pixels and over by about 40 pixels. You would want to make the same change to the other groups representing the other tabs, so that the objects don't appear to move around as the user clicks on the individual tabs.

Popover Implementation Details

Popover buttons are implemented as LiveCode groups. A popover is shown by clicking the popover button, and it is hidden by clicking the close button inside the popover. When a popover is hidden, it is moved over to the right side of the card.



(1) Clicking the popover button (2) opens the popover on the 1Contact Details - Modified card. There are (3) Close buttons at the top and bottom of the popover group. In FileMaker, these buttons link to the single line instruction "Close Popover" and are automatically converted to the following script:

```
on mouseUp
 closePopover the short name of the owner of me
end mouseUp
```

Note: One difference with this implementation compared with FileMaker, the popover is not a modal window, because it isn't a window it is simply a group of objects which gets moved into position and made visible to the user. Fields inside the popover are usable, and field data edited

in the popover group is automatically written to the database.

Popover showLoc position

By default, FmPro Migrator selects a location where the popover is to be shown. This might not be the perfect location, but it can be easily changed. For the example shown, the popover displays a little too far left and is too high. This causes some unexpected movement of the scrollbars when moving the cursor.

To change the showLoc location, select the View -> Show Invisible Objects menu item from the LiveCode menus. This will display the red rectangle object named Popover1_showLoc. This object has its coordinates locked by default, but it can be unlocked and moved anywhere the developer desires.

Why is the showLoc rectangle visible with the opaque setting of the group being true?
The reason is due to the layer setting for the objects. The popover group is set to layer = 285, because it gets created first, then the objects in the popover group get created, the hideLoc rectangle is created with layer = 297, and then the showLoc rectangle is created and set to layer = 299. The layer value effectively defines the z-ordering of the objects. So objects with a higher layer number sit above the objects at a lower level. This property can be easily changed as needed.

Popover hideLoc position

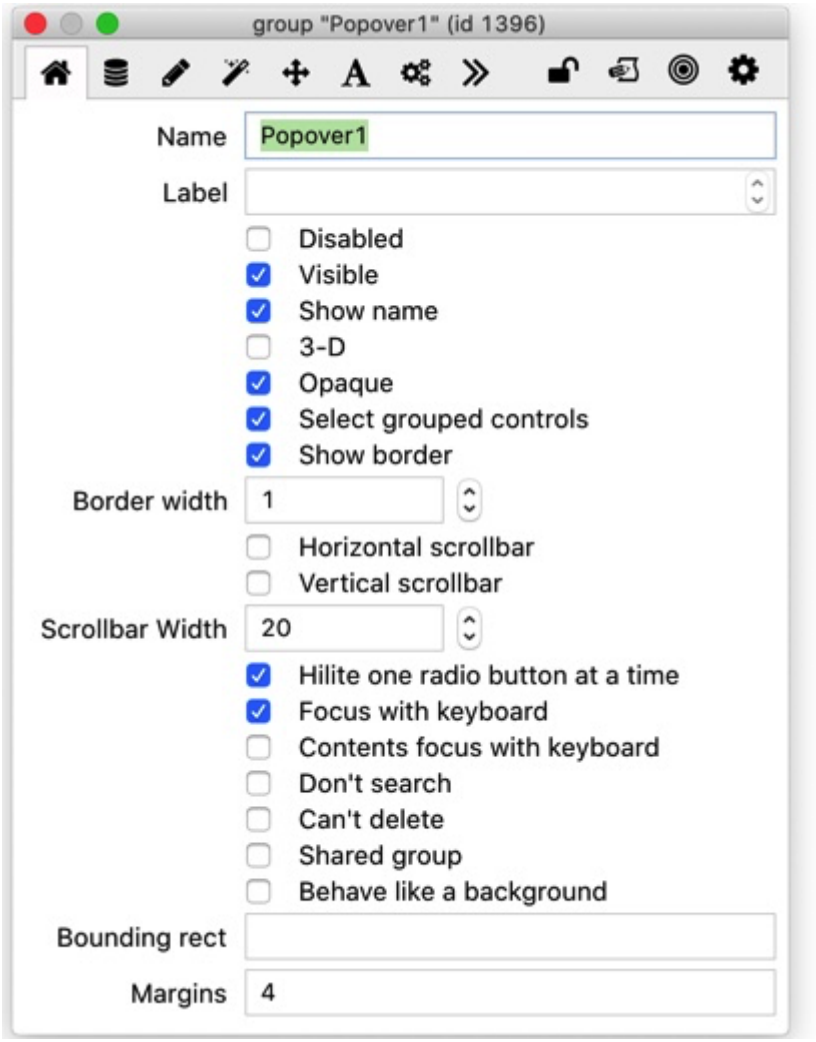
```
on openPopover pPopoverName
   -- open the popover at the position specified by the showLoc rect object
   local tShowLocControlName,tShowLoc
   put pPopoverName & "_showLoc" into tShowLocControlName
   put the location of control tShowLocControlName into tShowLoc
   set the location of group pPopoverName to tShowLoc
   show group pPopoverName
end openPopover

on closePopover pPopoverName
   -- hide the popover and move it back to the position specified by the hideLoc rect object
   local tHideLocControlName,tHideLoc
   put pPopoverName & "_hideLoc" into tHideLocControlName
   put the location of control tHideLocControlName into tHideLoc
   if the environment is not "development" then hide group pPopoverName
   set the location of group pPopoverName to tHideLoc
end closePopover
```

There is a corresponding red hideLoc rectangle over to the right side of the card. This object determines the location where the popover group gets placed after it is closed.

The code shown above is used throughout the application to show/hide popovers.

Note: (4) Notice that the popover group is moved but not hidden during development in the IDE as a convenience. Of course this can be easily changed by removing the "if the environment is not development" part of the instruction.

Renaming & Setting Popover Properties

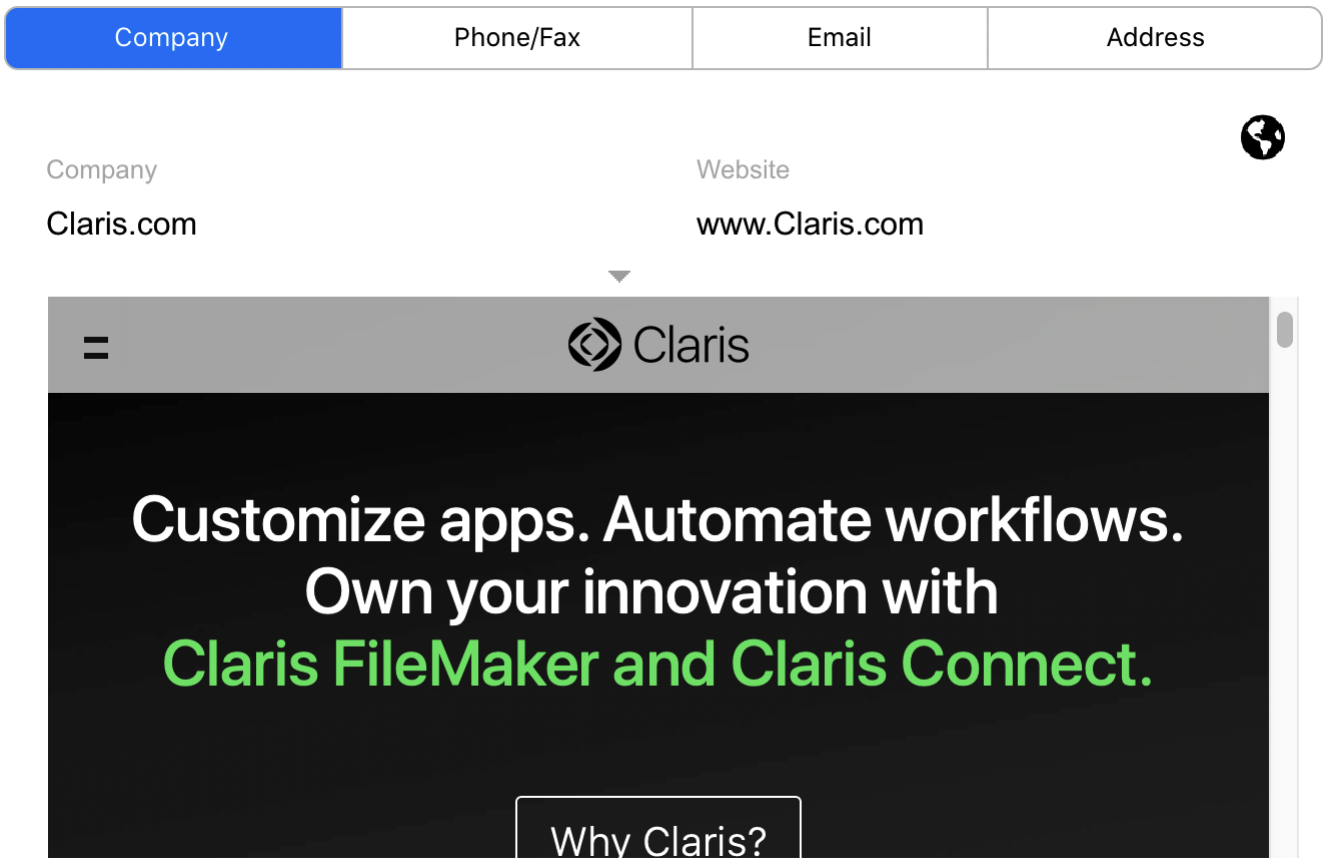


When the popover is visible in the LiveCode IDE, its properties can be changed including the name, show name property, opaque background properties etc.

To edit a popover group, click into the card with the pointer tool (editing mode), this will show the outline of the Layout_Objects_Group which contains all of the objects of the card. Click the Edit Group button in the LiveCode top toolbar. Then click the popover group and click Edit Group again. To reduce the number of steps required to edit a group, you can enable the Select Grouped option on the LiveCode toolbar.

WebViewer Implementation Details

Webviewer Implemented as Browser Widget - Company Tab



The webviewer is implemented as a LiveCode browser widget, the URL is populated via code in the refreshFields card handler using the following code:

```
    -- ====================== Browser URL ======================
    set the url of widget "Browser1" to
setBrowserURLPrefix(gDataCRArray[tCurrentBaseTableName][1]["website"])
```

The setBrowserURLPrefix() function adds "https://" before the URL unless there is already a prefix starting with http/https/file already in the URL.

As the user navigates between records, the URL and browser contents change automatically based upon the data stored in the database.

Note: On macOS, notice that the browser and other objects on the Company tab of the tab control are hidden when clicking another tab - this is by design, and matches the original functionality.

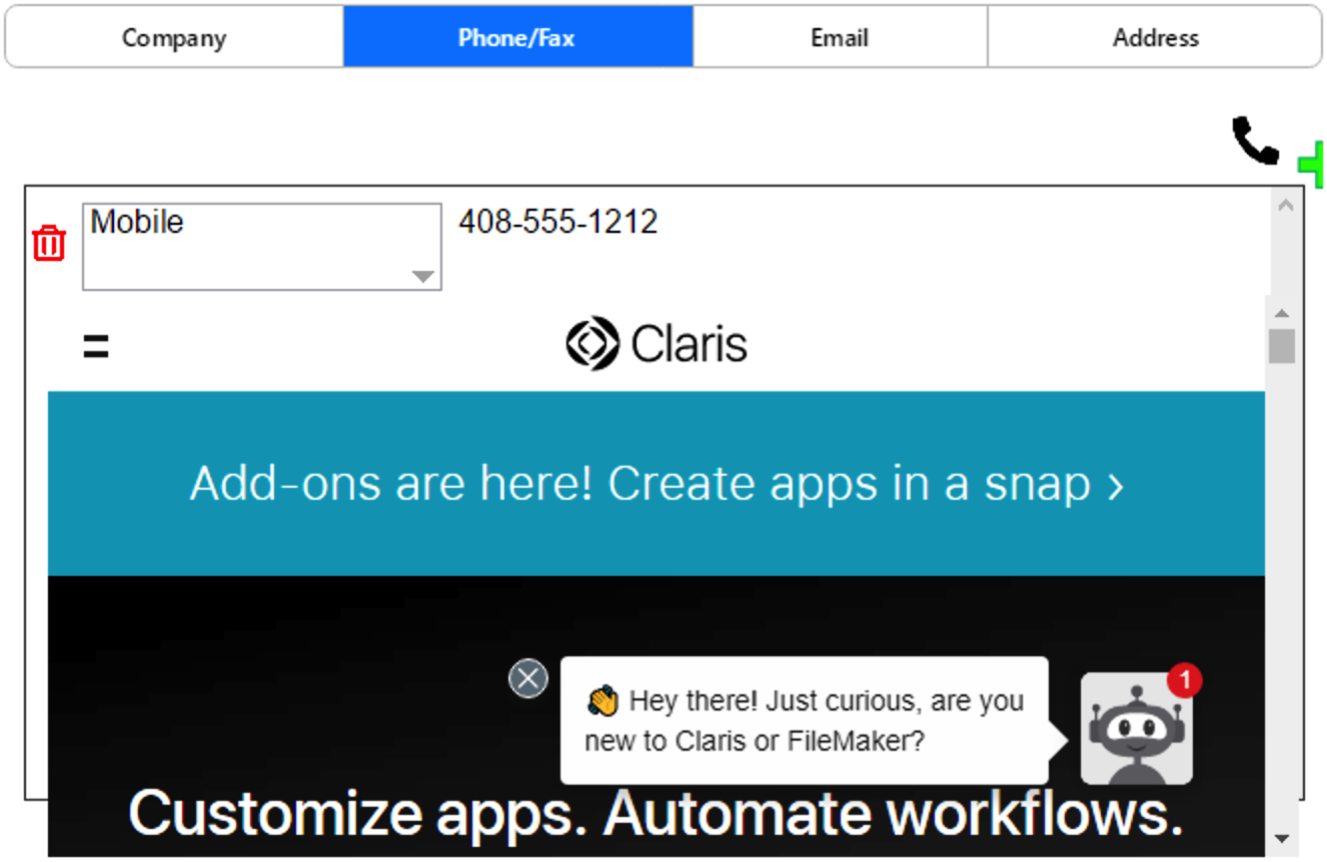
FileMaker WebViewer Change

```
If ( IsEmpty ( Contacts::Website );"";"https://" & Contacts::Website)
```

Inside the original FileMaker webviewer, the calculation shown above was replaced with a simple reference to the field containing the URL. The checking of the URL prefix was implemented using the setBrowserURLPrefix() LiveCode function.

The unmodified version of the layout is shown in the card named 4Contact Details. FmPro Migrator has commented out the conditional, leaving it for manual development by the developer after detecting the "(" character.

Windows Browser Widget Differences

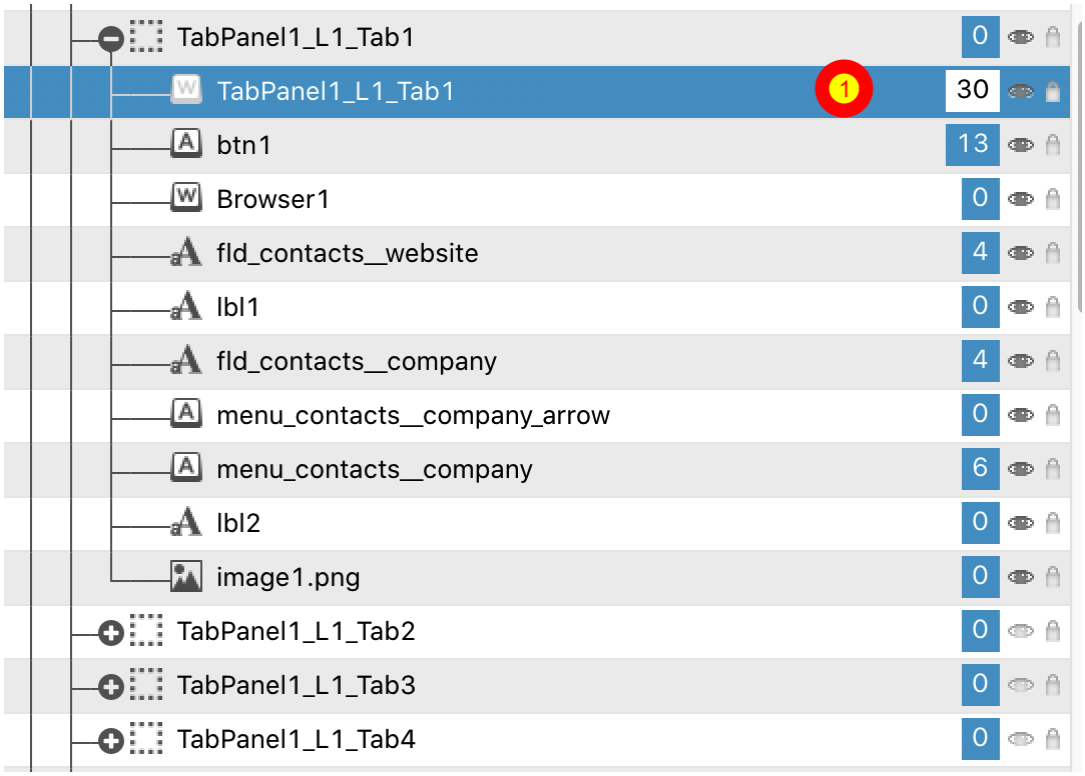


On Windows, LiveCode uses the CEF (Chromium Embedded Framework) to implement the browser widget.
However, the widget gives the appearance of functioning like a palette window which floats over everything else. [At the present time, this is a known problem on Windows. But an easy workaround is shown below.] This means that when the different tabs are clicked, the browser covers the DataGrid rows which should be displayed.

FmPro Migrator implements tab controls as separate groups, so clicking a tab shows the group which was clicked and hides the other groups.

The following code enables a simple workaround for this difference in functionality between platforms.

Hiding the Browser Widget - Segmented Control



Each tab control tab is implemented as a group of objects in a group with a segmented control widget representing the tab.

Each segmented control contains code which shows/hides the other groups as needed when the user clicks a segment. In this example, there are 30 lines of code performing these tasks.

Hide Widget Code

```
on hiliteChanged
   local pItem
   put the hilitedItemNames of me into pItem
   switch pItem
      case "company"
         -- do nothing
         break
      case "phone_fax"
         hide group "TabPanel1_L1_Tab1"
         show group "TabPanel1_L1_Tab2"
         lock messages
         set the hilitedItems of me to 1
         hide widget "Browser1"
         unlock messages
         break
      case "email"
         hide group "TabPanel1_L1_Tab1"
         show group "TabPanel1_L1_Tab3"
         lock messages
         set the hilitedItems of me to 1
         hide widget "Browser1"
         unlock messages
         break
```

Adding the code:

```
hide widget "Browser1"
```

to each of the other tabs will hide the browser widget when the segment is clicked.

Show Widget Code

```
on hiliteChanged
   local pItem
   put the hilitedItemNames of me into pItem
   switch pItem
      case "company"
         hide group "TabPanel1_L1_Tab2"
         show group "TabPanel1_L1_Tab1"
         lock messages
         set the hilitedItems of me to 2
         show widget "Browser1"
         break
      case "phone_fax"
         -- do nothing
         break
      case "email"
         hide group "TabPanel1_L1_Tab2"
         show group "TabPanel1_L1_Tab3"
         lock messages
         set the hilitedItems of me to 2
         hide widget "Browser1"
         unlock messages
         break
```

Adding the code:

show widget "Browser1"

to the "company" tab for each of the other segmented controls, and adding

hide widget "Browser1"

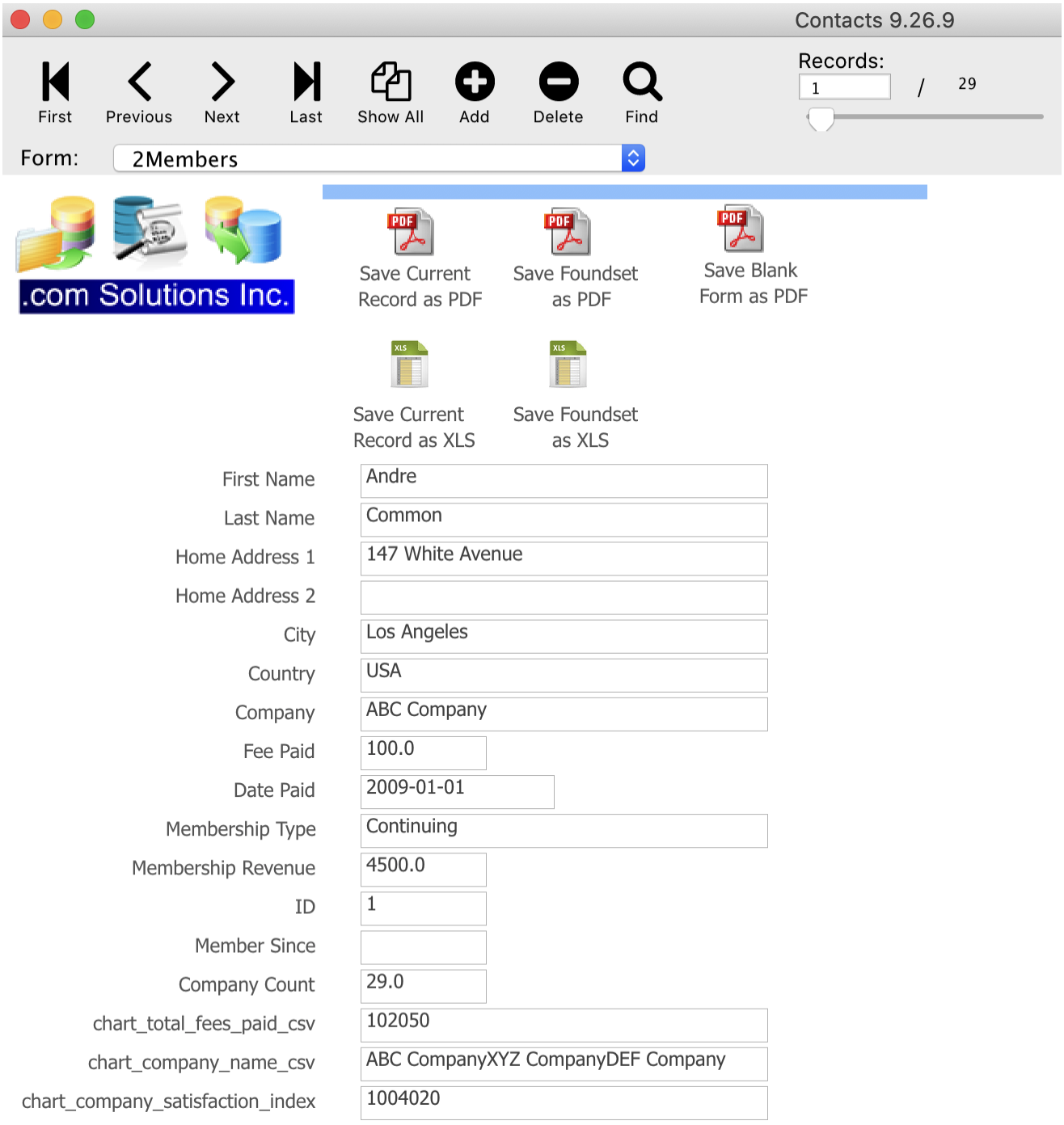
to each of the other tabs will hide the browser widget when the segment is clicked.

This way the browser widget will be displayed when it is needed, and hidden when it is not needed.

Note: All of the changes shown in these screenshots has been implemented within the included stack: Contacts <Version>_browser_changes.livecode

Members Layout Features

Members Layout - Buttons & Description

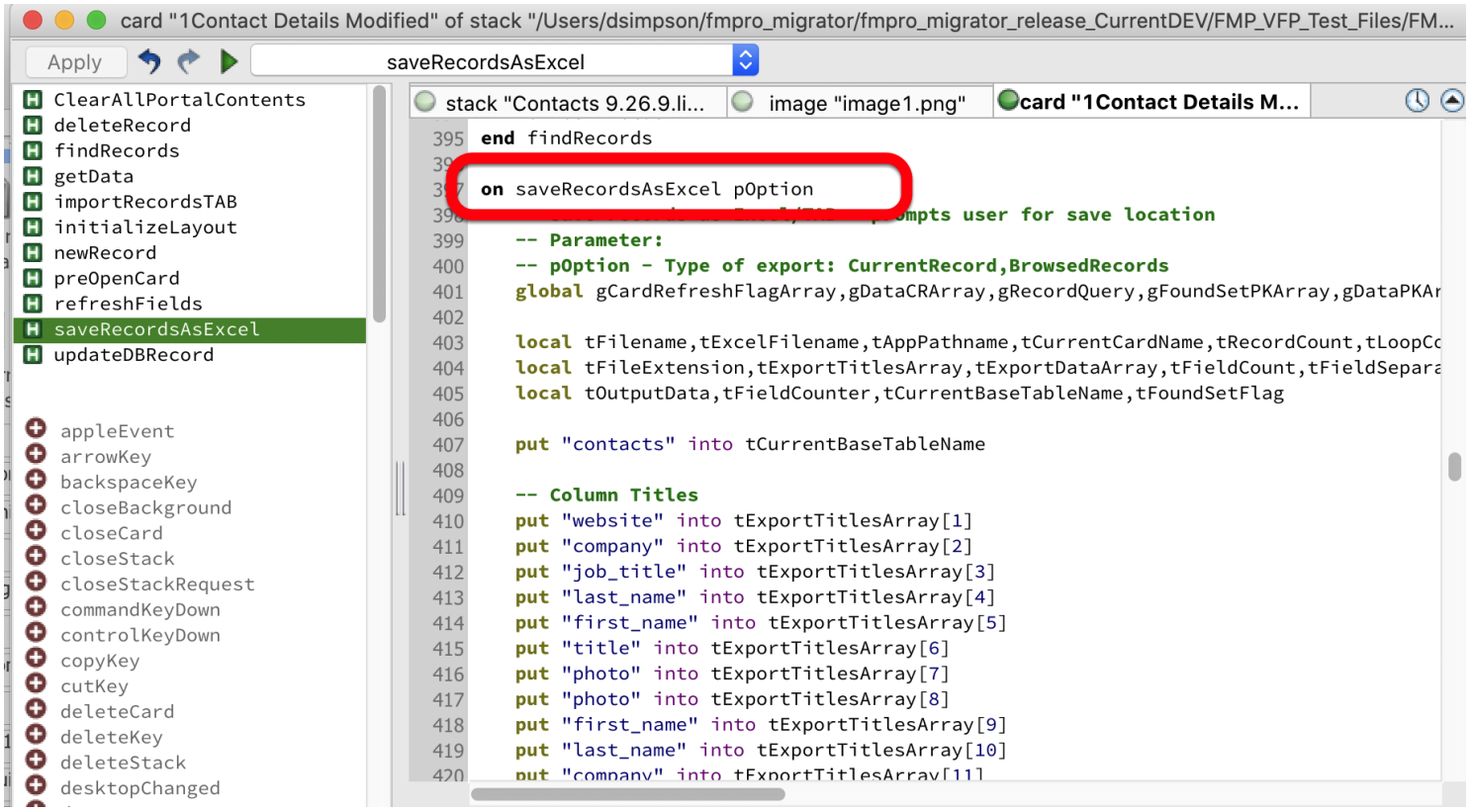


The Members layout includes a sample of buttons across the top which were assigned to single step script steps in FileMaker.

These buttons include:
Save Current Record as PDF

Save Current Record as XLS
Save FoundSet as PDF
Save FoundSet as XLS
Save Blank Record as PDF

Save Current Record Buttons

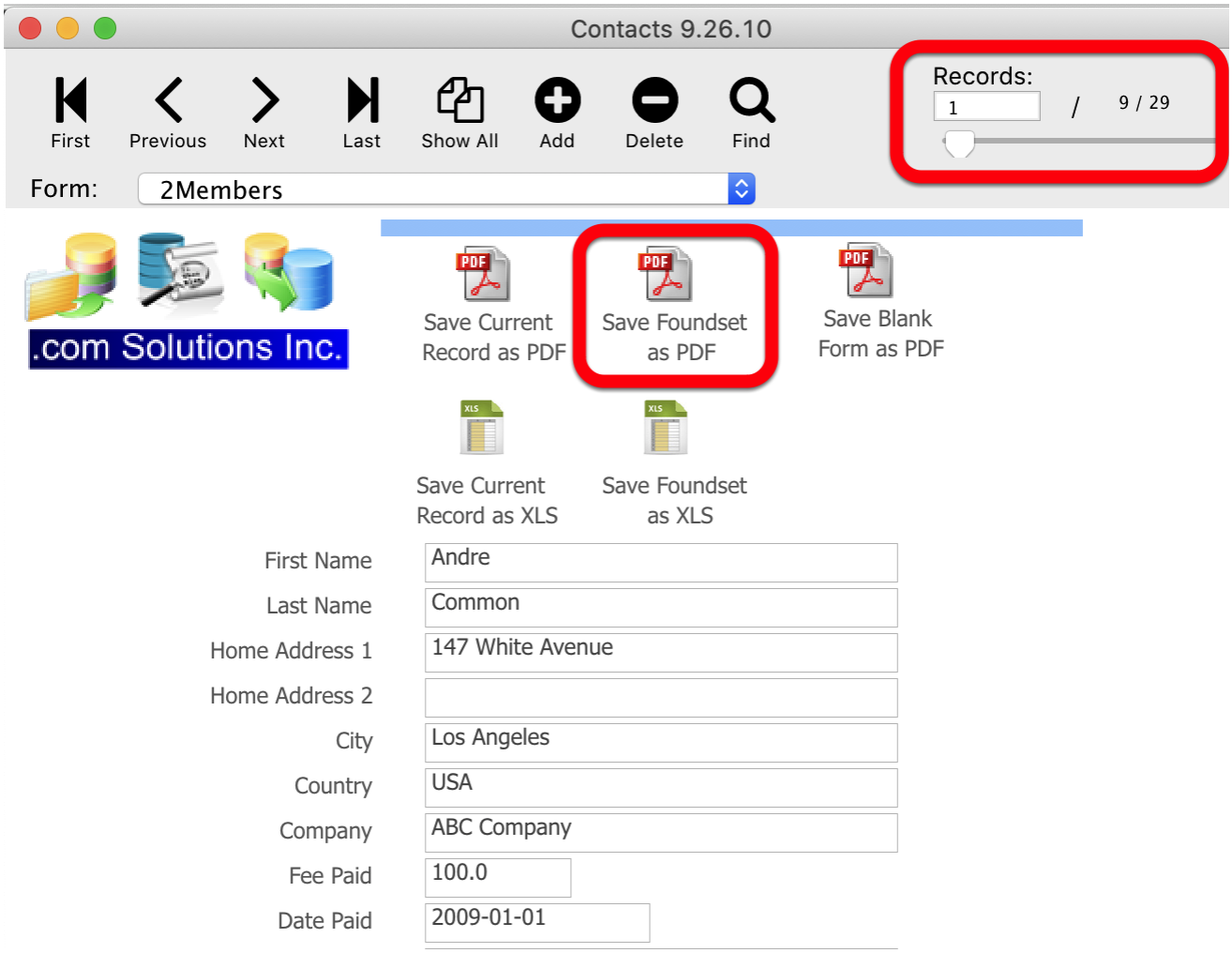


For each of these buttons, a PDF or tab delimited file with an .XLS file extension will be created and saved to disk. The file will then be opened automatically with the appropriate application.

The saveRecordsAsExcel feature uses a card level handler with this name, which has been generated with the specific fields which exist on this card of the stack.

The saveRecordsAsPDF feature uses a stack level handler used throughout the application which doesn't require changes based upon the names of the fields on the card.

Save Foundset Buttons



Clicking the Save Foundset as PDF will create a PDF of the foundset of records, if there is a foundset (as shown in this screenshot 9 of 29 records). If a foundset doesn't exist, then the PDF will be created with all of the records in the table.

The Save Foundset as XLS works the same way, exporting a foundset if it exists or exporting all of the records as a TAB delimited file having the XLS file extension.

In each situation, the file is then opened after saving. Of course you can easily change this behavior within the handlers by commenting out the following command:

```
launch url "file:" & tPDFFilename
```

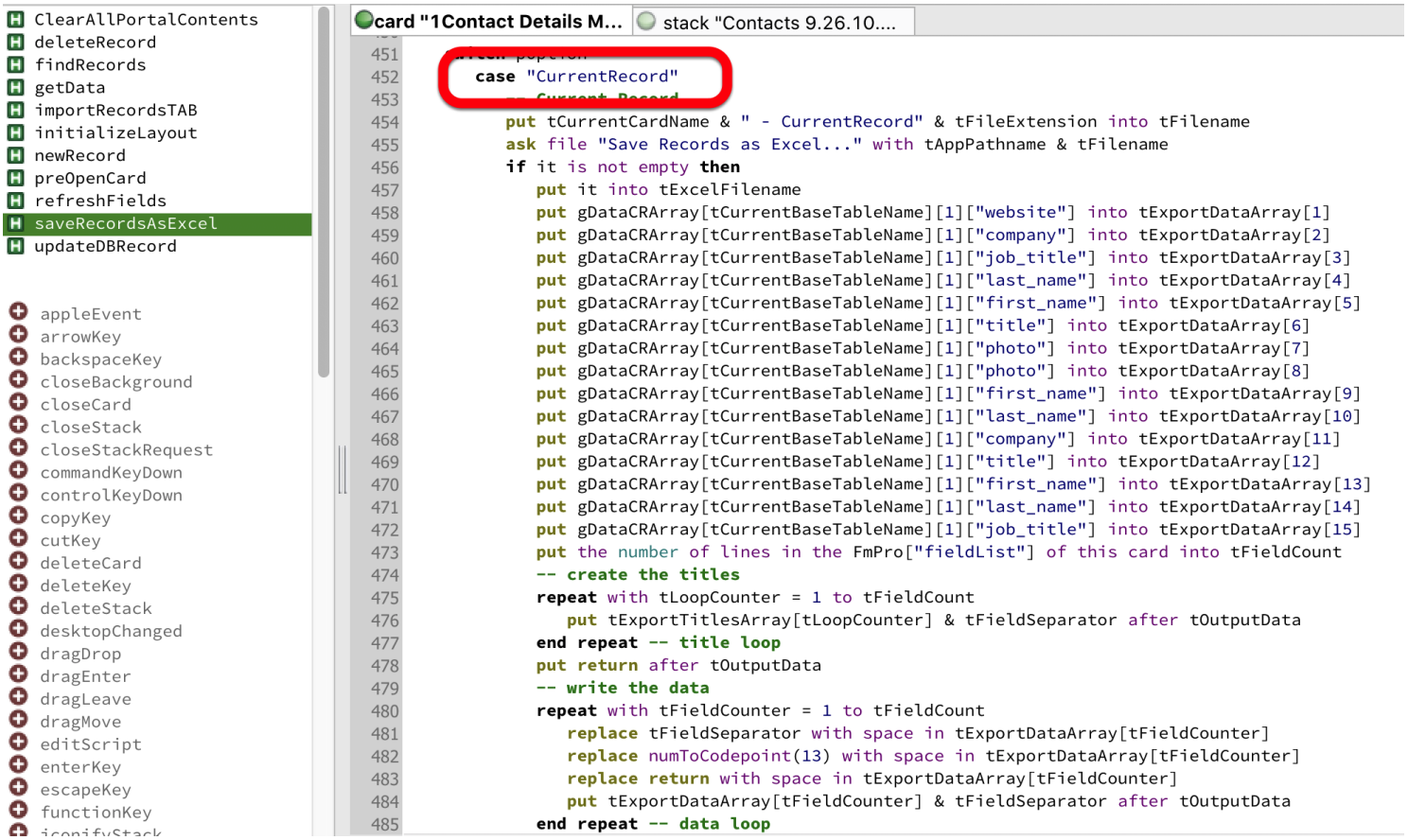
Save Current Record as PDF Code

stack "Contacts 9.26.10....

```
994
995         case "CurrentRecord"
996             put tCurrentCardName & " - CurrentRecord.pdf" into tFilename
997             ask file "Save Records as PDF..." with tAppPathname & tFilename
998             if it is not empty then
999                 put it into tPDFFilename
1000                lock screen
1001                printSetup
1002                put the width of group "Layout_Objects_Group" into tLayoutObjectsGrouupWidth
1003                put the height of group "Layout_Objects_Group" into tLayoutObjectsGrouupHeight
1004                open printing to pdf tPDFFilename
1005                print card tCurrentCardName from the left of group "Layout_Objects_Group",the top of group "Layout_Obj
1006                close printing
1007                printRestore tLayoutObjectsGrouupWidth,tLayoutObjectsGrouupHeight
1008                unlock screen
1009                launch url "file:" & tPDFFilename
1010                resizeStack -- restore display of scrollbars
1011            end if -- check for cancelled save dialog
1012            break
1013        case "RecordsBeingBrowsed"
1014            put tAppPathname & tCurrentCardName & " - Foundset.pdf" into tFilename
```

The Save Current Record as PDF code is in the "CurrentRecord" CASE block of the switch statement within the saveRecordsAsPDF stack level handler. It prompts the user to specify a filename and path while providing a default name which can be changed by the user. After saving the width and height of the Layout_Objects_Group, it opens printing to PDF, prints the card, closes printing then restores the size of the Layout_Objects_Group. Once the PDF has been saved it is opened automatically.

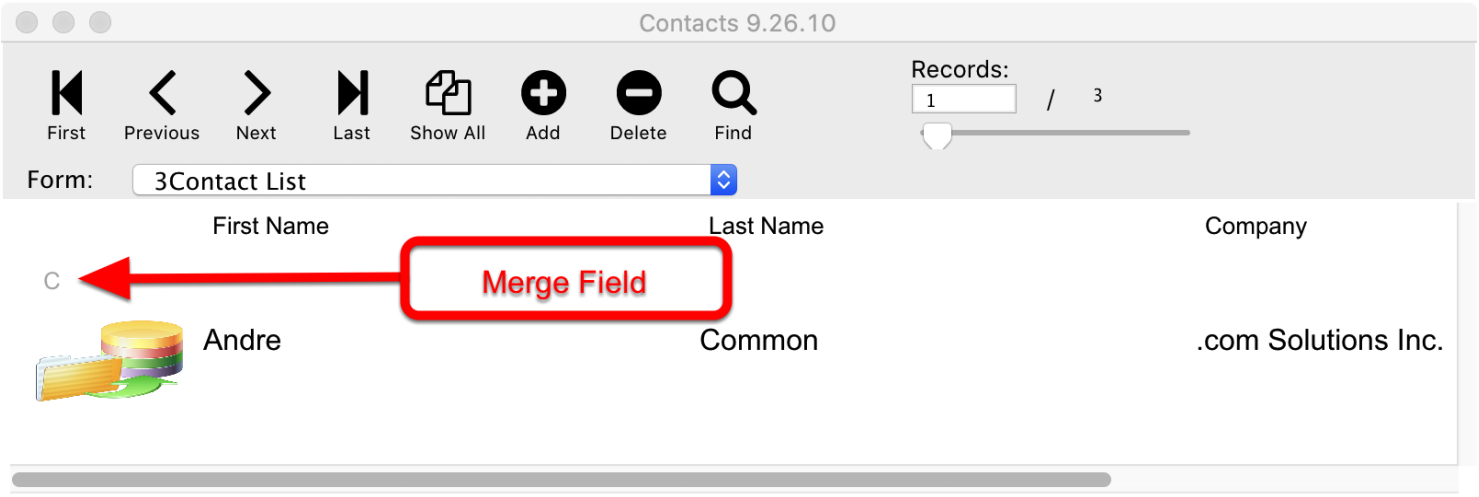
Save Current Record as XLS Code



Exporting XLS data is performed by the saveRecordsAsExcel handler located in the script for each converted layout. The "CurrentRecord" CASE block loops thru the fields to gather data for the tExportDataArray. Then if fills the tExportTitlesArray, outputs the titles to the tOutputData variable and replaces embedded return characters and TAB characters in the tExportDataArray contents as it outputs the data to the tOutputData variable.
Once the tOutputData variable has been filled with the titles and the field data, it saves the data to the output file specified by the user.

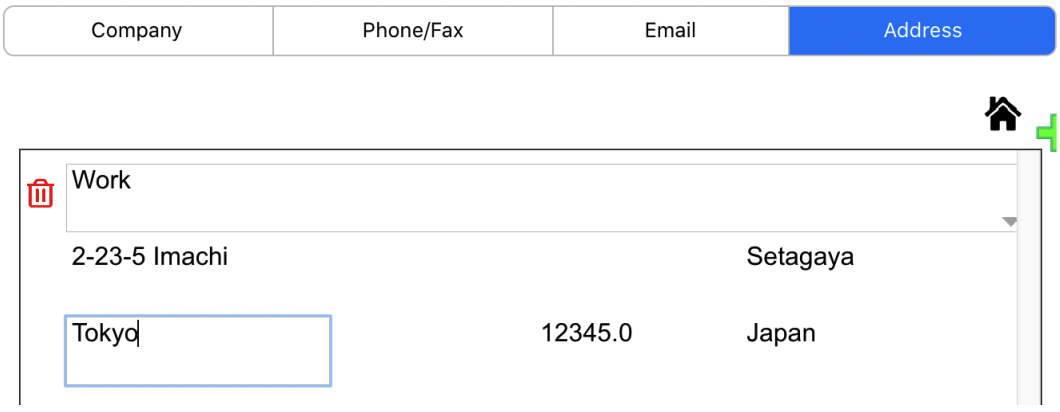
Contact List Layout Notes

Contact List Layout



As a converted list view layout, the fields just display one record of data, until the record navigation buttons are clicked in the status area above the card.
The merge field <<INITIAL>> above the image field has been converted and does show data correctly when navigating between records. However the stored calculation defined for this field unfortunately wasn't converted, but would be straightforward to implement within the Stored_Calculation_Library.
The reason the data is being displayed is that it was copied from the FileMaker database where the calculation was performed. New records wouldn't have a value in that field unless a stored calc was added manually.

List implementation Idea #1 - For Display



A LiveCode DataGrid can display data in a scrolling list view for display purposes (not for printing). The DataGrid shown in this screenshot is from the 1Contact Details - Modified converted layout.

Converting a FileMaker portal into a fully functional DataGrid would require adding a relationship to the database.

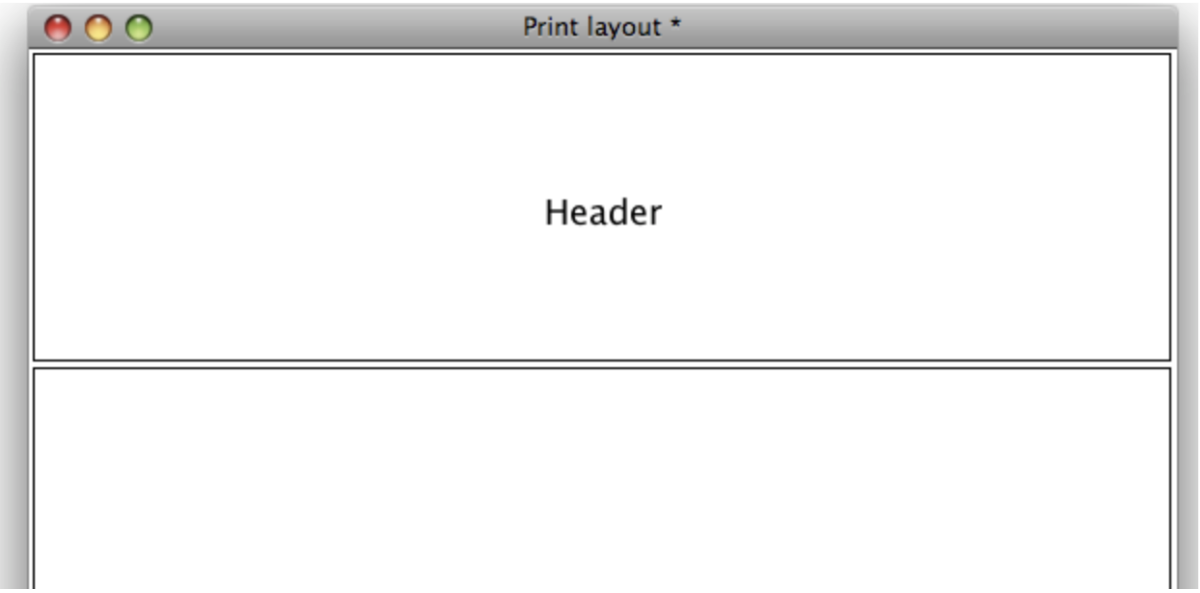
However if no relationship was available, it would be possible to manually add and edit a DataGrid to a card. This would involve writing code to update the grid with records, and existing examples in the converted Contacts stack provide examples of this functionality.

List implementation Idea #2 - For Printing

Printing a complex layout

To print a more complicated layout, create a stack and set its rectangle to the current **printRectangle**. Add rectangular areas for each component you will be printing. Then set *Geometry* properties (see the section on the *Geometry Manager*, in the **LiveCode Script** guide for more information) on each of these rectangles so they resize correctly when the stack is scaled. Set up your print routine so that you open this stack invisibly then resize it to the **printRectangle**. This will trigger the geometry routines and scale the rectangular areas correctly. Then run your sequence of print commands to print into each rectangle.

In the figure below, we have set the size of the stack to the **printRectangle** then added 4 rectangle graphics. We have named each graphic and turned on the *Show Name* property for each so you can see the name.



If a list view is required for printing purposes, some additional work would be required. By default, LiveCode doesn't include a list type view like FileMaker.

Some ideas can be gained from the LiveCode printing lesson shown in the screenshot above:
https://livecode.com/docs/9-5-0/core-concepts/printing-in-livecode/

The section of the lesson regarding "Printing a Complex Layout" includes a discussion of how to print Header/Footer/Body section information to a PDF file or a printer.