.com Solutions Inc.

LiveCode Conversion Procedure

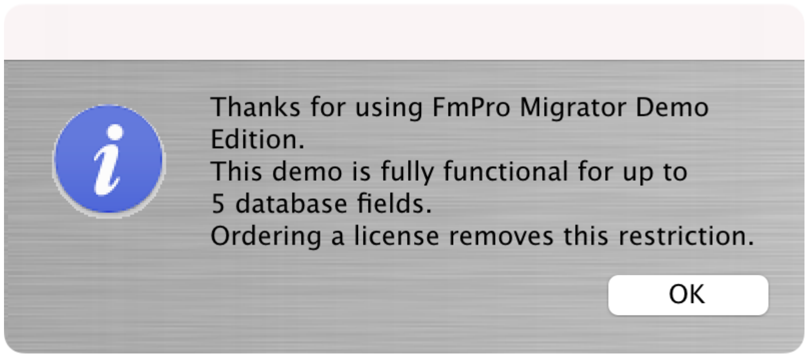
LiveCode Conversion Procedure

Licensing FmPro Migrator

As of FmPro Migrator v11, all FmPro Migrator downloads are fully functional with features unlocked via a single license key.
This section of the manual shows how to enter the license key to unlock the features within FmPro Migrator.

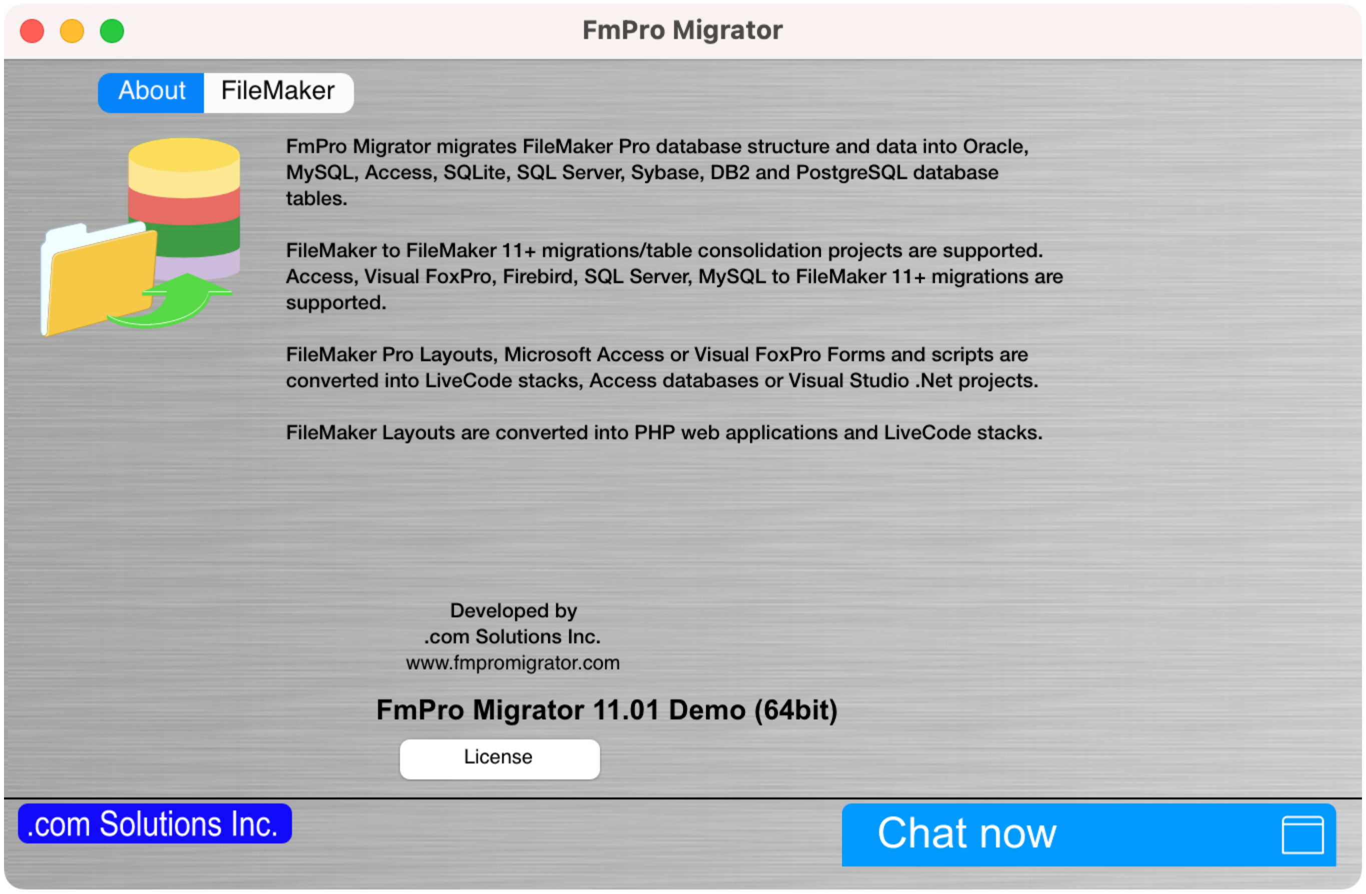
FmPro Migrator 11.01
4/14/2024

Demo Edition Dialog


Thanks for using FmPro Migrator Demo Edition.
This demo is fully functional for up to 5 database fields.
Ordering a license removes this restriction.

When launched the first time, FmPro Migrator will be running in Demo mode as shown in this screenshot. Clicking the Ok button opens the order page of the website.
In addition to transferring data for 5 database fields, the conversion features shown on the GUI tab of the Migration Process window will convert 5 layouts or forms/reports & scripts.

Demo Mode - About Tab



FmPro Migrator migrates FileMaker Pro database structure and data into Oracle, MySQL, Access, SQLite, SQL Server, Sybase, DB2 and PostgreSQL database tables.

FileMaker to FileMaker 11+ migrations/table consolidation projects are supported. Access, Visual FoxPro, Firebird, SQL Server, MySQL to FileMaker 11+ migrations are supported.

FileMaker Pro Layouts, Microsoft Access or Visual FoxPro Forms and scripts are converted into LiveCode stacks, Access databases or Visual Studio .Net projects.

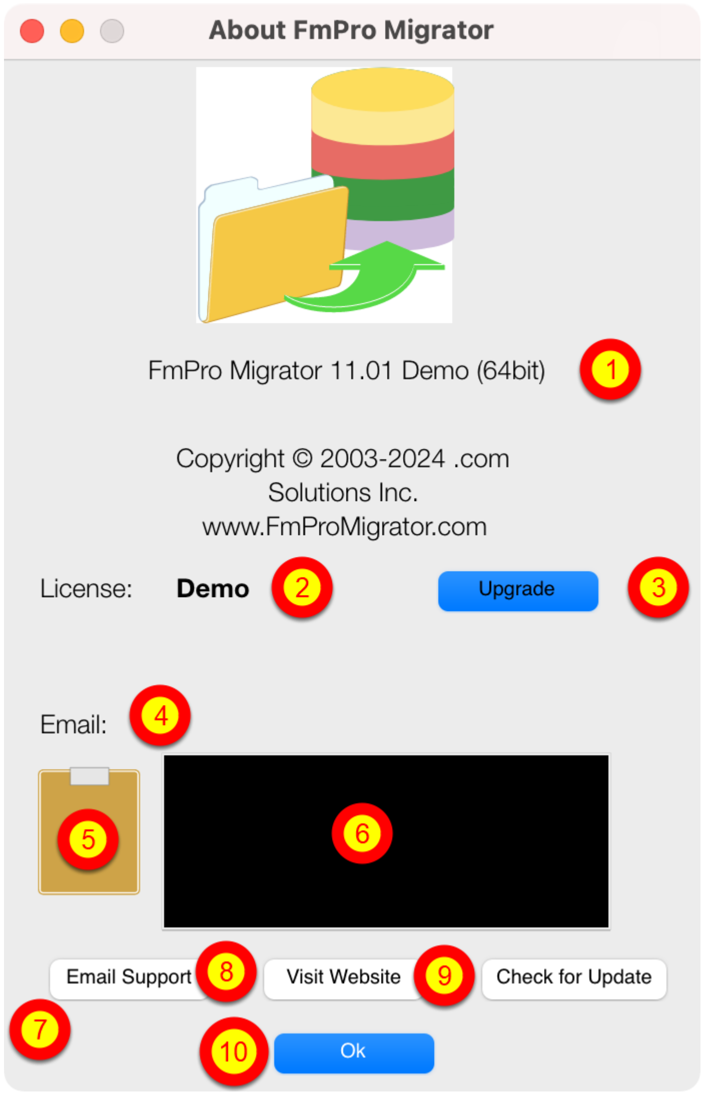
FileMaker Layouts are converted into PHP web applications and LiveCode stacks.

Developed by
.com Solutions Inc.
www.fmpromigrator.com

FmPro Migrator 11.01 Demo (64bit)

Clicking the "License" button on the About tab, or selecting the Help -> License/About menu items will open the About window where you can enter the license key.

About/License Window


About FmPro Migrator

FmPro Migrator 11.01 Demo (64bit)

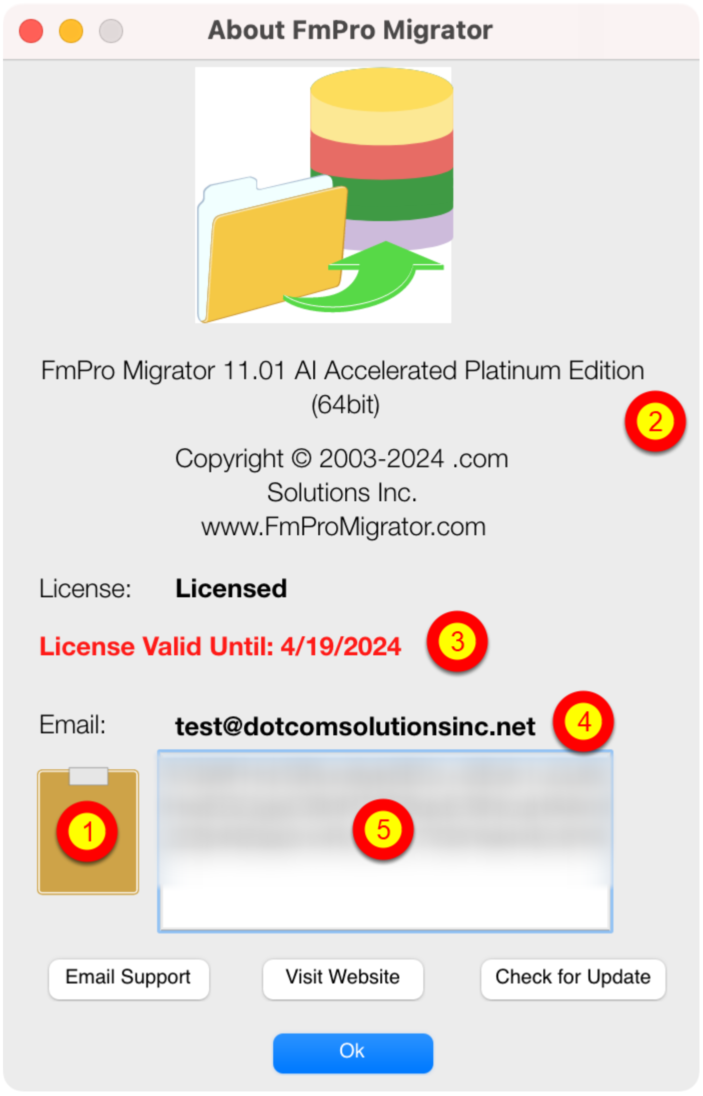
Copyright © 2003-2024 .com
Solutions Inc.
www.FmProMigrator.com

License: **Demo**

Email:

On the About FmPro Migrator window:
(1) Product name and version licensed, (2) Demo or Licensed will be displayed, (3) The Upgrade button opens the website order page, (4) Email address associated with your license key, (5) Clipboard button reads the license key from the clipboard, (6) License key field, (7) Email Support button opens the Contact form on the website, (8) Visit Website opens the order page in Demo mode or the product page in Licensed mode, (9) Check for Update opens the downloads page with the latest software version, (10) Ok button closes the About window.

Pasting License Key



FmPro Migrator 11.01 AI Accelerated Platinum Edition (64bit)

Copyright © 2003-2024 .com Solutions Inc.
www.FmProMigrator.com

License: **Licensed**

License Valid Until: 4/19/2024

Email: **test@dotcomsolutionsinc.net**

Your product license key will be displayed in your web browser when your order has been completed. It will also will be sent via email at the same time, please check your SPAM folder if the email doesn't arrive in a few minutes.

(1) Copy the license key to the clipboard and click the clipboard icon. Once validated (2) the product name will change, (3) the License validation date will be updated, (4) your email address will be displayed, and (5) the license key will be displayed in the field under the Email address when clicking inside the field. In this screenshot the license key has been obscured.

And that is all you need to do to license all of the features of FmPro Migrator.

This document explains the process of converting FileMaker Pro®, Microsoft Access and Visual FoxPro databases into LiveCode (www.LiveCode.com) stacks with FmPro Migrator Platinum Edition. FmPro Migrator Platinum Edition converts each layout/form/report into a LiveCode card, containing all of the fields, portals, text, images and buttons from the original layout.
FmPro Migrator Platinum Edition generates a functional database application within a few seconds, including hundreds of lines of LiveCode code within each card of the generated stack file. Individual objects including Tab Panels, radio button groups, checkbox button groups and Data Grid objects also include the LiveCode code required for an easy to use database front-end application.

This feature leverages the automated layout and relationship importing features of FmPro Migrator Platinum Edition, while using FileMaker Pro Advanced, the LiveCode IDE along with the SQL database you choose to use to store the data for your application.

LiveCode stack files generated by FmPro Migrator Platinum Edition incorporate the following commonly used database application features:

Display Records - When the stack is opened and a connection is made to the converted SQL database, the records associated with the converted Layout/Form are displayed in a form viewing mode. The fields, text labels, embedded graphics and images are displayed in the same manner as the original database file. Record navigation is implemented using 4 controls:
Next Button, Previous Button, Scrollbar and Record Number field entry. SQL database BLOB column fields containing images are automatically displayed.
Related records are automatically displayed within individual fields and portals/subforms are converted into LiveCode Data Grids. The related records displayed within Data Grids are automatically updated when advancing to another record in the parent table.

Update Records - Clicking into any field provides the user with the ability to update the contents of the field. Date fields are configured with a Date Picker control. Custom Value Lists from the original database are displayed as drop down menus, pop-up menus, radio button group or checkbox group depending upon the original field definition.
Related records displayed within Data Grids can be updated just as easily as records in the main form.

Insert Records - Individual records can be inserted as a new record in either Browse or Find operating modes. Clicking the Insert Portal Row button to the top right of any Data Grid object will add a new record to the related database table. As with the original database implementation, the primary key for the parent table will be automatically inserted into the foreign key column of the related record.

Delete Records - Individual records can be deleted in either Browse or Find modes. If a Delete

Row button or image object was embedded in the original FileMaker Portal, then this same functionality will also be implemented to delete the selected Data Grid row.

Query Records - Query by example record searching is automatically implemented for each card in the stack file. Once a set of records has been found as a result of the query, the navigation buttons can be used to navigate thru the found set of records. Individual records can be inserted or deleted in found set mode as well as within the standard Browse mode.

This document includes information for the built-in library sub stacks:
fmFunctions_Library - This library duplicates the functionality of 129 of the most commonly used FileMaker built-in functions. Atool is provided to remap the original FileMaker functions to these LiveCode versions within the converted FileMaker scripts.

Globals_Library - The gGlobals array within the _gGlobalsInitialize handler includes FM tables/fields which are used as global variables. These globals are initialized during app startup and can be updated anytime by the application, just like FileMaker globals.

Images_Library - This library provides a place where commonly used images can be centrally stored within the new application stack.

SQL_Library - The SQL_Library provides database independent code to connect to your SQL database, insert, query, update, delete records, and load relationships from a JSON file.

Stored_Calc_Library - Stored field calculations are defined in this library for each table of the database from the original FileMaker database definition. Code which is too complex for automatic conversion is commented to prevent errors. This code runs during record update/insert database tasks.

Unstored_Calc_Library - Stored field calculations are defined in this library for each table of the database from the original FileMaker database definition. Code which is too complex for automatic conversion is commented to prevent errors. This code runs as records are read from the database by the refreshFields handler on each card. By default, it is commented to improve performance, because these calcs do not necessarily need to run for every converted layout. The developer needs to decide whether it should run, then enable this code as needed.

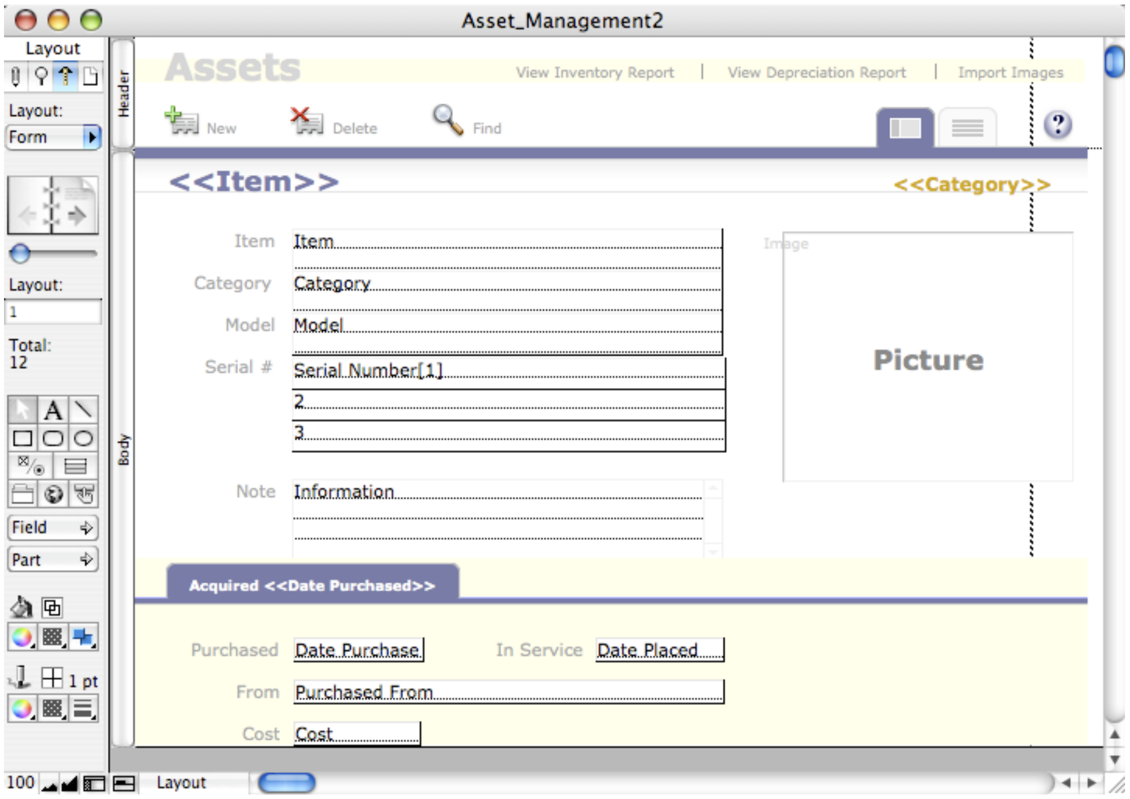
Validation_Library - Field validation code during database insert/update tasks is defined for each database table inside this library.

Value_List_Library - Custom and field based value lists are defined within this library. Changes made here are reflected throughout the entire application. Value lists from Microsoft Access and Visual FoxPro are also defined inside this library.

Note: The LiveCode Community icon is displayed in FmPro Migrator for the LiveCode conversion feature. This feature is compatible with LiveCode Community and LiveCode Commercial, even though the LiveCode Community Edition is no longer being updated. Pricing for the LiveCode Commercial versions has been made significantly more affordable. For details, please see the LiveCode.com website.

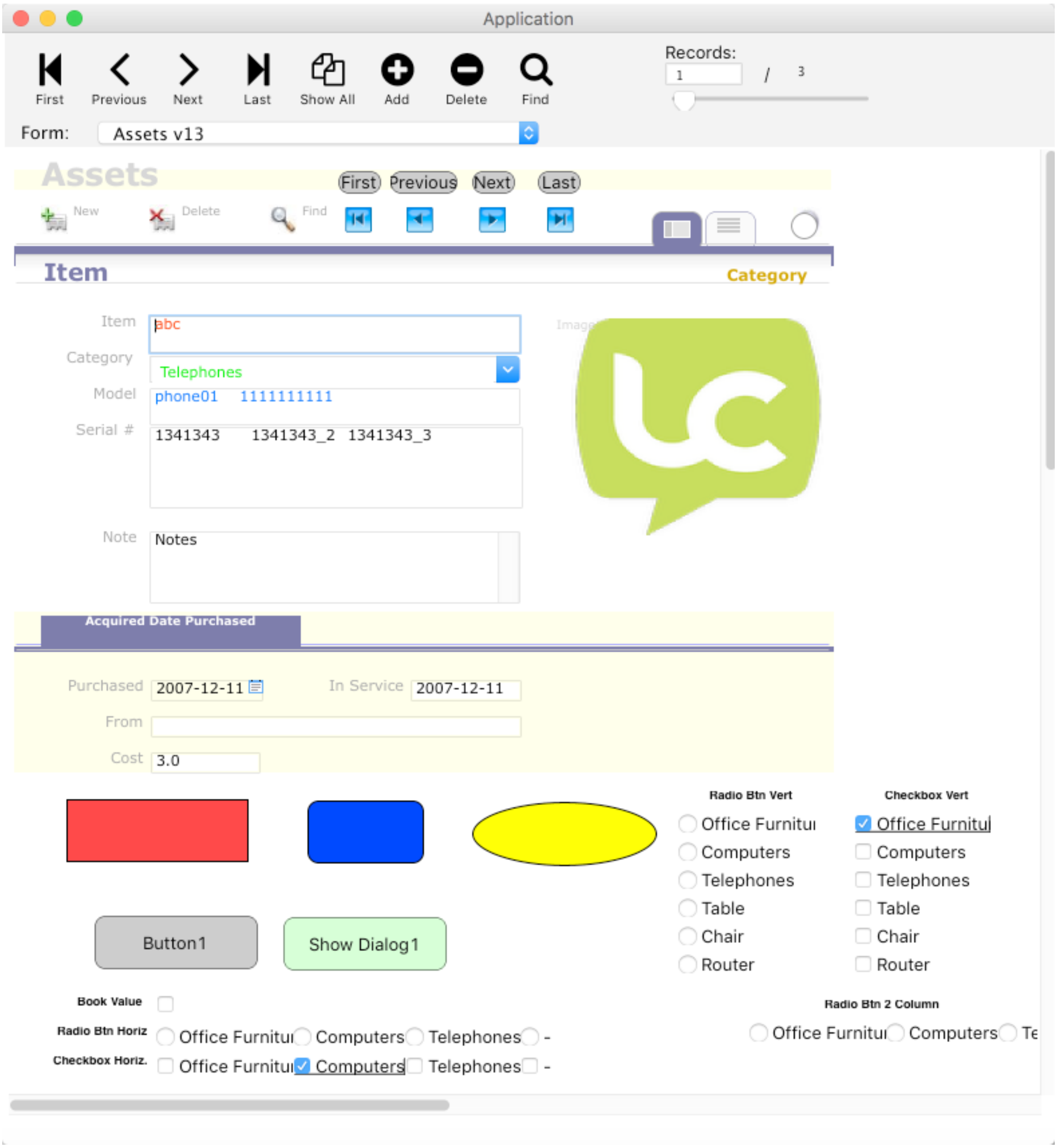
Document Version 12
FmPro Migrator11.01
4/15/2024
[Added new Licensing FmPro Migrator info, updated some of the screenshots.]

Original FileMaker Pro Layout



FmPro Migrator reads the XML definition of a FileMaker layout, and then converts this information into a LiveCode card containing all of the fields, portals, text, images and buttons from the original layout. An original layout for the sample Asset Management database is shown in the previous image.
Note: Microsoft Access and Visual FoxPro projects are also converted into the XML definition of a FileMaker layout, when the project metadata is imported into FmPro Migrator.

Converted LiveCode Card



Records: 1 / 3

Form: Assets v13

Item: abc

Category: Telephones

Model: phone01 1111111111

Serial #: 1341343 1341343_2 1341343_3

Note: Notes

Purchased: 2007-12-11

In Service: 2007-12-11

From:

Cost: 3.0

Radio Btn Vert: Office Furnitu, Computers, Telephones, Table, Chair, Router

Checkbox Vert: [✓ Office Furnitu], Computers, Telephones, Table, Chair, Router

Radio Btn Horiz: Office Furnitu, Computers, Telephones, -

Checkbox Horiz.: Office Furnitu, [✓ Computers], Telephones, -

Radio Btn 2 Column: Office Furnitu, Computers, Te

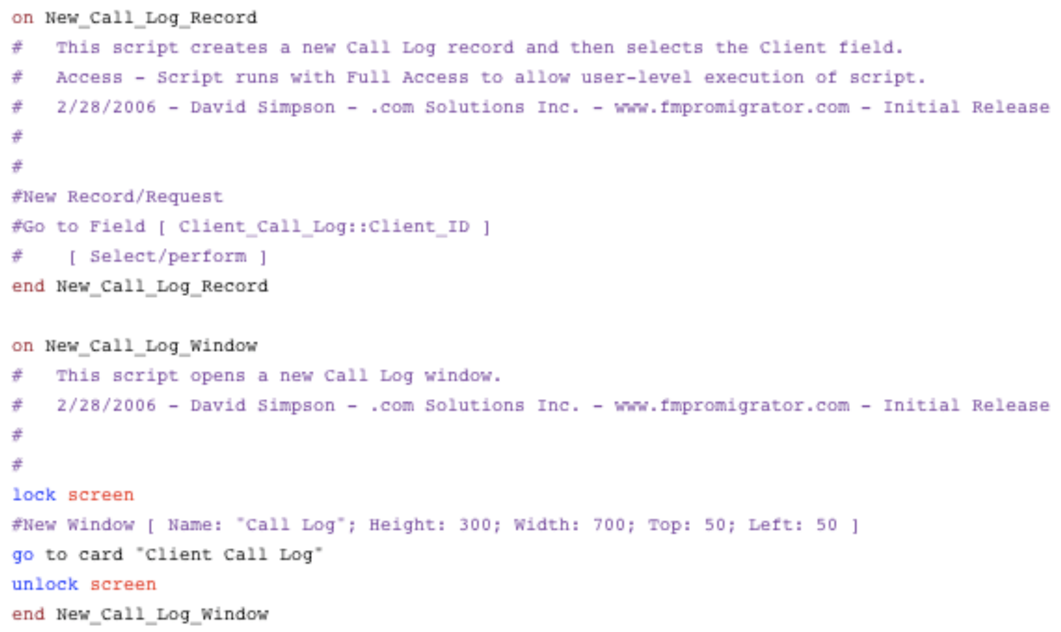
The resulting LiveCode card is shown in this screenshot. Each field is filled with the name of the original Tablename::Fieldname information from the original FileMaker Pro Layout.

A Status Area background object is provided at the left hand side of the card, to simulate the Browse/Find modes from the original database. Menus and scripts are provided in the template stack file for hiding/showing the status area and moving the card objects to accommodate the Status Area.

Each original layout name is used as the name for the LiveCode card, and when the Layout drop down menu is selected, a "go to card" instruction is executed, taking the user to the selected card.

The card objects are placed into a Geometry Manager enabled group named "Layout_Objects_Group". As the card is resized, the Layout_Objects_Group is also resized. If the window is too small to show all of the objects, scrollbars automatically appear to enable scrolling by the user.

Converted FileMaker Script Code



```
on New_Call_Log_Record
#   This script creates a new Call Log record and then selects the Client field.
#   Access - Script runs with Full Access to allow user-level execution of script.
#   2/28/2006 - David Simpson - .com Solutions Inc. - www.fmpromigrator.com - Initial Release
#
#
#New Record/Request
#Go to Field [ Client_Call_Log::Client_ID ]
#    [ Select/perform ]
end New_Call_Log_Record

on New_Call_Log_Window
#   This script opens a new Call Log window.
#   2/28/2006 - David Simpson - .com Solutions Inc. - www.fmpromigrator.com - Initial Release
#
#
lock screen
#New Window [ Name: "Call Log"; Height: 300; Width: 700; Top: 50; Left: 50 ]
go to card "Client Call Log"
unlock screen
end New_Call_Log_Window
```

If FileMaker scripts have been copied into FmPro Migrator Platinum Edition via the ClipBoard, then these scripts will be converted into LiveCode handlers. The 34 most commonly used (out of 140) ScriptMaker steps are converted directly into LiveCode code. FileMaker script commands such as Go to Layout are converted into the LiveCode "go to card" command, the Freeze Window/Refresh Window commands are converted into "lock screen/unlock screen" commands. FileMaker script commands which fill variables and fields are converted into the LiveCode "put" command.
All of these converted scripts will require additional development after the conversion process has been completed by FmPro Migrator Platinum Edition, as these conversions are primarily intended to reduce the amount of typing required by the LiveCode developer.

After the scripts have been created during the automated conversion process, post processing is done on the scripts using the included FM Functions to LiveCode Mapping.livecode stack. More details on the post processing tasks is described in the Post Conversion Steps part of this manual.

Importing FileMaker Database Info

This document explains the process of converting FileMaker Pro, Microsoft Access or Visual FoxPro projects into LiveCode projects.

FileMaker Pro 11 Notes

Issue #1: Unescaped Unicode Characters
Note: There is a documented issue with FileMaker Pro/Advanced 11 database DDR XML file exporting which can cause problems during the conversion process. FileMaker 11 puts unescaped high ASCII and Unicode characters onto the clipboard and into the DDR XML file. These errors can prevent the conversion process from working properly, as a valid XML file needs to be read for processing purposes. The copying of info to and from the clipboard may also be affected by this issue.

Workaround #1: If this problem affects the database file you are processing, consider switching to FileMaker Pro Advanced 10 for the exporting process.

Workaround #2: If your database also contains FileMaker 11 charts, then consider manually copying only those layouts containing charts into FmPro Migrator via the clipboard.
When switching between different versions of FileMaker for layout importing, select the correct version of FileMaker from the source database menu on the FileMaker tab of the FmPro Migrator main window. FmPro Migrator does handle mixed layout versions without difficulty, as the format version of each layout is checked during the processing, and version differences are handled automatically.

Issue #2: Transferring Container Field Images

FileMaker 11 introduces a new and greatly improved ODBC driver. However FileMaker 11 no longer supports the use of "SELECT * FROM TableName" SQL command in order to retrieve container field data. All previous versions of FileMaker supported the export of the JPEG preview version of the container field contents. FmPro Migrator provides a container field export feature which exports the requested data type into a file onto the disk. But this process only works correctly if all records contain the exact same type of data for exporting, for the selected field.

Workaround #1: Switching back to FileMaker 10 for container field data transfers to SQL database servers, is probably the easiest solution to this limitation. But be aware that the older ODBC drivers (FileMaker 7, 8, 9, 10) may exhibit a bug which can lose records when transferring data. Please double-check the count of records which were transferred to make sure that this problem has not occurred.

Issue #3: Portal Fields

It is more challenging to properly associate FileMaker portal fields with the portal where they are actually used. FmPro Migrator may assign portal fields to the wrong portal.

Workaround #1:
If this happens, you can manually change the DataGrid Row Template assignment within the LiveCode IDE.

Workaround #2:
FileMaker 12+ databases using the .fmp12 file format store this information in a more easily readable format. If you can upgrade a file into the .fmp12 file format, it will be possible to use the Image Export to SQL Databases feature.

FileMaker Pro 12+ Notes

Advantage #1:

FileMaker 12+ database files store information about Portals in a more easily readable format within the XML definition of the layout. Using a FileMaker 12+ version file can make the assignment of fields more accurate during the conversion process. Otherwise, it is possible that FmPro Migrator could assign fields to the wrong DataGrid within the converted stack file. This is one advantage to converting a file from the .fp7 to the the new FileMaker .fmp12 file format.

Pre-Migration Tasks - LiveCode Conversion

Prior to importing FileMaker, Access or Visual FoxPro database applications into FmPro Migrator, review the database schema and make appropriate changes:

1) The each database table must have a primary key column.
2) FmPro Migrator looks for columns having the Unique and Not Empty validation properties in order to automatically determine which column should be created as a Primary Key column in the SQL database. The PK column attributes can be changed after importing by double-clicking the column name displayed in the Fields List on the Tables tab of the Migration Process window.
3) The detailed instructions for data transfer to SQL databases recommends deleting Global, Unstored Calc and Summary fields prior to transferring the data. If these columns are deleted, then the generated stack will display errors about missing columns unless these columns are restored.
4) FileMaker databases containing repeating fields should be redesigned to eliminate the use of repeating fields prior to conversion into the LiveCode application. FmPro Migrator Platinum Edition makes this process possible during the database table migration process. Transfer the repeating fields data to any SQL database using FmPro Migrator Platinum Edition. Then import the related records back into a new table within FileMaker using an ODBC import from the SQL database. Create a relationship from the parent table to the newly imported repeating fields table

in FileMaker. Create portals on layouts to replace the original repeating fields. Then, when the conversion process is done, the portals will be converted into data grid objects in the stack.

A Note About Other Databases

This migration process is specifically optimized for the conversion of FileMaker Pro databases into LiveCode stacks. This is due to the fact that every layout and field in a FileMaker database is expected to be data bound to a database table or column. However, Microsoft Access and Visual FoxPro database applications may have fields which are populated with data via Queries or FoxPro scripts.

After importing one of these other databases into FmPro Migrator, it could be beneficial to perform a conversion of the database into a FileMaker Pro .fp7 database file. Then work on this file to assign tables and fields to each field and layout so that it is functional within FileMaker Pro. Then, when all changes have been made to the FileMaker Pro database, import the FileMaker Pro database into FmPro Migrator for conversion into the LiveCode stack. If these types of changes are not made, it is likely that some converted cards will generate errors due to a missing table or field names.

Importing FileMaker Pro Database Info into FmPro Migrator

1) Download and following the instructions in the How to Import FileMaker Pro Databases into FmPro Migrator PDF manual from the FmPro Migrator support web page. Or, watch the videos linked from the FmPro Migrator support web page. Select Help from the Help menu in FmPro Migrator and your web browser will open this web page. It is generally a good idea to migrate the data into the SQL database, prior to migrating the Layouts into another development environment.

2) If you are transferring data from FileMaker Pro to a SQL database server, then download the appropriate manual on the support page for the destination SQL database. The Pre-Migration Preparation Process PDF provides another resource for migrating the data from FileMaker Pro to SQL database servers.

At the completion of these procedures, your data should already be migrated to the destination SQL database, and the Layouts, Value Lists, Scripts and Relationships should have been imported into FmPro Migrator.

Importing Microsoft Access Database Info

1) Prior to performing any of the migration procedures listed below, it is recommended that you review the database structure of the Microsoft Access database(s) you are migrating. It is important to insure that each table is configured with a Primary Key column. FmPro Migrator looks for columns having the Unique and Not Empty validation properties in order to automatically determine which column should be created as a Primary Key column in the SQL database.

2) Download and following the instructions in the How to Import Microsoft Access Databases into FmPro Migrator PDF manual from the FmPro Migrator support web page. Select Help from the Help menu in FmPro Migrator and your web browser will open this web page.

3) If you are transferring data from Microsoft Access to a SQL database server, then you will likely use an importing procedure to import the data from the Access .mdb/.accdb file(s) into the destination SQL database. [FmPro Migrator supports directly copying data to FileMaker Pro databases, but not into SQL database servers.]

At the completion of these procedures, your data should already be migrated into the destination SQL database, and the Forms/Reports, Value Lists, Scripts and Relationships should have been imported into FmPro Migrator.

Importing Visual FoxPro Applications

1) Prior to performing any of the migration procedures listed below, it is recommended that you review the database structure of the Visual FoxPro project you are migrating. It is important to insure that each table is configured with a Primary Key column. FmPro Migrator looks for columns having the Unique and Not Empty validation properties in order to automatically determine which column should be created as a Primary Key column in the SQL database. Also, columns marked as NOT NULL should not contain NULL values, or the data transfer process will fail.

2) Download and following the instructions in the How to Import Visual FoxPro Projects into FmPro Migrator PDF manual from the FmPro Migrator support web page. Select Help from the Help menu in FmPro Migrator and your web browser will open this web page.

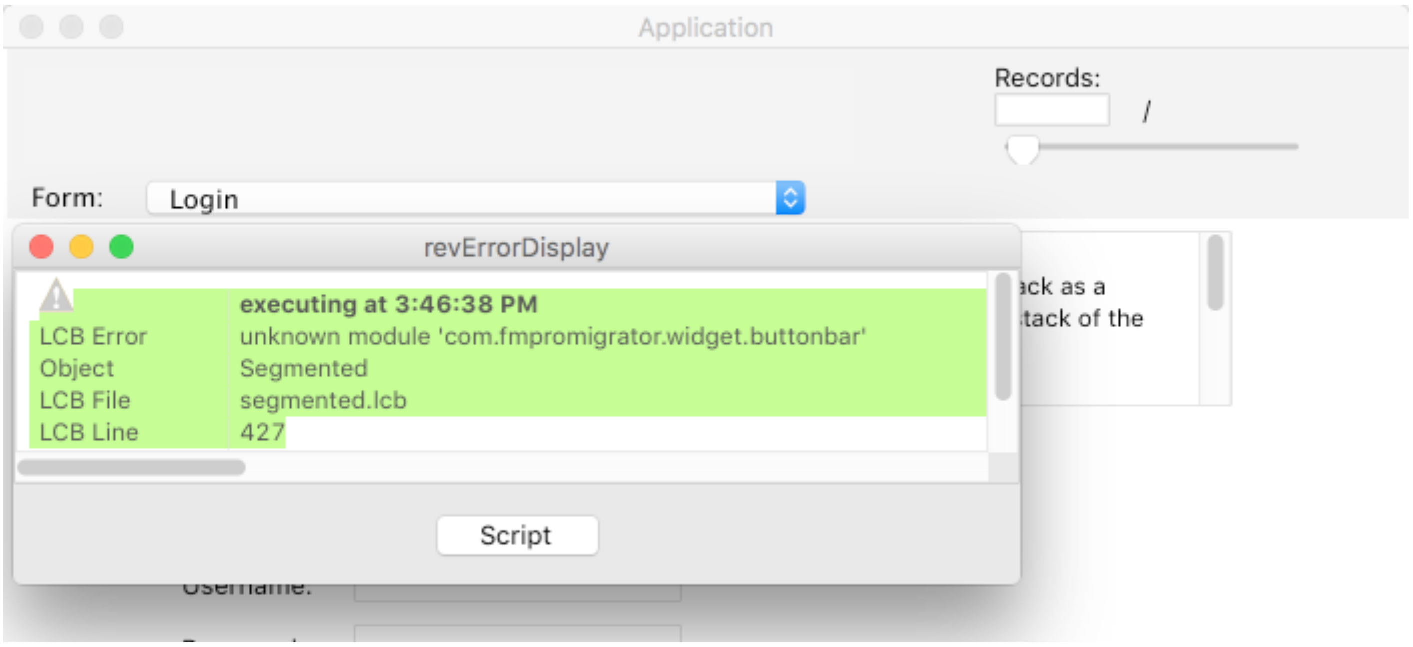
3) If you are transferring data from Microsoft Access to a SQL database server, then you may use an importing procedure to import the data from the DBF file(s) into the destination SQL database. Or you may use FmPro Migrator to transfer data from the DBF files into the destination SQL database server.

At the completion of these procedures, your data should already be migrated into the destination SQL database, and the Forms/Reports, Value Lists, Scripts and Relationships should have been imported into FmPro Migrator.

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.

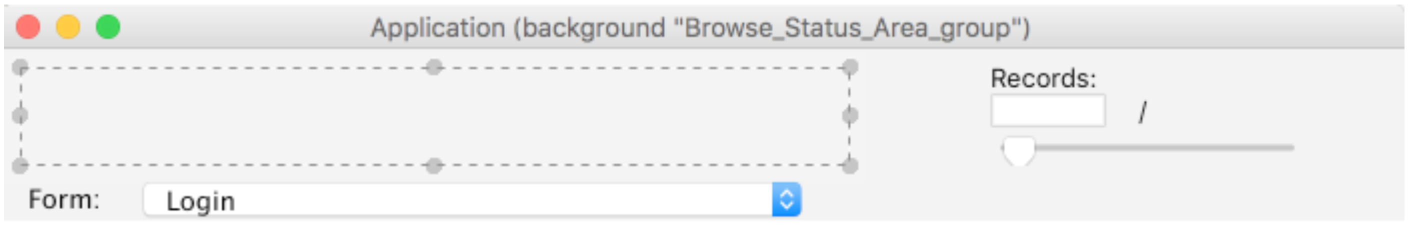
Button Bar Widget - Unknown Module Error



```
executing at 3:46:38 PM
LCB Error    unknown module 'com.fmpromigrator.widget.buttonbar'
Object       Segmented
LCB File     segmented.lcb
LCB Line     427
```

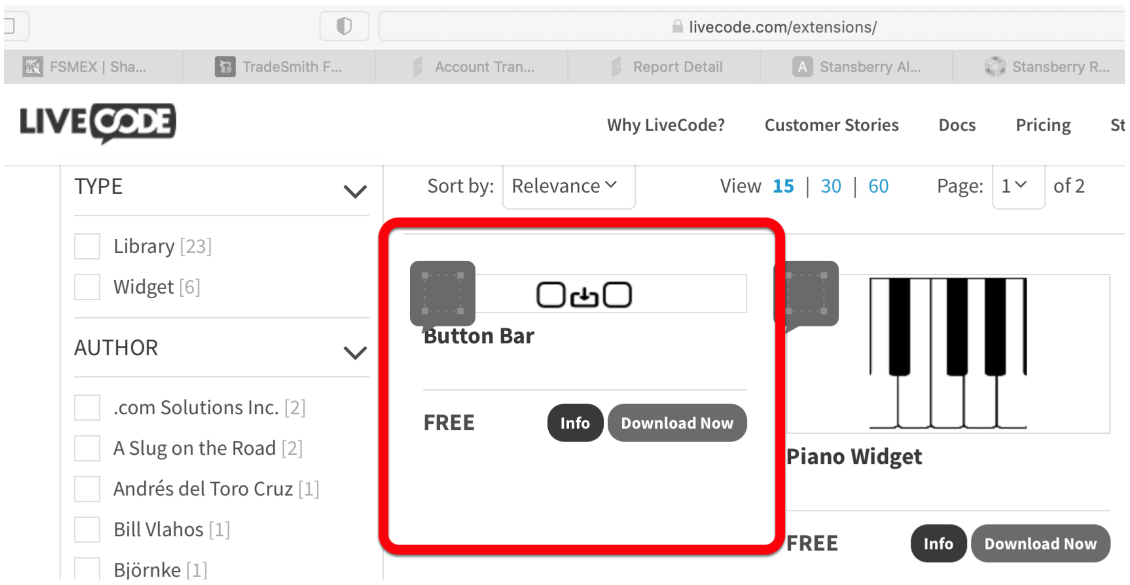
If you open the newly created Application.livecode stack before installing the Button Bar widget, you will see the Unknown Module error dialog.
This error will not occur during the actual conversion process because the Button Bar widget is built into FmPro Migrator and is displayed on screen during the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

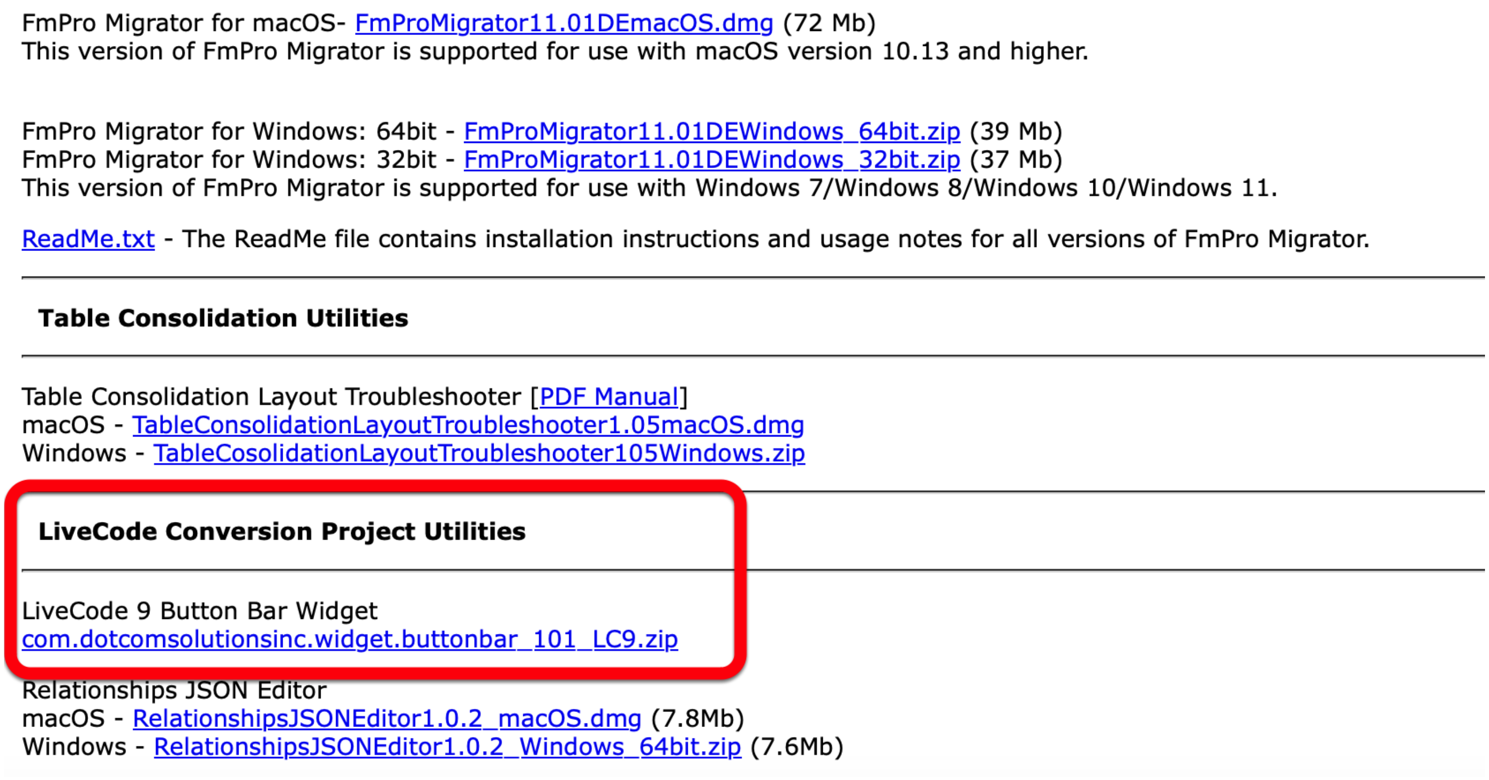
Download Button Bar Widget - LiveCode Store Web Site



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 for macOS- FmProMigrator11.01DEmacOS.dmg (72 Mb)
This version of FmPro Migrator is supported for use with macOS version 10.13 and higher.

FmPro Migrator for Windows: 64bit - FmProMigrator11.01DEWindows_64bit.zip (39 Mb)
FmPro Migrator for Windows: 32bit - FmProMigrator11.01DEWindows_32bit.zip (37 Mb)
This version of FmPro Migrator is supported for use with Windows 7/Windows 8/Windows 10/Windows 11.

ReadMe.txt - The ReadMe file contains installation instructions and usage notes for all versions of FmPro Migrator.

Table Consolidation Utilities

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

LiveCode Conversion Project Utilities

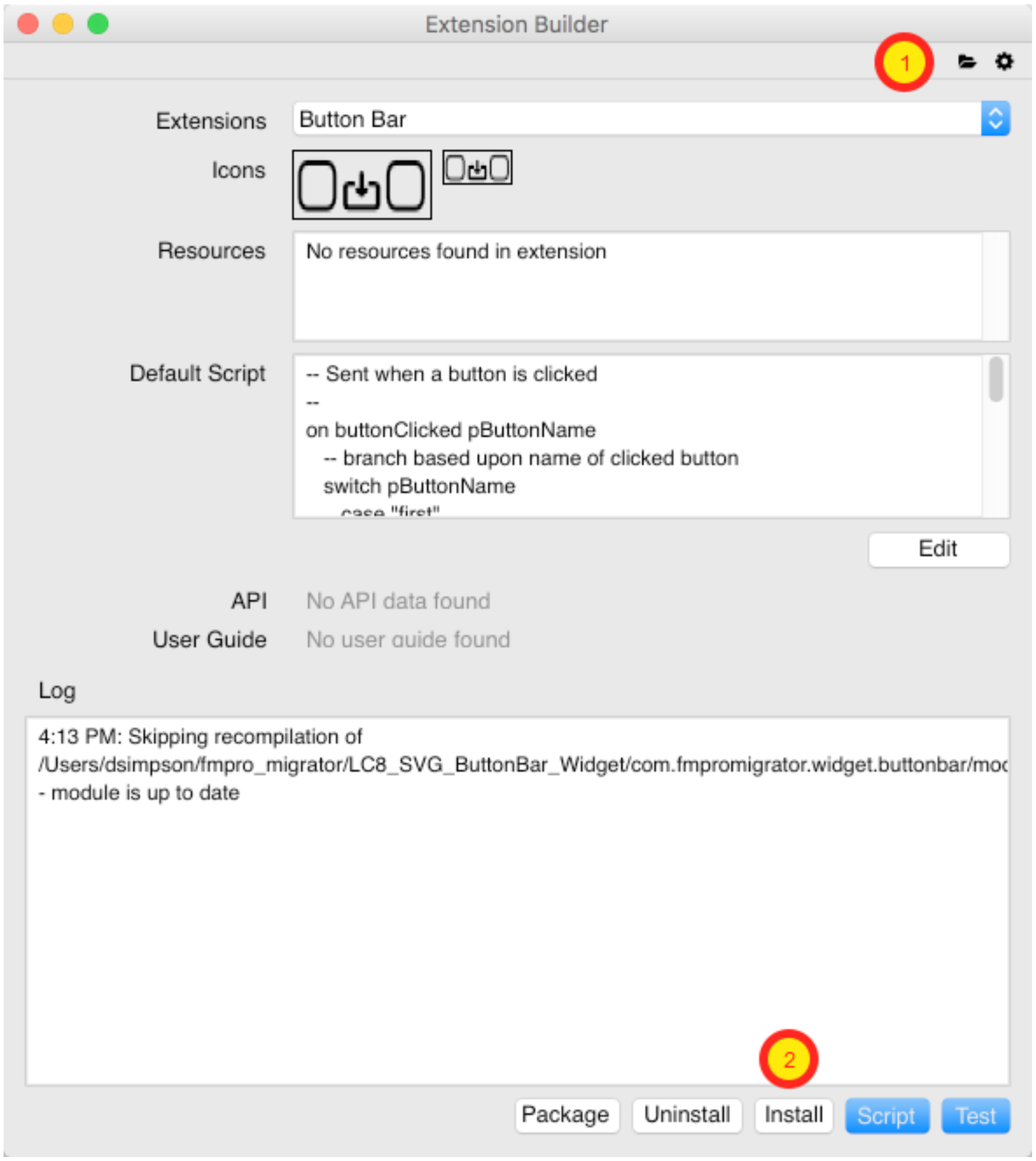
LiveCode 9 Button Bar Widget
com.dotcomsolutionsinc.widget.buttonbar_101_LC9.zip

Relationships JSON Editor
macOS - RelationshipsJSONEditor1.0.2_macOS.dmg (7.8Mb)
Windows - RelationshipsJSONEditor1.0.2_Windows_64bit.zip (7.6Mb)

The Button Bar widget can be downloaded from the same web page where FmPro Migrator was downloaded. Download and unzip the Button Bar widget folder.

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

Extension Builder Window



No resources found in extension

```
-- Sent when a button is clicked
--
on buttonClicked pButtonName
   -- branch based upon name of clicked button
   switch pButtonName
      case "first"
```

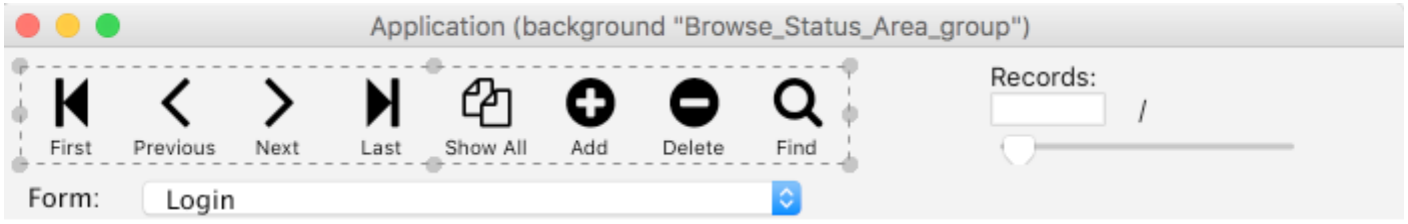
No API data found

No user guide found

```
4:13 PM: Skipping recompilation of
/Users/dsimpson/fmpro_migrator/LC8_SVG_ButtonBar_Widget/com.fmpromigrator.widget.buttonbar/moc
- module is up to date
```

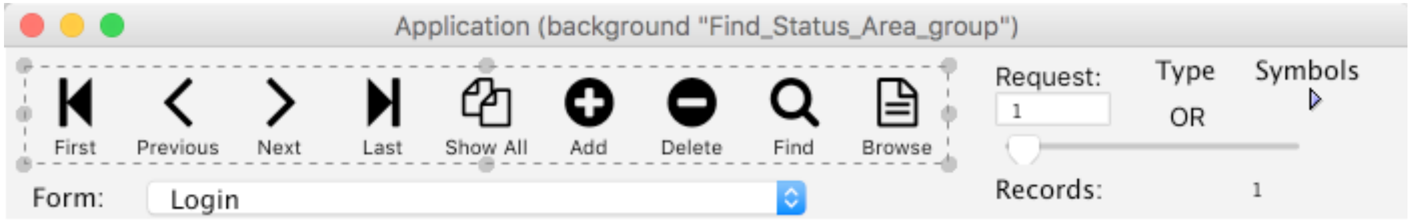
From within the LiveCode IDE, select the Tools -> Extension Builder menu item. (1) Click the folder icon in the upper right corner of the Extension Builder window, select the buttonbar.lcb file from the Button Bar widget folder. (2) Click the Install button.
Quit and re-open the LiveCode IDE so that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

Button Bar Widget - Displayed in Browse Status Area Group



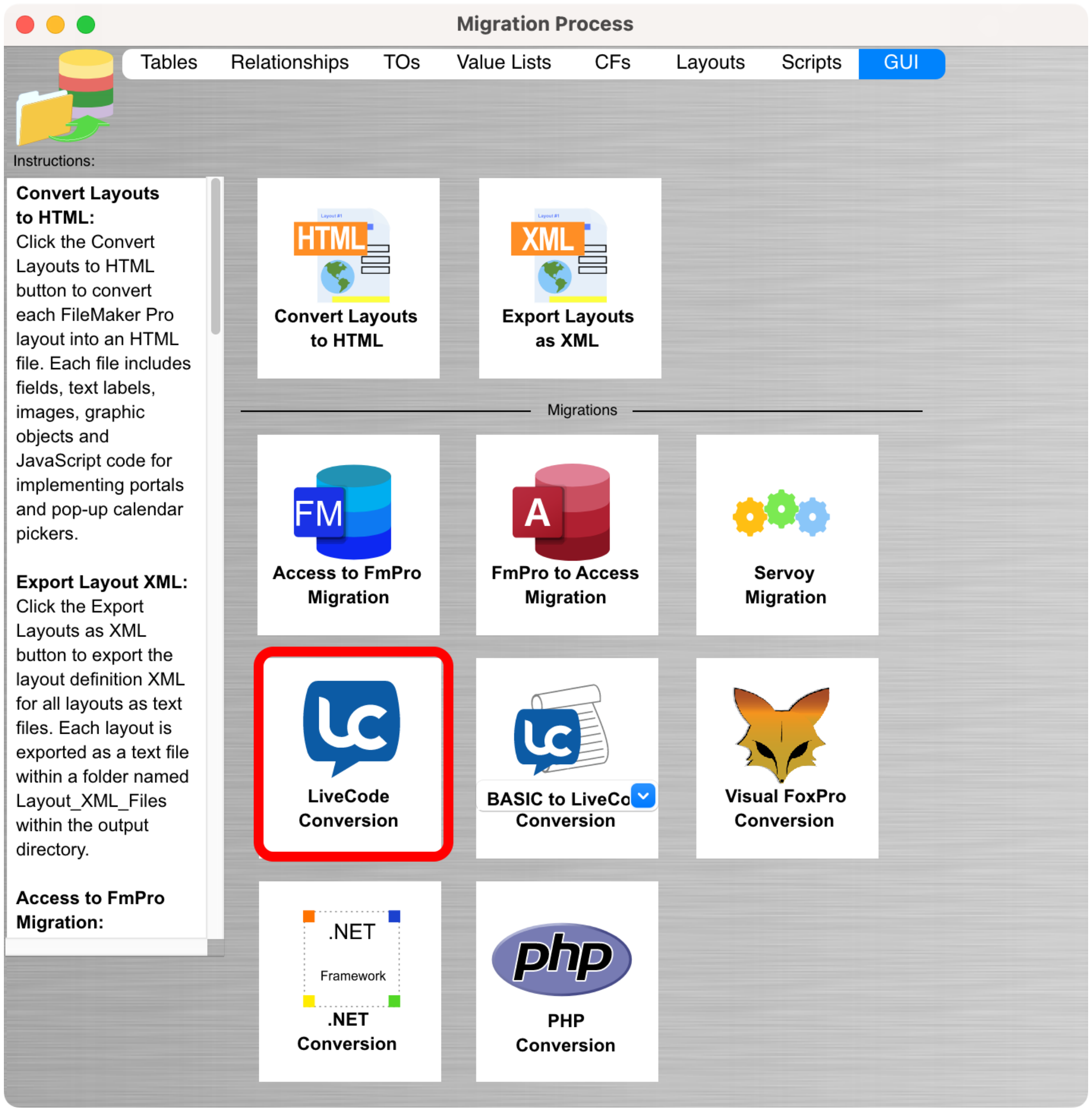
Once the Button Bar widget has been installed, open the newly created Application.livecode LiveCode stack to see it displayed.

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.

Step 3 - Convert Database to LiveCode



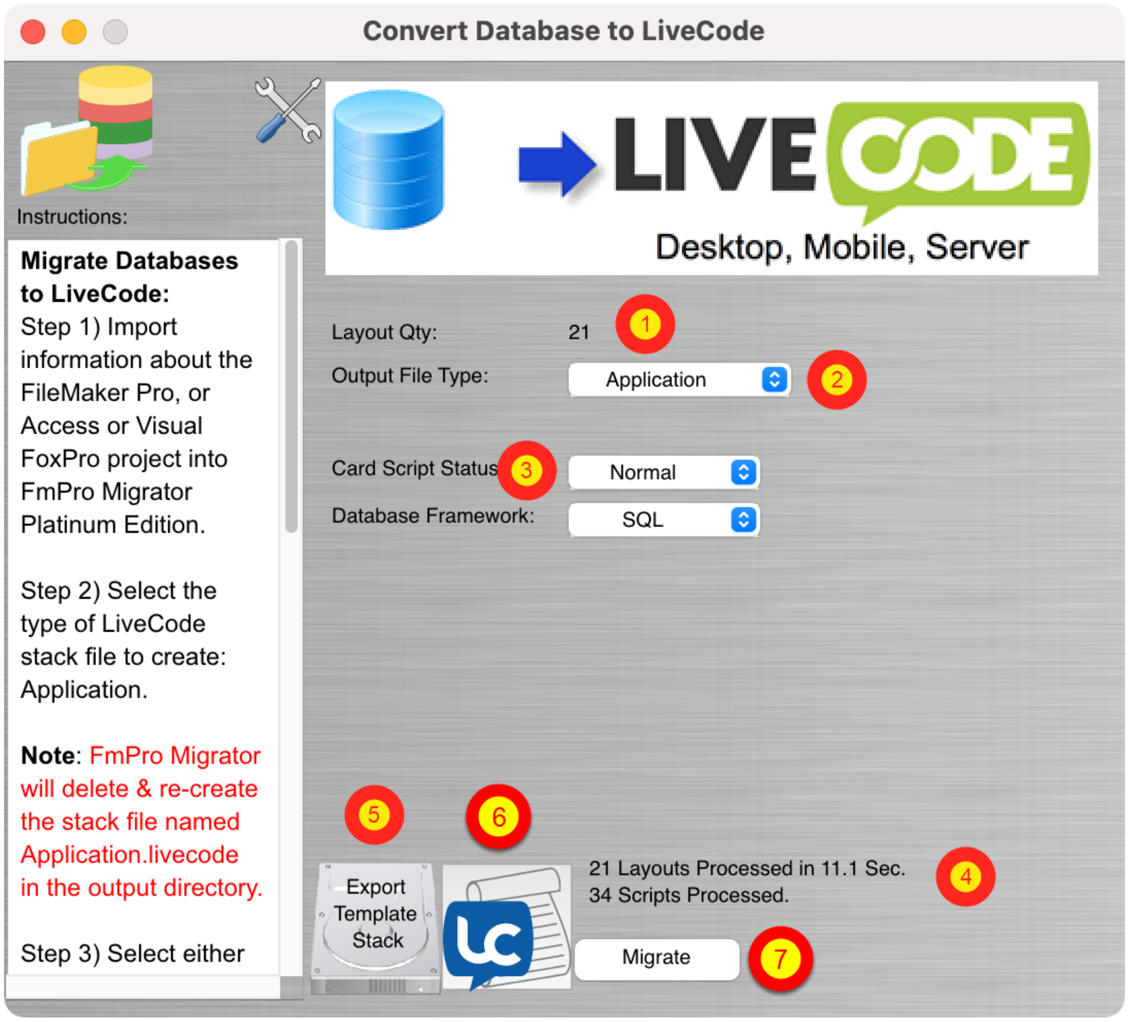
Convert Layouts to HTML:
Click the Convert Layouts to HTML button to convert each FileMaker Pro layout into an HTML file. Each file includes fields, text labels, images, graphic objects and JavaScript code for implementing portals and pop-up calendar pickers.

Export Layout XML:
Click the Export Layouts as XML button to export the layout definition XML for all layouts as text files. Each layout is exported as a text file within a folder named Layout_XML_Files within the output directory.

Access to FmPro Migration:

Click the LiveCode Conversion button to open the LiveCode Conversion window.

LiveCode Conversion Window


Convert Database to LiveCode

Instructions:

Migrate Databases to LiveCode:
Step 1) Import information about the FileMaker Pro, or Access or Visual FoxPro project into FmPro Migrator Platinum Edition.

Step 2) Select the type of LiveCode stack file to create: Application.

Note: FmPro Migrator will delete & re-create the stack file named Application.livecode in the output directory.

Step 3) Select either

Layout Qty: 21
Output File Type: Application
Card Script Status: Normal
Database Framework: SQL

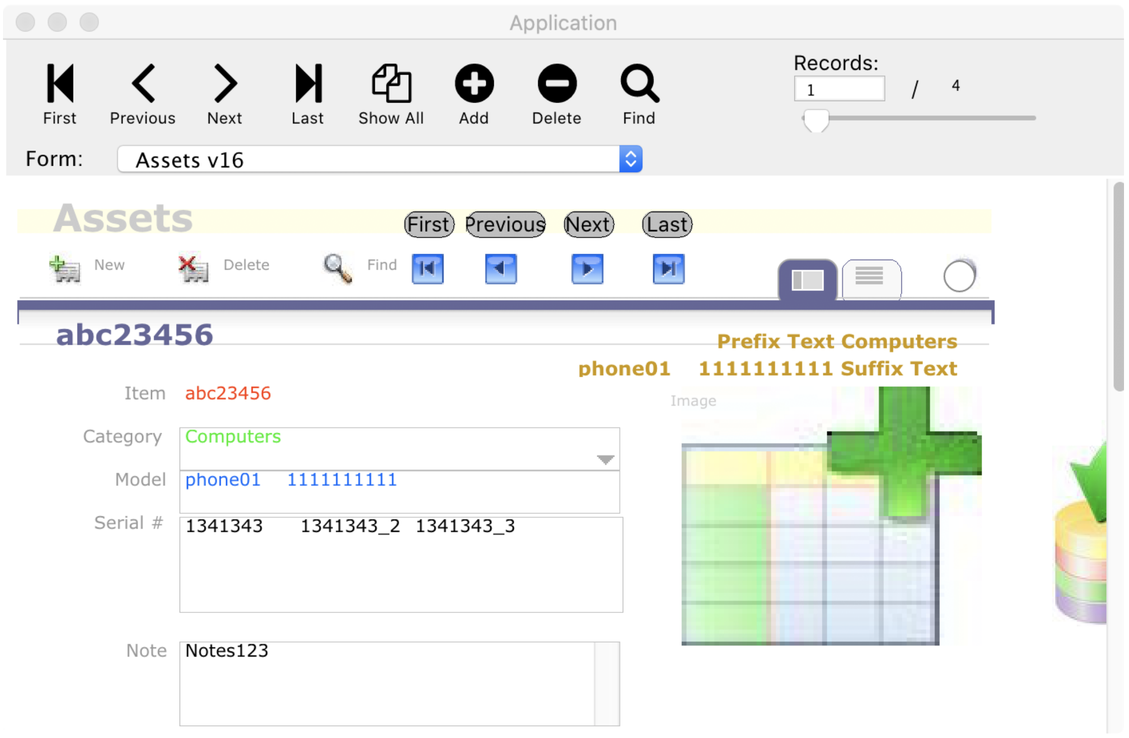
21 Layouts Processed in 11.1 Sec.
34 Scripts Processed.

Some of the features available in the LiveCode Conversion window include:

(1) The Layouts Qty label displays the number of layouts which will be converted. This number represents the number of layouts which have been captured and stored in the FmPro Migrator project file.
(2) Output file types include: Application.
The horizontal Button Bar widget and status area will be added to the stack instead of the previously used vertical status area.
(3) Card Script Status options include: Normal, Commented.
Sometimes you may want to just convert a database and then quickly see the results without converting the data or building a database. Selecting the Commented option comments out the getData and refreshFields handlers on each card. No errors will be displayed navigating between cards due to there not being a database available.
(4) Processing Status - Lists the number of layouts, scripts and elapsed time for the processing.
(5) Export Template Stack button - Exports the template stack into a file named Application.rev in the output directory (overwrites existing copies).

(6) Code Conversion Wokbench button - for converting scripts into LiveCode from FileMaker, Microsoft Access or Visual FoxPro.
(7) Migrate button - Click this button to perform the migration.

Post Conversion Development



abc23456

Prefix Text Computers
phone01 1111111111 Suffix Text

Item abc23456

Category Computers

Model phone01 1111111111

Serial # 1341343 1341343_2 1341343_3

Note Notes123

The Post Conversion Development section of this manual provides a flowchart of the steps needed and LiveCode utility stacks included for use after the stack has been created.

In addition to converting the FileMaker layouts, FmPro Migrator Platinum Edition also converts FileMaker scripts into LiveCode scripts. These scripts will be placed into two output files will be written to the output directory:

FmPro Original Scripts.txt file - This is a text version of the original Script Workspace scripts, and is made available for documentation purposes so that LiveCode developers can review the original unconverted scripts.

FmPro Converted Scripts.txt - This text file contains the converted LiveCode code. Any instructions which could not be converted will be commented and the original comments will be retained within the scripts.

Supported FileMaker Pro Layout Object Types

The following FileMaker Pro Layout object types are currently supported by the conversion process:

Fields
Custom Value Lists
Field Value Lists
Merge Fields
Portals
Text
Lines
Charts [LiveCode 8+]
Rectangles
Rounded Rectangles
Ovals/Circles
Grouped Objects
Images
Buttons
Popover Buttons
Tab Controls
WebViewer

Each FileMaker Pro layout object is re-created as an equivalent LiveCode card object, using the formatting and style attributes of the original object.
Stylized FileMaker Pro layout text objects are converted into LiveCode text labels, having the embedded text styles of the original object defined with the HTMLText property.

The supported field display options include Check Box Set, Radio Button Set, Pop-Up Menu, Drop Down List and Calendar. Pop-Up menus and Calendar objects are created with embedded Rev script code to automatically populate the underlying field.

Note: The Chart object is converted into a LineGraph widget when LiveCode 8+ versions are selected. Some additional work will be required to populate the LineGraph widget with data, as it is placed onto the card as a placeholder. Since the LineGraph widget does not support all of the chart types supported by FileMaker, 3rd party charting tools including ChartsMaker and ChartsEngine may also be considered.

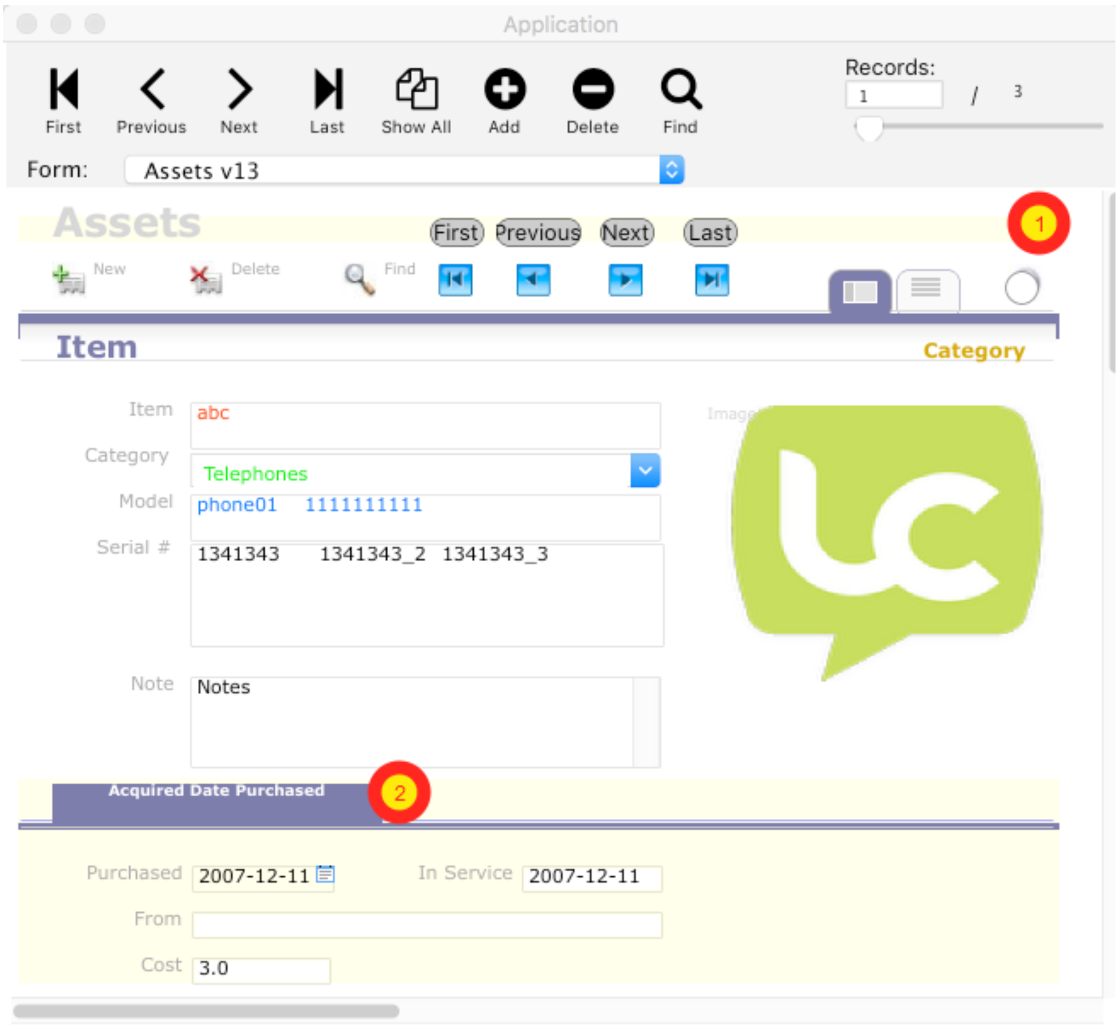
Unsupported FileMaker Pro Layout Object Types

The following FileMaker Pro Layout object types are not currently supported by the conversion process:

Conditional Formatting - Conditional formatting can be used to automatically resize fields and change the appearance or color of field data based upon a calculation formula. The automated resizing or movement of objects can be done by using LiveCode's Geometry Manager. Within LiveCode, the movement of objects can even be made dependent upon the movement of other nearby objects, offering more control of objects than is available within FileMaker Pro.

Embedded Page#, Date, Record# Layout objects - These objects should be manually replaced with some other object like a text label holding the data to be displayed. Plus the addition of a little bit of revTalk code to display the proper Page#, Date etc. However the record# is presently displayed within the record number field in the status area, under the record navigation slider.

Cosmetic Changes Required - Example #1



Records: 1 / 3

Form: Assets v13

Item: abc

Category: Telephones

Model: phone01 1111111111

Serial #: 1341343 1341343_2 1341343_3

Note: Notes

Purchased: 2007-12-11 In Service: 2007-12-11

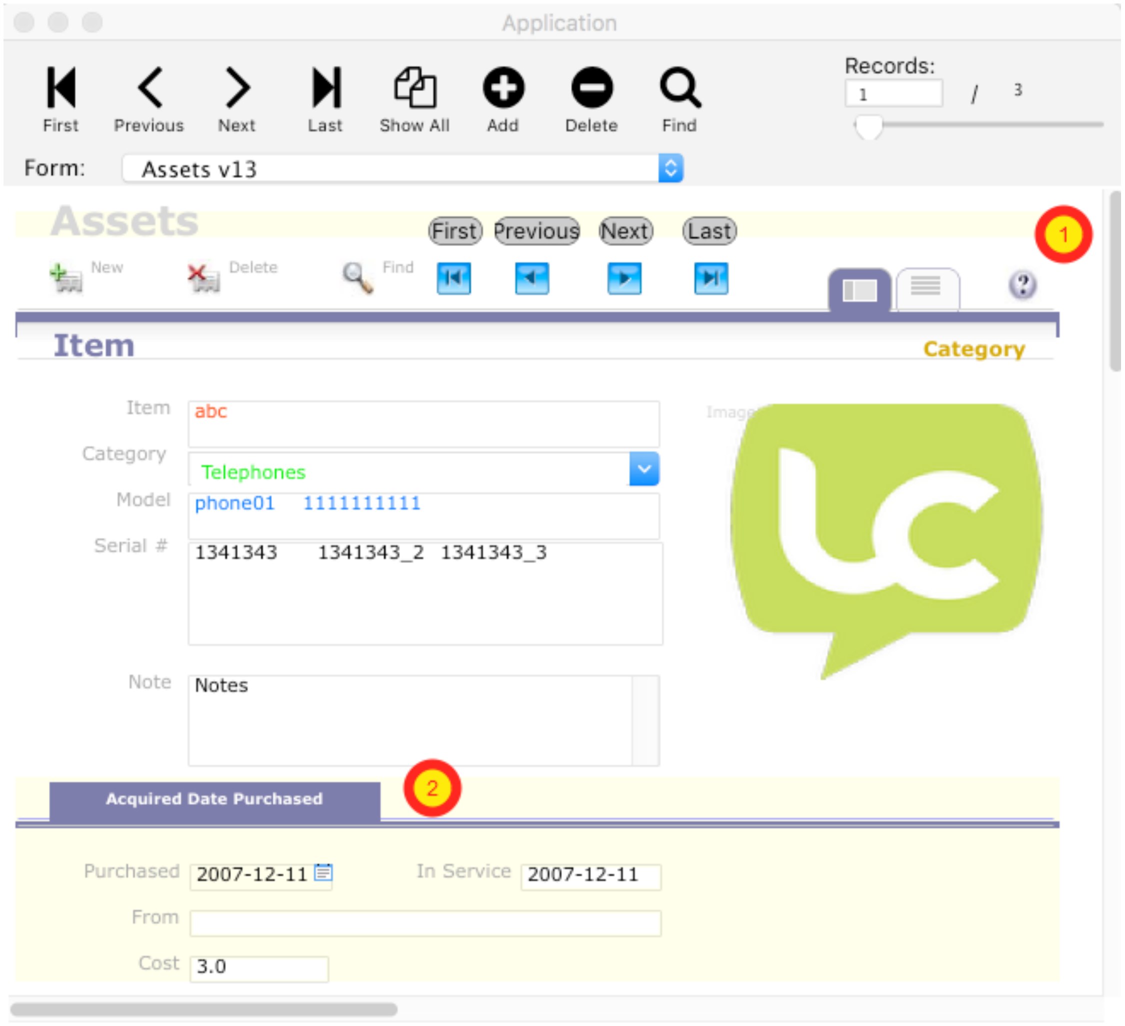
From:

Cost: 3.0

In general, each created object within the LiveCode stack will appear very similar to its counterpart in the original application. However, some changes may be required within the generated stack file. This image of a converted LiveCode stack file shows 3 changes which should be made:
1) The Help button icon has been created, but the (?) image is sitting under the shadowed circle

object. FmPro Migrator Platinum Edition builds each layout object in the order it is found within the Layout XML file. Therefore some objects may be created in the wrong order for display purposes. The solution is to change the layer of each object so that they display properly. The shadowed circle object was also moved left by 1 pixel.
2) The text over the purple rectangle object is displayed a little bit too high.

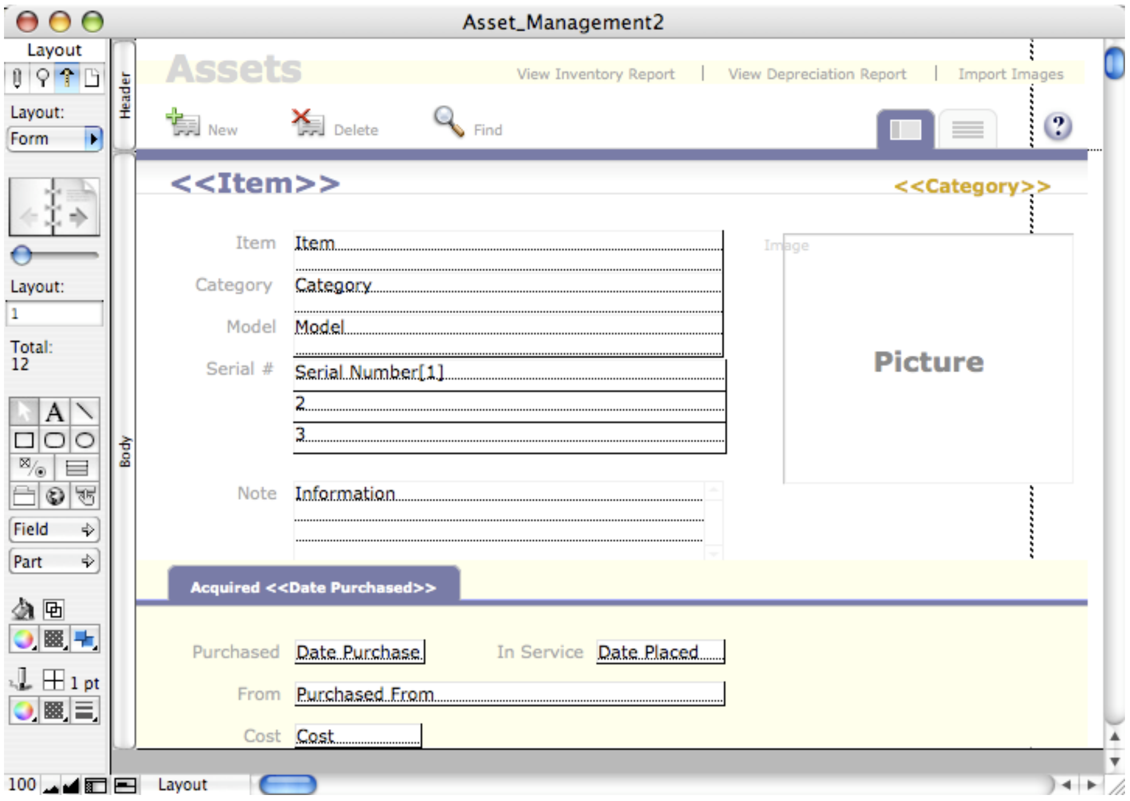
Cosmetic Changes Applied - Example #1



Item	abc
Category	Telephones
Model	phone01 1111111111
Serial #	1341343 1341343_2 1341343_3
Note	Notes
Purchased	2007-12-11
In Service	2007-12-11
From	
Cost	3.0

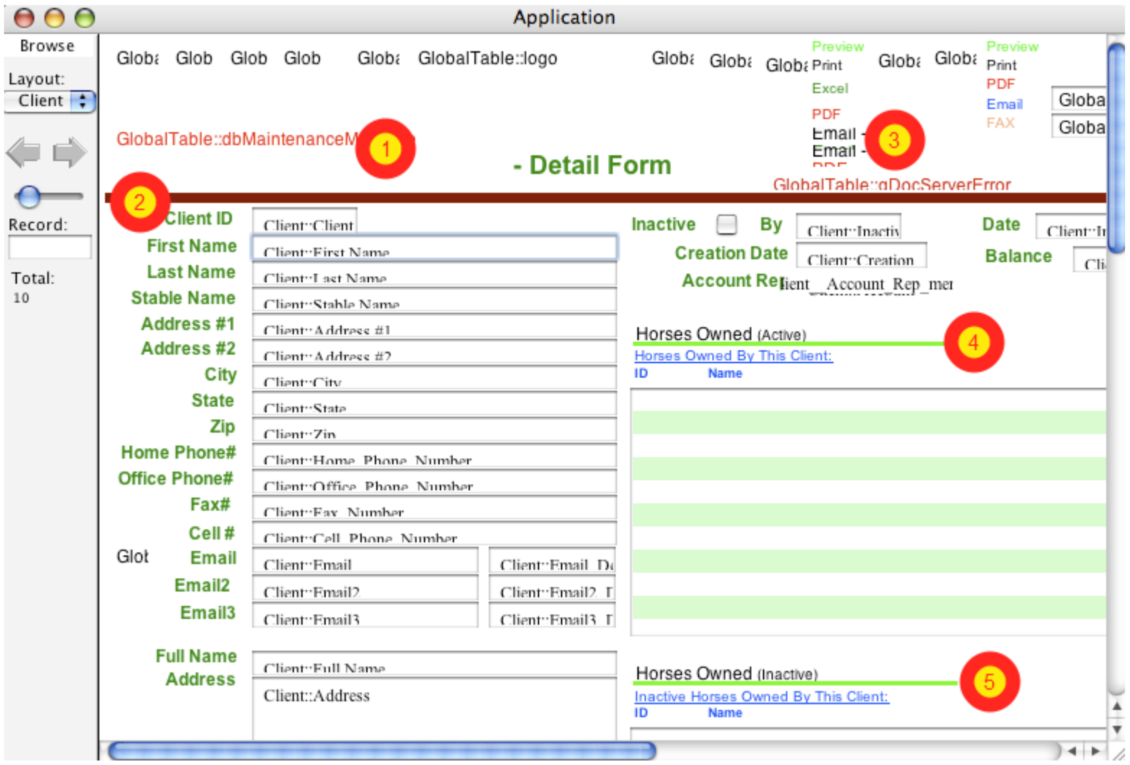
Click on the group named "Layout_Objects_Group", click the Edit Group button in the LiveCode IDE to make changes to the objects on the card.
This image shows the stack after the 3 cosmetic changes were made.
1) The graphic was adjusted by sending the white circle to the back of the canvas.
2) The text label margin property was adjusted from 2 to 8 to change its position to be more readable.

Original FileMaker Pro Layout



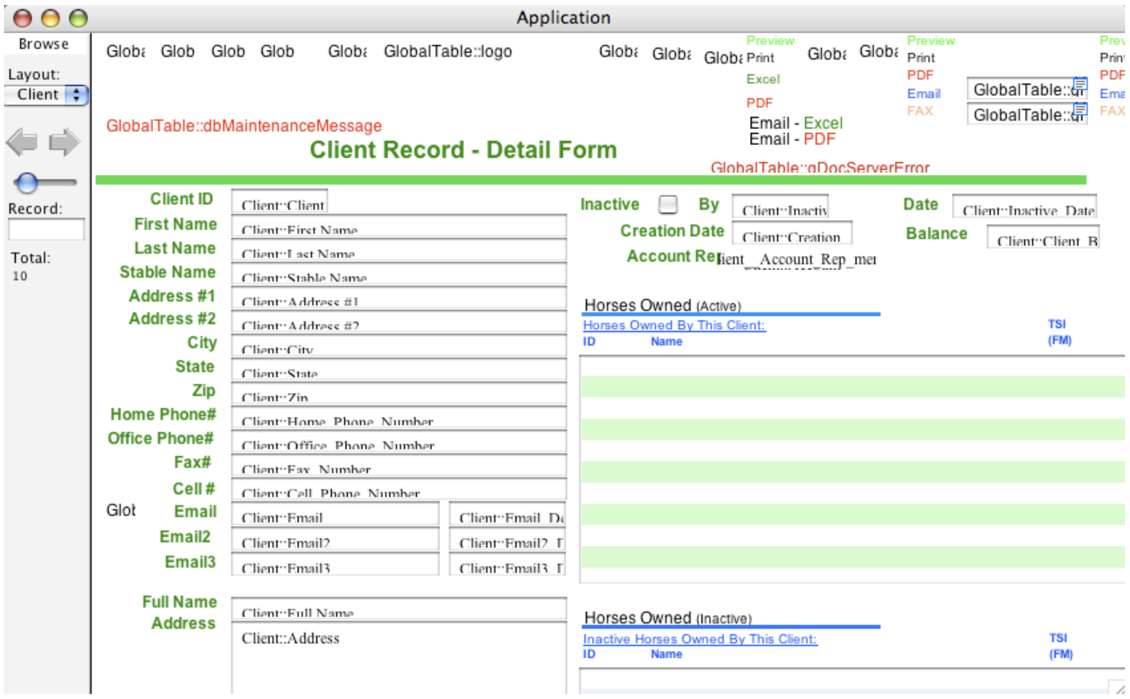
And for comparison purposes, here is an image of the original FileMaker Pro Layout.

Cosmetic Changes Required - Example #2



Here is another converted Layout stack file which needs different cosmetic changes:
1) FmPro Migrator Platinum Edition creates each LiveCode field as an opaque field, in order to insure that fields having a background color are displayed properly. Text objects however, are created with their opaque property set to false, allowing other objects to show thru. But the dbMaintenanceMessage object shown on this layout is not a Text object, it is a field object, so it has been created to be opaque - thus hiding the text label which it overlaps. Therefore in this location, the dbMaintenanceMessage field should have its opaque property set to false.
2) This horizontal line is supposed to be colored green, but the correct color was not properly located within the original XML code. This same problem also occurs with the objects marked (4) and (5). However the remaining horizontal line objects on this same layout (not shown) did retain their correct colors.
3) These two text label buttons ended up wrapping within the field. Changing the text margin from 2 to 4 and widening the fields solved these issues.

Cosmetic Changes Applied - Example #2


Application

Browse
Layout:
Client
Record:
Total:
10

Globa Glob Glob Glob Globa GlobalTable::logo Globa Globa Globa Globa Globa

Preview
Print
Excel
PDF

Preview
Print
PDF
Email
FAX

Email - Excel
Email - PDF

GlobalTable::dbMaintenanceMessage

Client Record - Detail Form

GlobalTable::gDocServerError

Client ID	Client::Client
First Name	Client::First Name
Last Name	Client::Last Name
Stable Name	Client::Stable Name
Address #1	Client::Address #1
Address #2	Client::Address #2
City	Client::City
State	Client::State
Zip	Client::Zip
Home Phone#	Client::Home_Phone_Number
Office Phone#	Client::Office_Phone_Number
Fax#	Client::Fax_Number
Cell #	Client::Cell_Phone_Number
Email	Client::Email
Email2	Client::Email2
Email3	Client::Email3
Full Name	Client::Full Name
Address	Client::Address

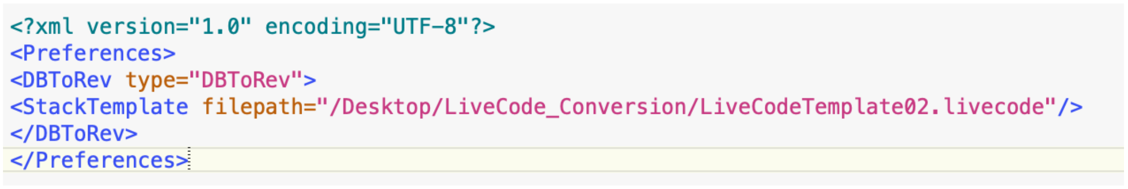
Inactive By Client::Inactiv Date Client::Inactive_Date
Creation Date Client::Creation Balance Client::Client_B
Account Re lient_Account_Rep_mer

Horses Owned (Active)
Horses Owned By This Client:
ID Name TSI (FM)

Horses Owned (Inactive)
Inactive Horses Owned By This Client:
ID Name TSI (FM)

This image shows the stack after the 5 cosmetic changes were made.

Fmig_Preferences.xml File



```
<?xml version="1.0" encoding="UTF-8"?>
<Preferences>
<DBToRev type="DBToRev">
<StackTemplate filepath="/Desktop/LiveCode_Conversion/LiveCodeTemplate02.livecode"/>
</DBToRev>
</Preferences>
```

FmPro Migrator utilizes an optional XML preferences file which can be used to customize the DB To LiveCode conversion process. This XML template file can be placed into the FmPro Migrator directory within the /User/Library/Preferences (or My Documents on Windows) directory to customize the generated stack file.

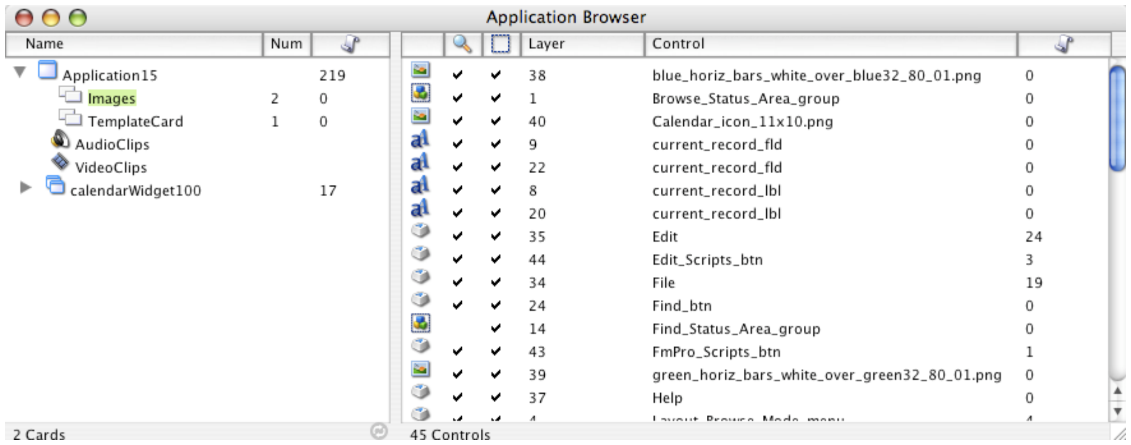
Using a Different StackTemplate File

The StackTemplate option enables LiveCode developers to extract and customize the standard template stack embedded within FmPro Migrator. Click on the Export Template Stack button on the Database to LiveCode window to extract the existing template stack file to the output directory (it will overwrite any existing file). You should move or rename this file and update the XML preferences file to point to the modified version of the file.

The custom version of your template stack file will be copied into the output directory, renamed as Application.rev and used as the destination stack where all of the converted layout objects will be created.

This means that you can add that one line command from article #3 into the showAllRecords handler in the card stack, and that correction along with any other changes you make will automatically be used as your new template stack. You might also want to change the look of the status area. You could create a new status area across the top of the card, to simulate the look of newer versions of FileMaker Pro. Or you might turn the status area into a floating palette.

The Template Stack



Application Browser

Name	Num	
Application15		219
Images	2	0
TemplateCard	1	0
AudioClips		
VideoClips		
calendarWidget100		17

2 Cards

		Layer	Control	
✓	✓	38	blue_horiz_bars_white_over_blue32_80_01.png	0
✓	✓	1	Browse_Status_Area_group	0
✓	✓	40	Calendar_icon_11x10.png	0
✓	✓	9	current_record_fld	0
✓	✓	22	current_record_fld	0
✓	✓	8	current_record_lbl	0
✓	✓	20	current_record_lbl	0
✓	✓	35	Edit	24
✓	✓	44	Edit_Scripts_btn	3
✓	✓	34	File	19
✓	✓	24	Find_btn	0
	✓	14	Find_Status_Area_group	0
✓	✓	43	FmPro_Scripts_btn	1
✓	✓	39	green_horiz_bars_white_over_green32_80_01.png	0
✓	✓	37	Help	0

45 Controls

FmPro Migrator Platinum Edition uses an internally-stored template file for the LiveCode stack which it generates from the FileMaker Pro Layouts. This file is written to the output directory as a file named Application.rev at the start of the conversion process.
LiveCode developers can supply their own stack in place of the Application.rev file, in order to make use of their own customized handlers and graphics. FmPro Migrator Platinum Edition always deletes and overwrites any existing Application.rev stack file.

Areplacement template stack can be specified in the FmPro_Migrator_Preferences.xml file.

Areplacement template stack should have the following features, in order to be compatible with FmPro Migrator Platinum Edition:

setLayoutList - stack-level handler - This handler puts the list of cards into the Layout list menus of the Browse and Find groups. If you don't want to have this feature, you can simply include a handler having this same name but containing no executable code.

displayCalendar - stack-level function - This handler displays the calendar pop-up menu within fields which have this feature configured within the original FileMaker Pro Layout. This function returns empty or the date picked by the user.

browserInit
browserGo
browserEnsure
browserFinalize
browserBack
browserForward
browserRefresh
browserStop
These stack-level handlers control one ore more revBrowser controls on a card. Each handler takes a parameter (pBrowserNum) as the number representing the Browser control, and is used

to define its name on the card. There could be multiple FileMaker WebViewer objects on a single Layout, therefore each WebViewer will be converted into a separate Revolution Browser control having a different name on the card.

statusAreaWidth - stack-level customProperty - This custom property indicates the width of the Status Area at the left side of the window. The Layout_Objects_Group is automatically moved over to accommodate the width of the Status Area after all of the cards have been created. Set this custom property to 0 if you don't want to have a Status Area within your template stack.

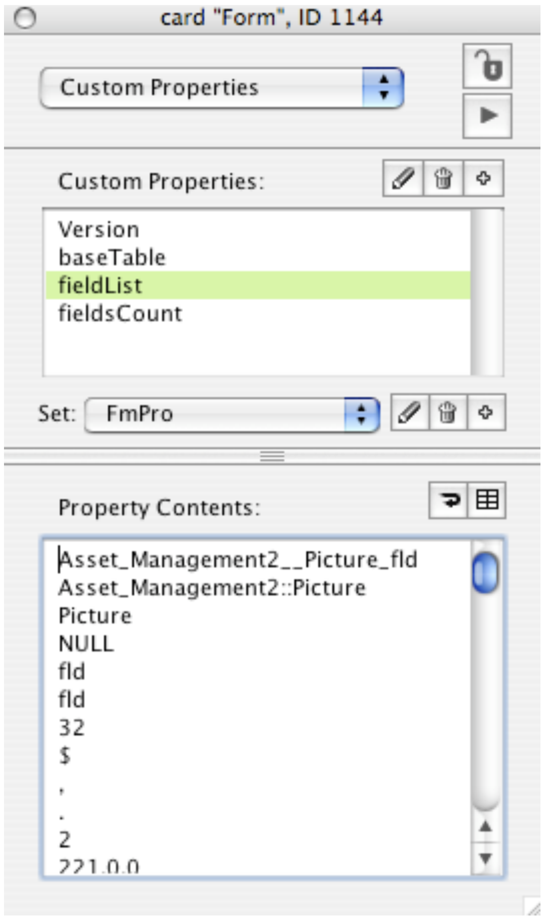
templateStackName - stack-level customProperty - This custom property provides the name of the template stack file.

segmented Widget - When FmPro Migrator builds a segmented widget, it copies the Segment Group containing a sample segmented widget from the Images card of the template stack. Updating the properties of this segmented widget will cause all of the newly created segmented widgets to have the same basic color properties for instance.

FmPro Custom Property Set

Each card in the generated LiveCode stack file includes a custom property set named FmPro. This custom property set includes detailed info about the field objects which have been created on the card by FmPro Migrator Platinum Edition. This information is expected to be useful for LiveCode developers who want to create data entry/validation and data display handlers for each of the fields on the card. This info is also used by the handlers which clear field data upon entering Find mode and when filling each field with data.
Having the fields listed in a custom property set also means that developers don't have to manually type in each of the field object names into their code either. Just loop thru the lines of data in the custom property set to work with the names and data types for each field on the card.

FmPro - customPropertySet Overview



```
Asset_Management2__Picture_fld
Asset_Management2::Picture
Picture
NULL
fld
fld
32
$
,
.
2
221,0,0
```

There are 4 customProperties within the FmPro customPropertySet.

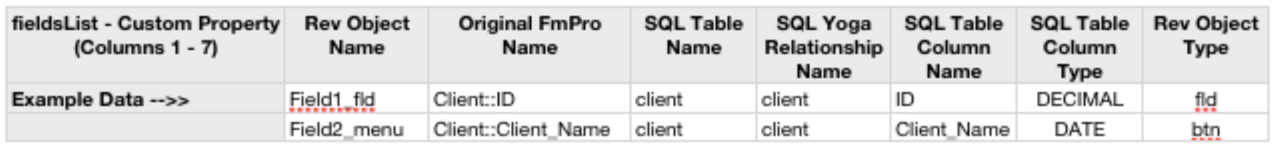
Version - customProperty

The Version customProperty defines the version of the customPropertySet schema. The current value for the Version customProperty is 1, this value will change as enhancements are made to the FmPro customPropertySet.

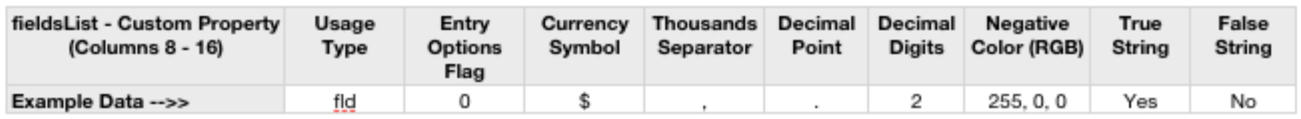
baseTable - customProperty

The baseTable customProperty represents the name of the base table referenced by the original layout.

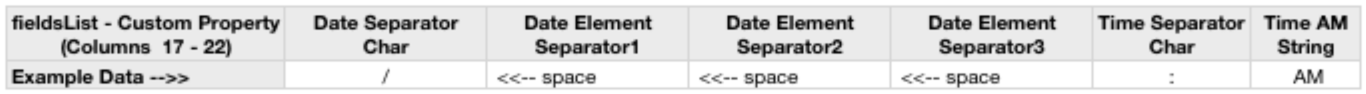
fieldList - customProperty



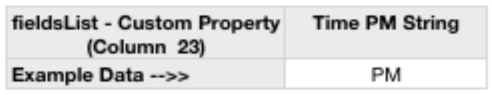
fieldsList - Custom Property (Columns 1 - 7)	Rev Object Name	Original FmPro Name	SQL Table Name	SQL Yoga Relationship Name	SQL Table Column Name	SQL Table Column Type	Rev Object Type
Example Data -->>	Field1_fld	Client::ID	client	client	ID	DECIMAL	fld
	Field2_menu	Client::Client_Name	client	client	Client_Name	DATE	btn



fieldsList - Custom Property (Columns 8 - 16)	Usage Type	Entry Options Flag	Currency Symbol	Thousands Separator	Decimal Point	Decimal Digits	Negative Color (RGB)	True String	False String
Example Data -->>	fld	0	$	,	.	2	255, 0, 0	Yes	No



fieldsList - Custom Property (Columns 17 - 22)	Date Separator Char	Date Element Separator1	Date Element Separator2	Date Element Separator3	Time Separator Char	Time AM String
Example Data -->>	/	<<-- space	<<-- space	<<-- space	:	AM



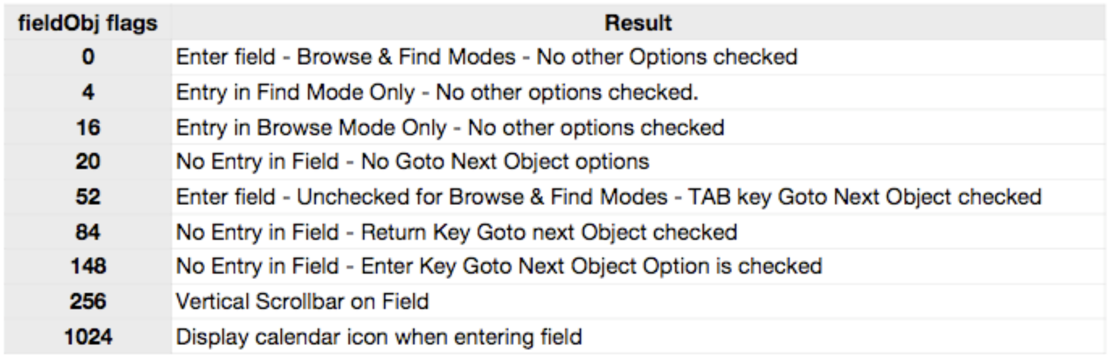
fieldsList - Custom Property (Column 23)	Time PM String
Example Data -->>	PM

The fieldsList is a TAB delimited list providing 23 parameters for each field which has been created on the card. These parameters are shown in the screenshot with sample values and are listed below:

(1) **Rev Object Name** - The name given to the LiveCode field when it was created.
(2) **Original FileMaker Pro Field Name** - This is the name of the Table Occurrence::Fieldname in the original FileMaker Pro database.
(3) **SQL Table Name** - The converted name of the original table, as it appears in the SQL database.
(4) **SQL Yoga Relationship Name** - If the field is in another table, the relationship name to use to get the data from the table.
(5) **SQL Table Column Name** - The converted name of the SQL database column name.
(6) **SQL Table Column Type** - The type of data contained in the field, matching the field types for the destination SQL database.
(7) **LiveCode Object Type** - The type of LiveCode object. Use this object type and its name when

referencing the object.
(8) **Usage Type** - The way in which the object is used on the card, either as a field or a menu. This parameter indicates whether you need to put data into the field (for a field) or into the label of an object (for a menu).
(9) **Entry Options Flag** - The entry options for entry of data within a field, as defined by the fieldObj flags XML parameter in the original layout. Multiple options may be applied to a field simultaneously. To determine if a particular value is selected, use the bitAND Revolution instruction. See the following screenshot for more details.
(10) **Currency Symbol** - The symbol used for currency values.
(11) **Thousands Separator Symbol** - The symbol used to separate thousands, in numeric values.
(12) **Decimal Point Symbol** - The symbol used for decimal points in numeric values.
(13) **Number of Decimal Digits** - The number of values displayed to the right of the decimal point symbol
(14) **Negative Color (RGB)** - The RGB values, separated by commas which should be used for negative numeric values.
(15) **True String** - The text string used for true values.
(16) **False String** - The text string used for false values.
(17) **Date Separator Char** - The symbol used for separating short date values. (i.e. MM/DD/YYYY).
(18) **Date Element Separator1** - The first date separator character string, used when displaying dates in long format.
(19) **Date Element Separator2** - The second date separator character string, used when displaying dates in long format.
(20) **Date Element Separator3** - The third date separator character string, used when displaying dates in long format.
(21) **Time Separator Char** - The symbol used for separating time values (i.e. HH:MM:SS)
(22) **Time AM String** - The suffix string used for displaying AM time values.
(23) **Time PM String** - The suffix string used for displaying PM time values.

Entry Options Details - (fieldObj flags values)



fieldObj flags	Result
0	Enter field - Browse & Find Modes - No other Options checked
4	Entry in Find Mode Only - No other options checked.
16	Entry in Browse Mode Only - No other options checked
20	No Entry in Field - No Goto Next Object options
52	Enter field - Unchecked for Browse & Find Modes - TAB key Goto Next Object checked
84	No Entry in Field - Return Key Goto next Object checked
148	No Entry in Field - Enter Key Goto Next Object Option is checked
256	Vertical Scrollbar on Field
1024	Display calendar icon when entering field

Multiple entry options can be applied to the same field. To decode these values, use the LiveCode bitAND command as follows:

```
put varFieldEntryOptions bitAnd 4 into tResult
```

```
if tResult = 4 then set the lockText of field Field1_fld to true
```

FileMaker Converted Script Steps

These script steps represent the FileMaker script steps which are directly converted into LiveCode by FmPro Migrator.

Note: Using the Code Conversion Workbench feature of the AI Accelerated Edition of FmPro Migrator enables the conversion of all script steps and lines of code from FileMaker Pro, Microsoft Access and Visual FoxPro.

List of Converted Script Steps

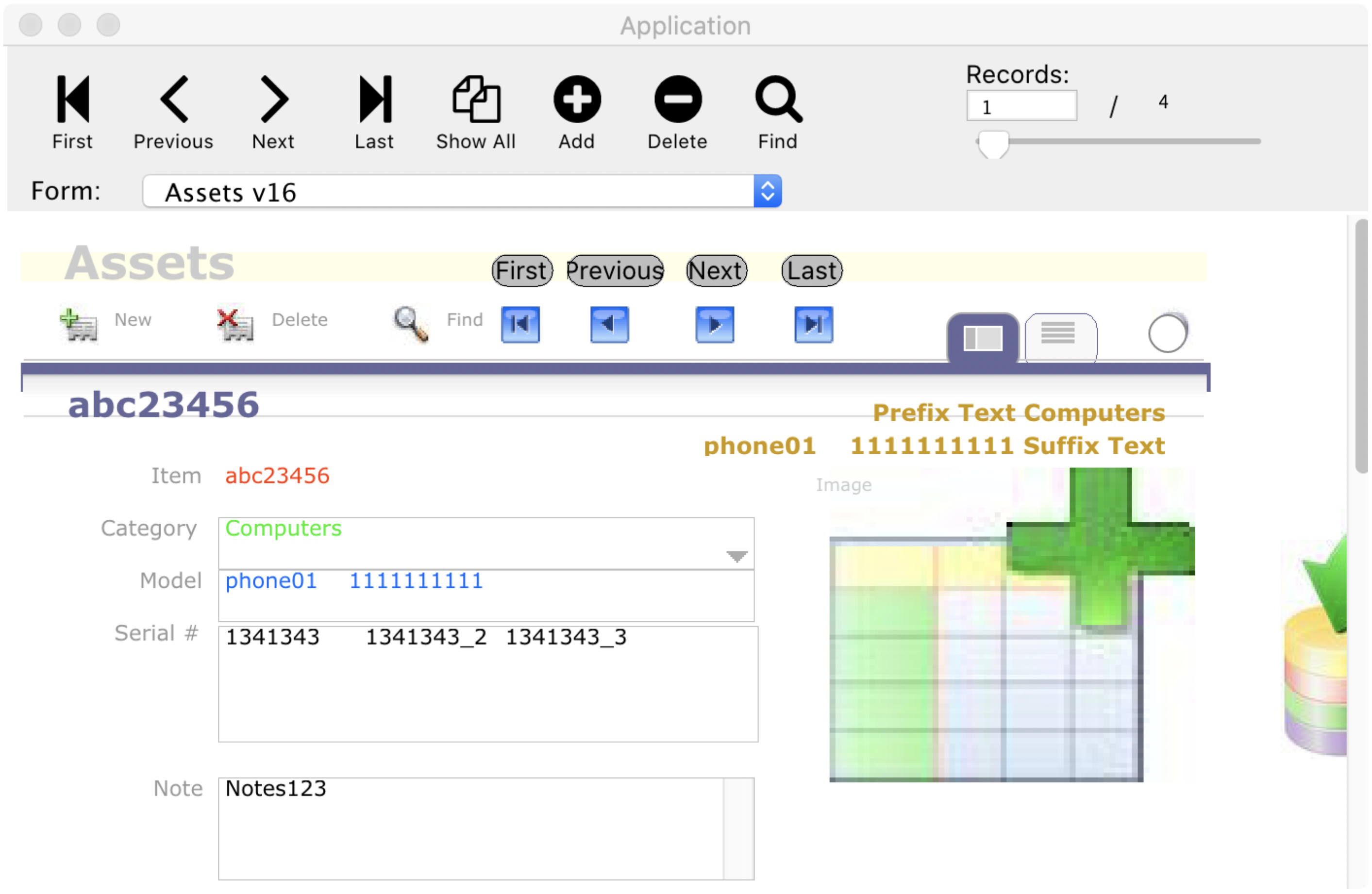
The following list documents the script steps which are converted from FileMaker scripts into LiveCode commands:

Go to Layout
Perform Script
Beep
#
Copy
Cut
If
End If
Else
Exit Application
Exit Script
Freeze Window
Halt Script
Open URL
Paste
Perform AppleScript
Print Setup
Print
Refresh Window
Select All
Send Event
Set Error Capture
Send Mail
Set Field
Set Variable
Show Custom Dialog
Show/Hide Status Area

Enter Find Mode
Enter Browse Mode
Speak

Post Conversion Development

Overview



abc23456

Prefix Text Computers

phone01 1111111111 Suffix Text

Item abc23456

Category Computers

Model phone01 1111111111

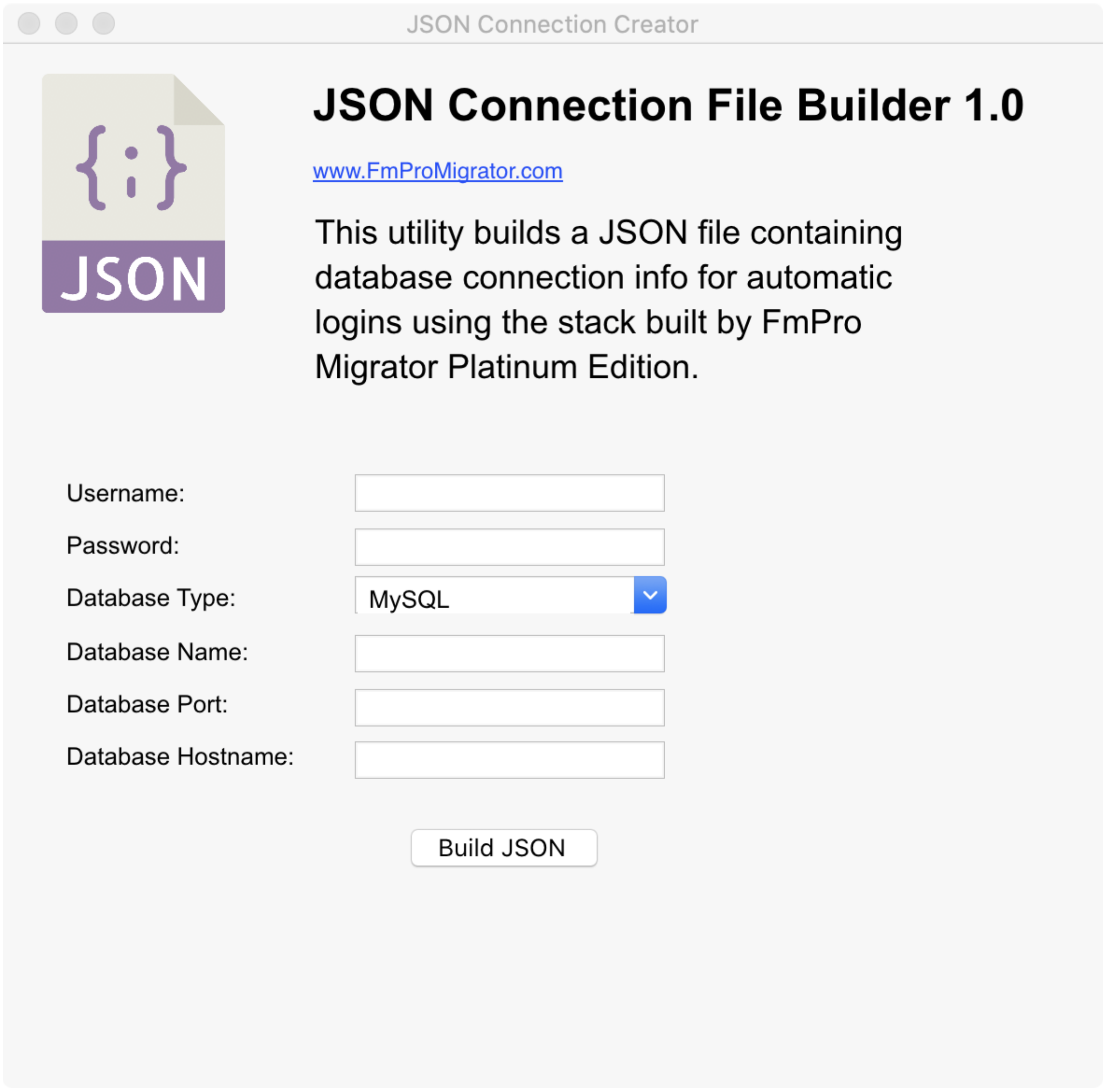
Serial # 1341343 1341343_2 1341343_3

Note Notes123

Aconverted application will generally display data in most fields and portals and provide record and form navigation via the Button Bar status area at the top of the window.
There will be additional steps required in order to have a fully functional application ready for production deployment. Asummary of these tools is discussed below.

The first tool you will probably use is the JSON Connection File Builder utility. This tool makes it quicker and easier for you to log in the converted application into the database server without manually re-typing login credentials many times per day.

JSON Connection File Builder



JSON Connection File Builder 1.0

www.FmProMigrator.com

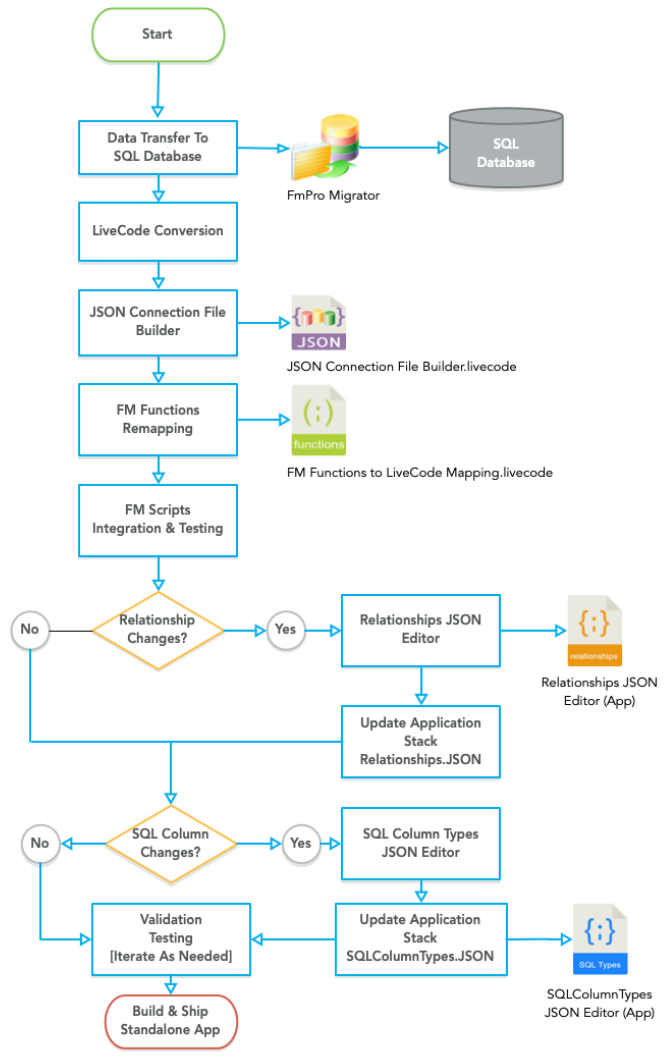
This utility builds a JSON file containing database connection info for automatic logins using the stack built by FmPro Migrator Platinum Edition.

Username:	
Password:	
Database Type:	MySQL
Database Name:	
Database Port:	
Database Hostname:	

The JSON Connection File Builder builds a database connection JSON file named DBConnection.JSON.
This JSON connection file has 2 main uses:
1) During development, you can quickly log the application under development into your test server without having to manually enter connection details.
2) For production usage, you can create connection files having the database type, name, port, hostname info pre-filled for users to log into the database server. Fields which are missing from the DBConnection.JSON file will be filled in by the user during login.

For more details, see the the section of this manual describing the JSON Connection File Builder.livecode stack.

Process Flowchart


Start
Data Transfer To SQL Database
FmPro Migrator
SQL Database
LiveCode Conversion
JSON Connection File Builder
JSON
JSON Connection File Builder.livecode
FM Functions Remapping
functions
FM Functions to LiveCode Mapping.livecode
FM Scripts Integration & Testing
Relationship Changes?
No
Yes
Relationships JSON Editor
relationships
Relationships JSON Editor (App)
Update Application Stack Relationships.JSON
SQL Column Changes?
No
Yes
SQL Column Types JSON Editor
Validation Testing [Iterate As Needed]
Update Application Stack SQLColumnTypes.JSON
SQL Types
SQLColumnTypes JSON Editor (App)
Build & Ship Standalone App

This flowchart provides an overview of the tools provided for the LiveCode conversion process and their usage during development. These tools are described in detail later in this manual.

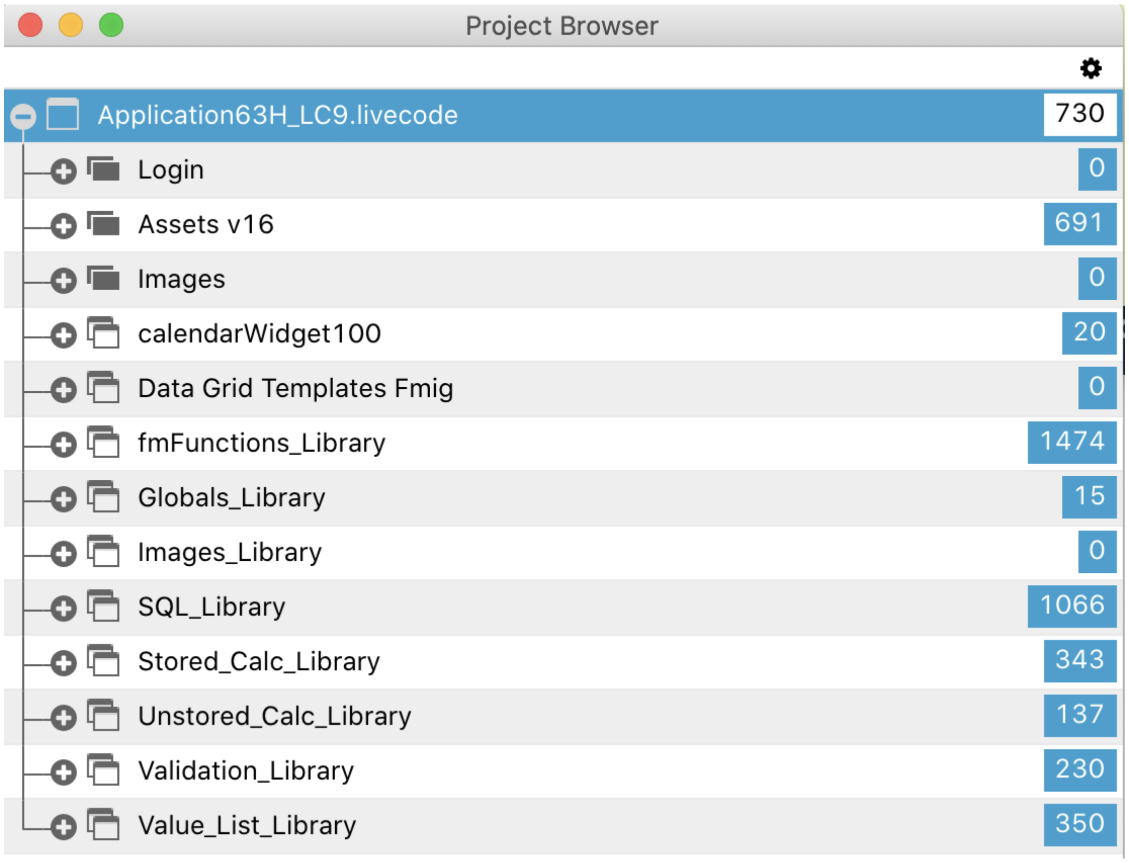
Conversion Process Output Files

Overview of Conversion Files

Overview

This chapter of the manual describes the various output files created by FmPro Migrator when performing a LiveCode conversion project.

Application.livecode Overview



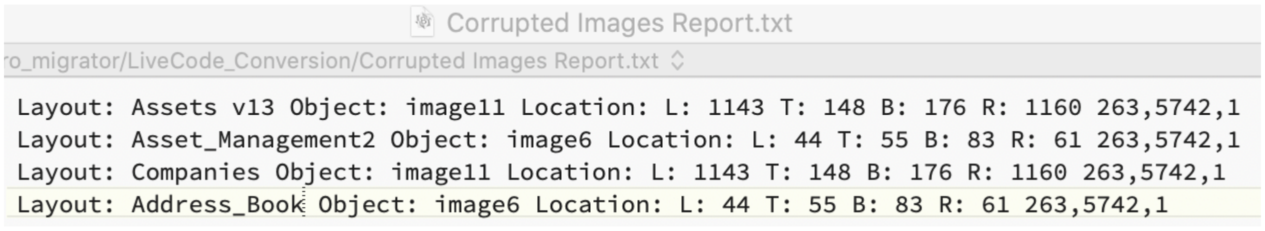
Application63H_LC9.livecode	730
Login	0
Assets v16	691
Images	0
calendarWidget100	20
Data Grid Templates Fmig	0
fmFunctions_Library	1474
Globals_Library	15
Images_Library	0
SQL_Library	1066
Stored_Calc_Library	343
Unstored_Calc_Library	137
Validation_Library	230
Value_List_Library	350

The main stack created by FmPro Migrator is the Application.livecode stack. All of the layouts/forms, Value Lists, relationships are placed into this stack during conversion. Scripts are also written to disk for further processing.

The utility stacks created by FmPro Migrator are available to help with further development of the scripts, calculations and relationships.

Regarding scripts, these should probably be added to new library stacks based upon the functionality of the application. For instance an application might have functionality including Invoices, Customers, Scheduling which could each be placed into separate library substacks.

Corrupted Images Report


Corrupted Images Report.txt
o_migrator/LiveCode_Conversion/Corrupted Images Report.txt

```
Layout: Assets v13 Object: image11 Location: L: 1143 T: 148 B: 176 R: 1160 263,5742,1
Layout: Asset_Management2 Object: image6 Location: L: 44 T: 55 B: 83 R: 61 263,5742,1
Layout: Companies Object: image11 Location: L: 1143 T: 148 B: 176 R: 1160 263,5742,1
Layout: Address_Book Object: image6 Location: L: 44 T: 55 B: 83 R: 61 263,5742,1
```

During the conversion of layouts to LiveCode cards, some image objects might not be readable within the imported layout XML code. If this problem occurs, the Corrupted Images Report.txt file will be created in the project directory. These errors will need to be fixed manually by importing an image object manually into the converted stack.

In order to facilitate making manual fixes to these missing images, each line of the report contains:

Layout: The name of the layout where the error occurred.
Object: The object name assigned to the object by FmPro Migrator when attempting to create the image object.
Location: The coordinates where the new object was supposed to be created.
Error - The last part of the line is the internal LiveCode error which occurred during the import paint command.

The LiveCode Error Lookup stack created by **Jacqueline Landman Gay** can be used to look up the LiveCode internal errors. It is available here:
http://livecodeshare.runrev.com/stack/712/LiveCode-Error-Lookup

For the first error shown as: 263,5742,1
263 - import: can't read file, mask file or display. Not much to be added here, it just means that the import paint command cannot read the contents of the file being imported. FmPro Migrator exports images into a temporary file, then uses the import paint command to import the image onto the card at the specified location.

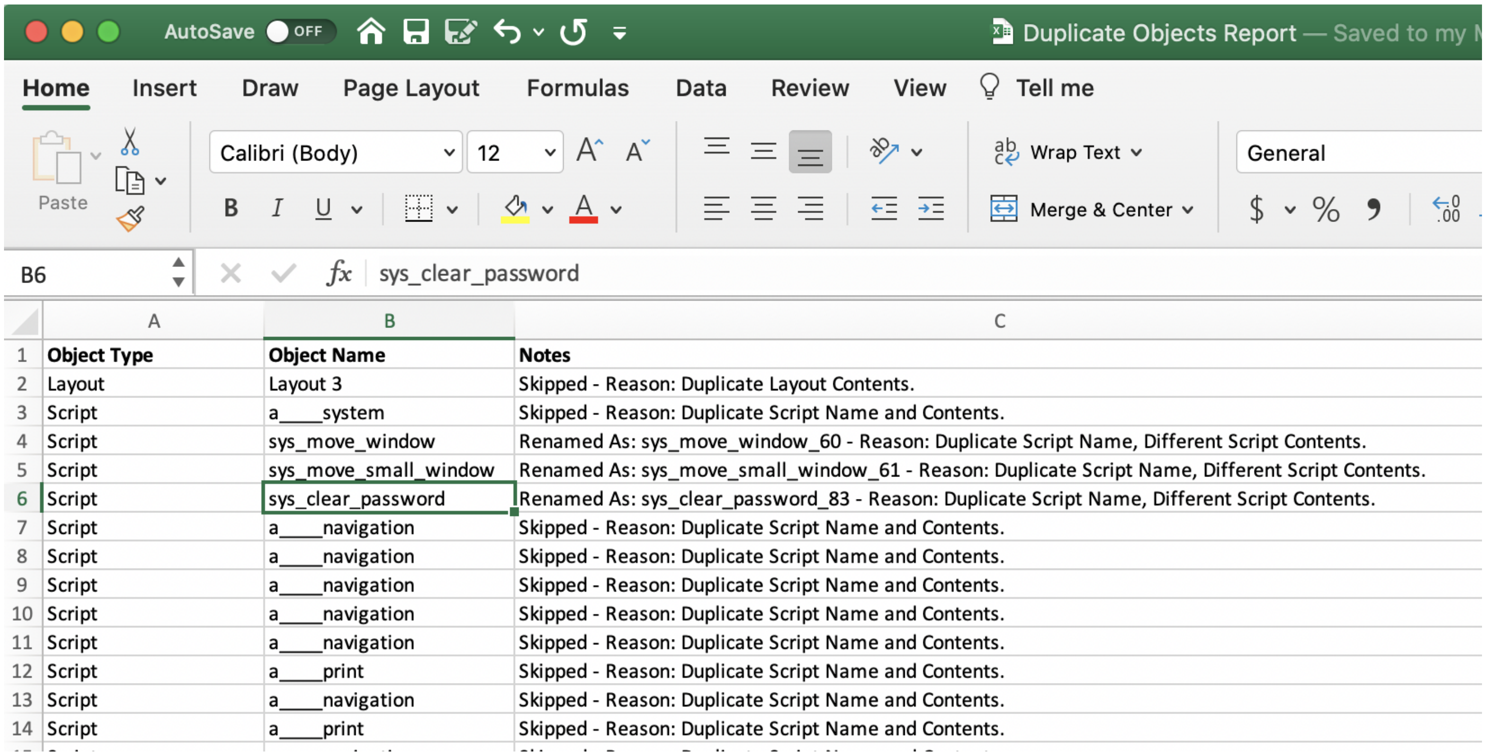
5742 - This is the line number of the script within FmPro Migrator where the import paint command was run. As a user, there is isn't much you can do with this info, but it could be helpful for .com Solutions Inc.

1 - Column number of the error.

Suggestion Text - This section of the error result is empty, because the LiveCode engine doesn't have any suggestions to offer. In general, it is possible that the image is really corrupted or possibly not able to be read and converted correctly within the source XML defining the layout.

Duplicate Objects Report.xls

Duplicate Objects Report Overview



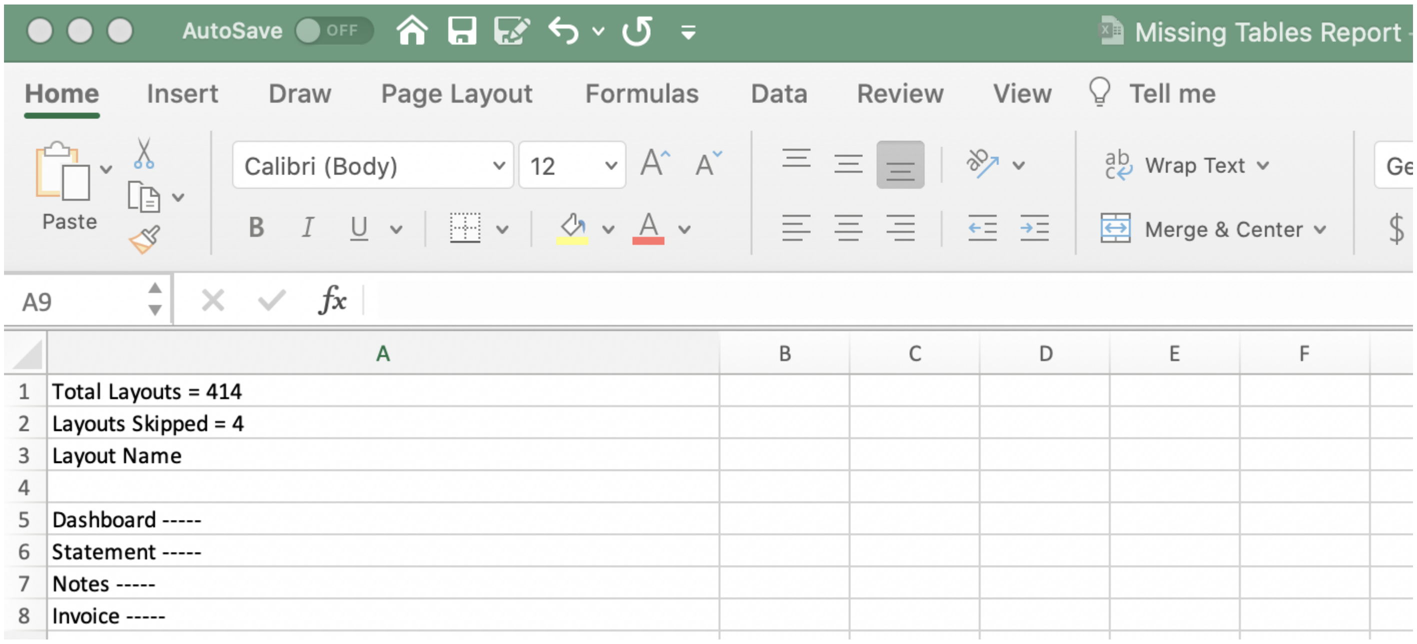
B6 sys_clear_password

	A	B	C
1	**Object Type**	**Object Name**	**Notes**
2	Layout	Layout 3	Skipped - Reason: Duplicate Layout Contents.
3	Script	a____system	Skipped - Reason: Duplicate Script Name and Contents.
4	Script	sys_move_window	Renamed As: sys_move_window_60 - Reason: Duplicate Script Name, Different Script Contents.
5	Script	sys_move_small_window	Renamed As: sys_move_small_window_61 - Reason: Duplicate Script Name, Different Script Contents.
6	Script	sys_clear_password	Renamed As: sys_clear_password_83 - Reason: Duplicate Script Name, Different Script Contents.
7	Script	a____navigation	Skipped - Reason: Duplicate Script Name and Contents.
8	Script	a____navigation	Skipped - Reason: Duplicate Script Name and Contents.
9	Script	a____navigation	Skipped - Reason: Duplicate Script Name and Contents.
10	Script	a____navigation	Skipped - Reason: Duplicate Script Name and Contents.
11	Script	a____navigation	Skipped - Reason: Duplicate Script Name and Contents.
12	Script	a____print	Skipped - Reason: Duplicate Script Name and Contents.
13	Script	a____navigation	Skipped - Reason: Duplicate Script Name and Contents.
14	Script	a____print	Skipped - Reason: Duplicate Script Name and Contents.

FmPro Migrator analyzes scripts, layouts and custom functions during conversion to determine if any duplicates will occur in the converted application. Duplicates can occur if multiple FileMaker databases are combined together for conversion. Achecksum is created for each of these objects and a comparison of the checksum and the name of the object is made in order to determine if the object should be skipped or renamed.

This report is saved as a TAB delimited file having an .XLS file extension so that Excel will immediately open it (with a warning that it isn't really an .XLS file).

Missing Tables Report Overview



	A	B	C	D	E	F
1	Total Layouts = 414					
2	Layouts Skipped = 4					
3	Layout Name					
4						
5	Dashboard -----					
6	Statement -----					
7	Notes -----					
8	Invoice -----					

The Missing Tables Report lists the layouts which could not be converted because they didn't have a table assigned to the layout. This problem will commonly occur with placeholder layouts used as group separators. So in many cases, nothing needs to be done - it is just an informative report.

<u>**Note**</u>: Since this report is also created as a TAB delimited file designed to be read by Excel, some layout names might not be displayed if they start with "--" characters. Since the file is a text file, it can also be opened with any text editor if Excel won't read the file properly.

create_relationships.sql

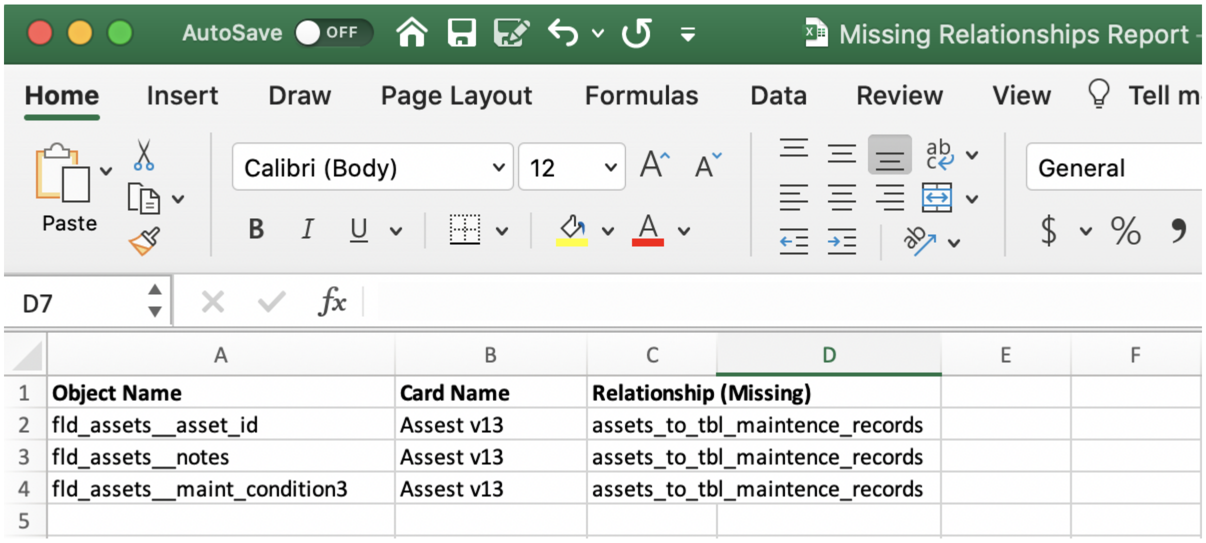
create_relationshps.sql Overview

```
-- -------------------- Relationship Creation SQL -------------------------
-- This output file was created by FmPro Migrator at 1:27:08 PM on Tuesday, October 20,
2020. By .com Solutions Inc. www.fmpromigrator.com
-- Source Database:
-- Destination Database:                 MySQL
-- Source DB Relationship Count:         1
-- Duplicate Relationships Skipped:      1
-- Non-EquiJoin Relationships Skipped:   0
-- Relationships Created:                0


-- [FM] tbl_Maintenance_Records->tbl_Assets => [SQL] tbl_Maintenance_Records->tbl_Assets
ALTER TABLE tbl_maintenance_records DROP INDEX tbl_maintenance_records_asset_id_idx;
CREATE INDEX tbl_maintenance_records_asset_id_idx ON tbl_maintenance_records(asset_id);
ALTER TABLE tbl_maintenance_records DROP FOREIGN KEY fk_tbl_maintenance_records_asset_id;
ALTER TABLE tbl_maintenance_records ADD CONSTRAINT fk_tbl_maintenance_records_asset_id
FOREIGN KEY (asset_id) REFERENCES tbl_assets(id) ON DELETE CASCADE;

-- [FM] tbl_Assets->tbl_Maintenance_Records => [SQL] tbl_Assets->tbl_Maintenance_Records
-- Skipped: Duplicate Relationship
```

During the conversion process, the list of relationships are read in order to create the create_relationships.sql file. This file should be run on the database server used by the application in order to create the table constraints.

Missing Relationships Report.xls

Missing Relationships Report Overview



	A	B	C	D	E	F
1	Object Name	Card Name	Relationship (Missing)			
2	fld_assets__asset_id	Assest v13	assets_to_tbl_maintence_records			
3	fld_assets__notes	Assest v13	assets_to_tbl_maintence_records			
4	fld_assets__maint_condition3	Assest v13	assets_to_tbl_maintence_records			
5						

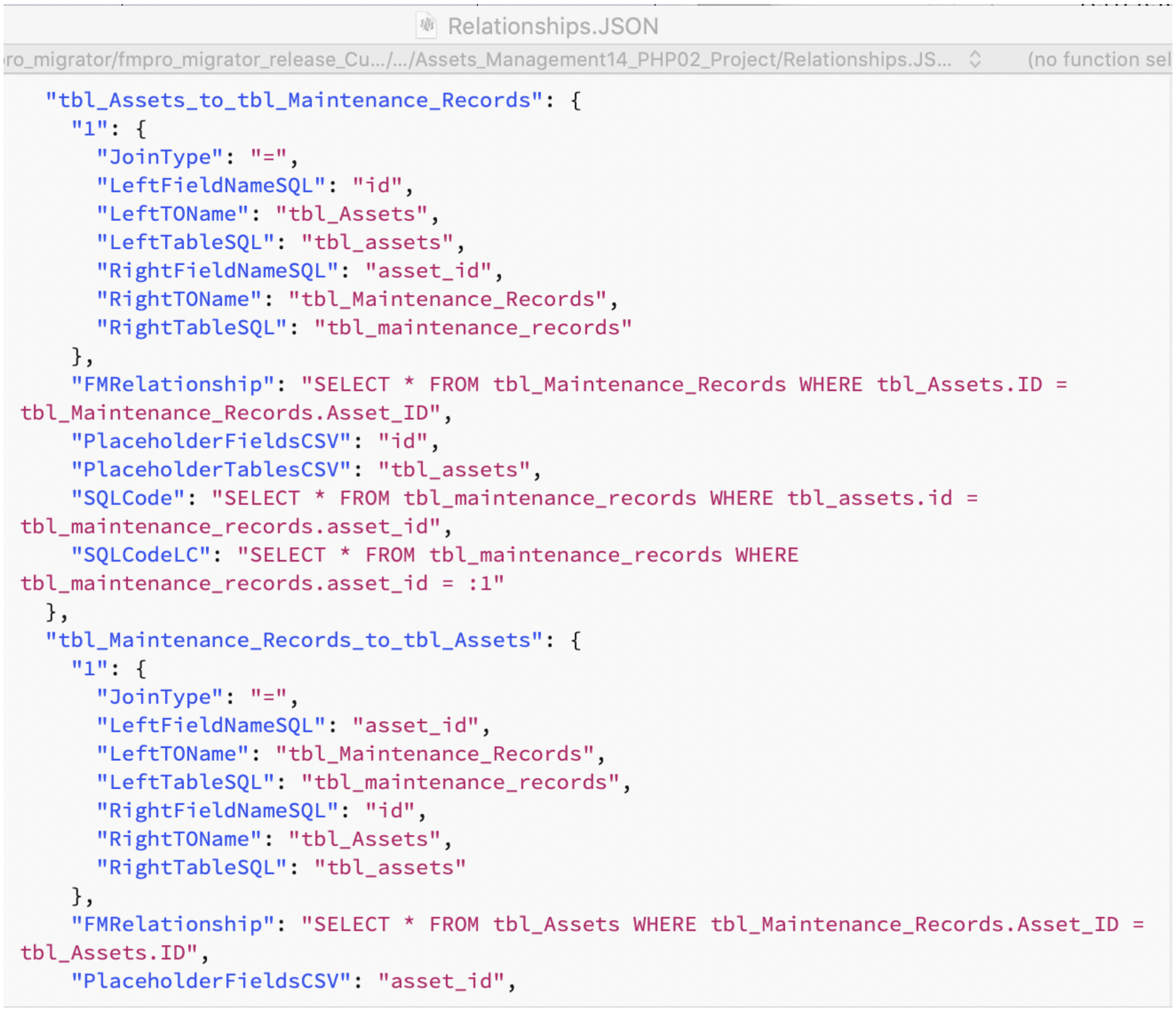
During the creation of each data bound field on a card, FmPro Migrator verifies that it can determine the base table required to read the data in the SQL database. If that test fails, the object is documented in the TAB delimited Missing Relationships Report.xls file.

FmPro Migrator is using the contents of the Relationships.JSON file stored as an array in memory in order to figure out the structure of each relationship within the new application.

If a relationship cannot be found, the findIndirectRelationships.livecode stack can be used to help troubleshoot the situation. Some FileMaker relationships may require being rebuilt manually to work with a SQL database. Manual changes made to the Relationships.JSON file need to be added back to the application stack as a custom property in order to be used by the application.

Relationships.JSON

Relationships.JSON Overview

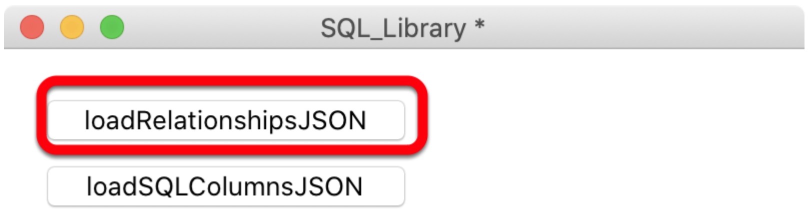

Relationships.JSON
ro_migrator/fmpro_migrator_release_Cu.../.../Assets_Management14_PHP02_Project/Relationships.JS...
(no function sel

```
  "tbl_Assets_to_tbl_Maintenance_Records": {
    "1": {
      "JoinType": "=",
      "LeftFieldNameSQL": "id",
      "LeftTOName": "tbl_Assets",
      "LeftTableSQL": "tbl_assets",
      "RightFieldNameSQL": "asset_id",
      "RightTOName": "tbl_Maintenance_Records",
      "RightTableSQL": "tbl_maintenance_records"
    },
    "FMRelationship": "SELECT * FROM tbl_Maintenance_Records WHERE tbl_Assets.ID =
tbl_Maintenance_Records.Asset_ID",
    "PlaceholderFieldsCSV": "id",
    "PlaceholderTablesCSV": "tbl_assets",
    "SQLCode": "SELECT * FROM tbl_maintenance_records WHERE tbl_assets.id =
tbl_maintenance_records.asset_id",
    "SQLCodeLC": "SELECT * FROM tbl_maintenance_records WHERE
tbl_maintenance_records.asset_id = :1"
  },
  "tbl_Maintenance_Records_to_tbl_Assets": {
    "1": {
      "JoinType": "=",
      "LeftFieldNameSQL": "asset_id",
      "LeftTOName": "tbl_Maintenance_Records",
      "LeftTableSQL": "tbl_maintenance_records",
      "RightFieldNameSQL": "id",
      "RightTOName": "tbl_Assets",
      "RightTableSQL": "tbl_assets"
    },
    "FMRelationship": "SELECT * FROM tbl_Assets WHERE tbl_Maintenance_Records.Asset_ID =
tbl_Assets.ID",
    "PlaceholderFieldsCSV": "asset_id",
```

FmPro Migrator generates the Relationships.JSON file from the relationships imported from within the original database. Since FileMaker relationships are bi-directional, two relationships are created for each imported relationship to represent this same functionality.

The contents of the Relationships.JSON file are automatically saved into the RelationshipsJSONProp custom property of the SQL_Library substack created by FmPro Migrator.

Importing the Relationships.JSON File



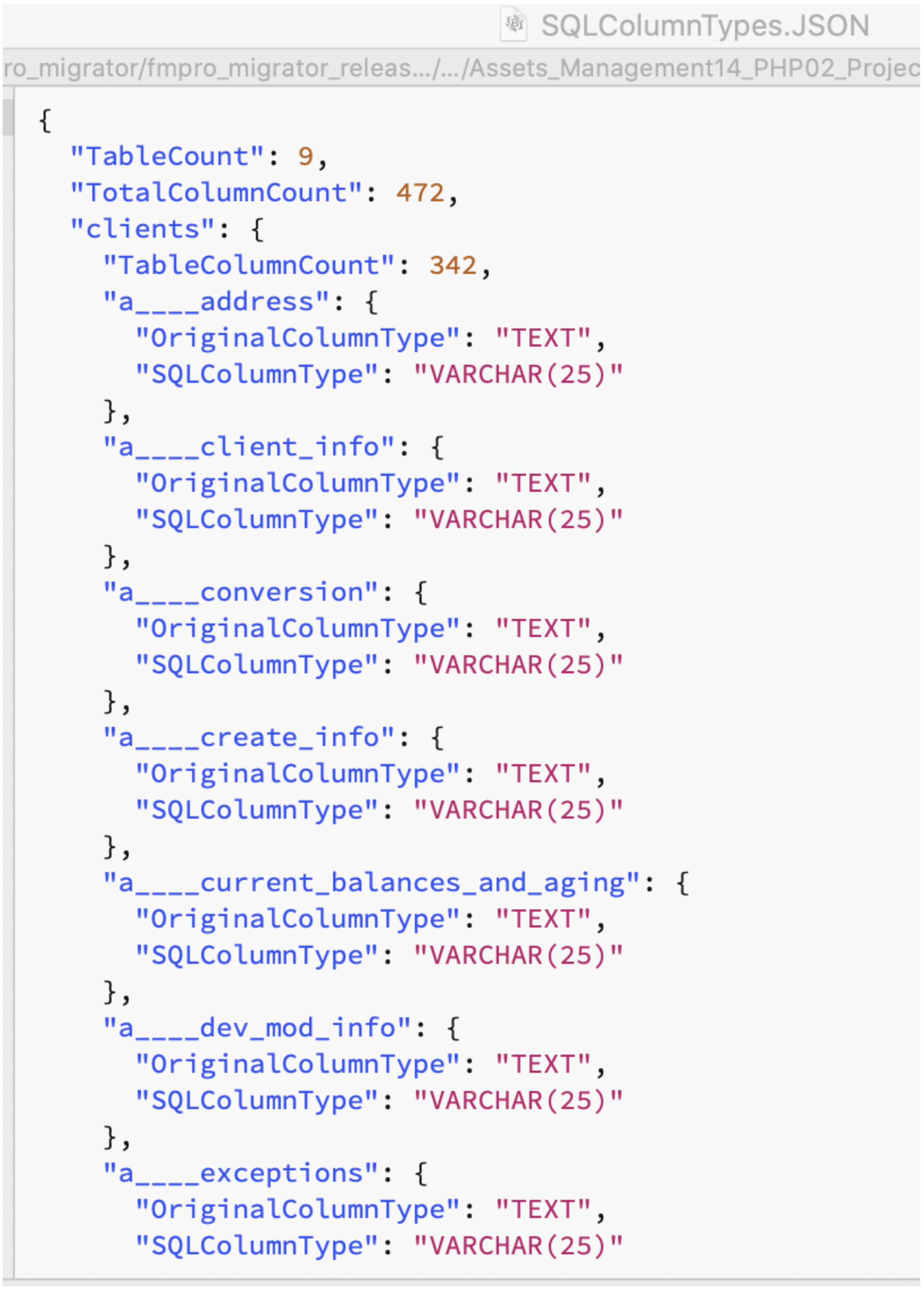
If the Relationships.JSON file is manually edited, it should be re-imported back into the SQ_Library by clicking the loadRelationshipsJSON button on the 1st card of the SQL_Library stack.

The Relationships.JSON data is converted into an array as soon as a successful connection is made to the SQL database server.

The Relationships.JSON file can be edited with the Relationships JSON Editor utility described in this manual.

SQLColumnTypes.JSON

SQLColumnTypes.JSON Overview


SQLColumnTypes.JSON

ro_migrator/fmpro_migrator_releas.../.../Assets_Management14_PHP02_Projec

```
{
  "TableCount": 9,
  "TotalColumnCount": 472,
  "clients": {
    "TableColumnCount": 342,
    "a____address": {
      "OriginalColumnType": "TEXT",
      "SQLColumnType": "VARCHAR(25)"
    },
    "a____client_info": {
      "OriginalColumnType": "TEXT",
      "SQLColumnType": "VARCHAR(25)"
    },
    "a____conversion": {
      "OriginalColumnType": "TEXT",
      "SQLColumnType": "VARCHAR(25)"
    },
    "a____create_info": {
      "OriginalColumnType": "TEXT",
      "SQLColumnType": "VARCHAR(25)"
    },
    "a____current_balances_and_aging": {
      "OriginalColumnType": "TEXT",
      "SQLColumnType": "VARCHAR(25)"
    },
    "a____dev_mod_info": {
      "OriginalColumnType": "TEXT",
      "SQLColumnType": "VARCHAR(25)"
    },
    "a____exceptions": {
      "OriginalColumnType": "TEXT",
      "SQLColumnType": "VARCHAR(25)"
```

The SQLColumTypes.JSON file is stored within the SQLColumnTypesJSONProp of the SQL_Library substack.
When a database connection has been made, this JSON data is converted into the

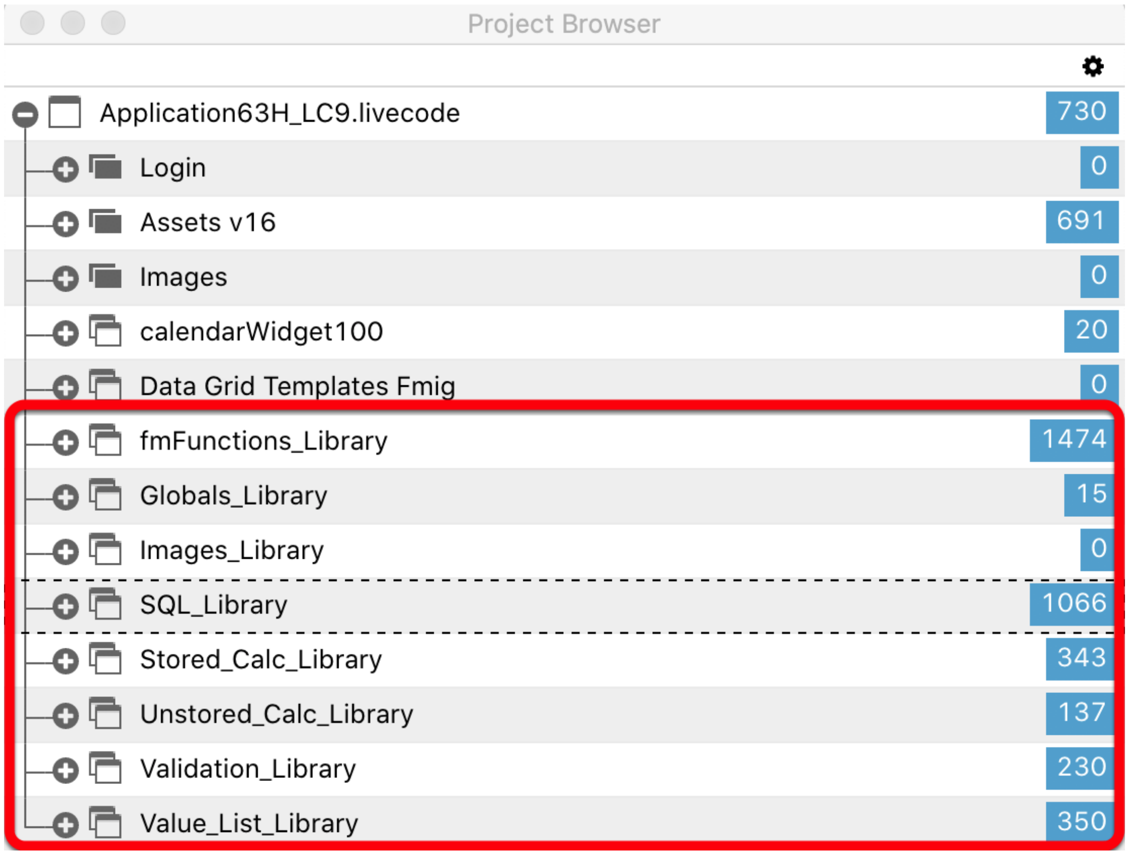
gSQLColumnTypesArray. The getBLOBColumnStatus() function in the SQL_Library uses this array to determine whether the SQL database column contains BLOB data when reading/writing data in the database.

This test is performed in order to avoid performing textEncode/textDecode on the data for BLOB columns.
The test is being done on the original database field type, looking for the type = "PICT" due to the consistency of always checking the FileMaker database type, since there are many database types used for various destination databases.

Library SubStacks

Overview of Library SubStacks

Library Substacks


Project Browser

Application63H_LC9.livecode	730
Login	0
Assets v16	691
Images	0
calendarWidget100	20
Data Grid Templates Fmig	0
fmFunctions_Library	1474
Globals_Library	15
Images_Library	0
SQL_Library	1066
Stored_Calc_Library	343
Unstored_Calc_Library	137
Validation_Library	230
Value_List_Library	350

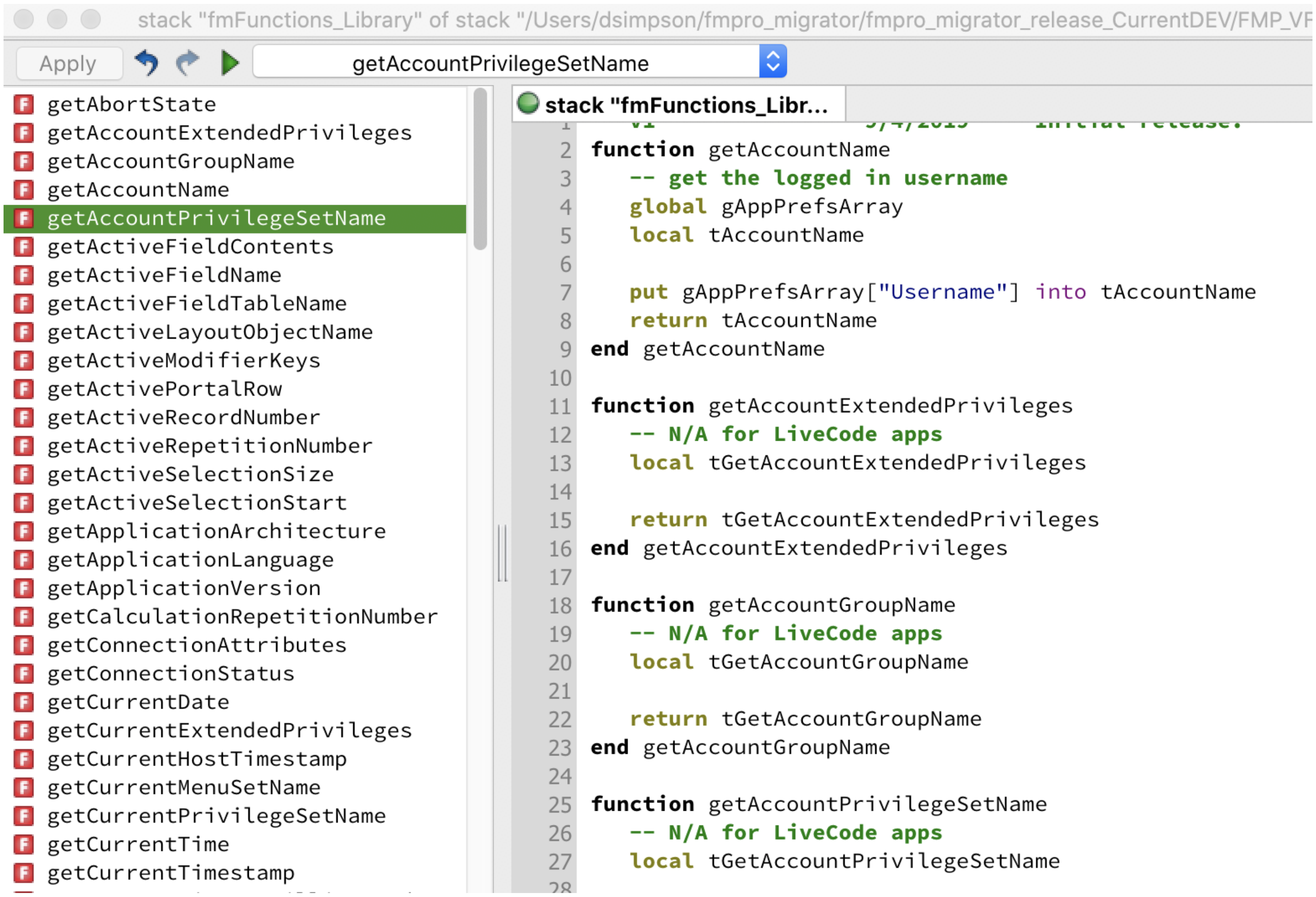
The stack generated by FmPro Migrator Platinum Edition includes 8 library substacks designed to implement the functionality of the original database application. The metadata from the source database is used to generate the code placed within these substacks. The following pages of this manual provide details about each library substack.

Library SubStack Initialization

The openStack handler of the converted stack contains "start using stack" commands for each of the library substacks. These commands insure that the handlers within the subStacks can be run from anywhere in the application.

```
on openStack
  global gStatusAreaVisible,gMode
  local tError
  initializeAppPrefs
  start using stack "SQL_Library"
  start using stack "Stored_Calc_Library"
  start using stack "Unstored_Calc_Library"
  start using stack "Value_List_Library"
  start using stack "Globals_Library"
  start using stack "fmFunctions_Library"
  start using stack "Validation_Library"

  if gMode is empty then
    put the result into tError
    put "Browse" into gMode
    put 1 into gStatusAreaVisible
    switchtoBrowseMode
  end if -- end of check for 1st launch
end openStack
```

fmFunctions_Library Overview


stack "fmFunctions_Library" of stack "/Users/dsimpson/fmpro_migrator/fmpro_migrator_release_CurrentDEV/FMP_VF
Apply
getAccountPrivilegeSetName
stack "fmFunctions_Libr...

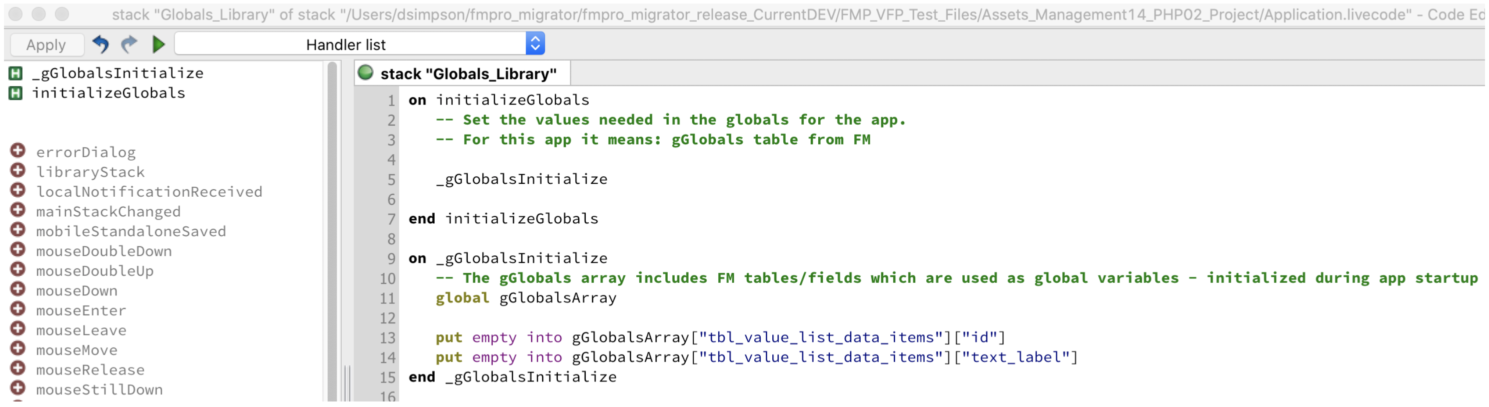
```
getAbortState
getAccountExtendedPrivileges
getAccountGroupName
getAccountName
getAccountPrivilegeSetName
getActiveFieldContents
getActiveFieldName
getActiveFieldTableName
getActiveLayoutObjectName
getActiveModifierKeys
getActivePortalRow
getActiveRecordNumber
getActiveRepetitionNumber
getActiveSelectionSize
getActiveSelectionStart
getApplicationArchitecture
getApplicationLanguage
getApplicationVersion
getCalculationRepetitionNumber
getConnectionAttributes
getConnectionStatus
getCurrentDate
getCurrentExtendedPrivileges
getCurrentHostTimestamp
getCurrentMenuSetName
getCurrentPrivilegeSetName
getCurrentTime
getCurrentTimestamp
```

```
function getAccountName
   -- get the logged in username
   global gAppPrefsArray
   local tAccountName

   put gAppPrefsArray["Username"] into tAccountName
   return tAccountName
end getAccountName

function getAccountExtendedPrivileges
   -- N/A for LiveCode apps
   local tGetAccountExtendedPrivileges

   return tGetAccountExtendedPrivileges
end getAccountExtendedPrivileges

function getAccountGroupName
   -- N/A for LiveCode apps
   local tGetAccountGroupName

   return tGetAccountGroupName
end getAccountGroupName

function getAccountPrivilegeSetName
   -- N/A for LiveCode apps
   local tGetAccountPrivilegeSetName
```

This library duplicates the functionality of 129 of the most commonly used FileMaker built-in functions. A tool is provided to remap the original FileMaker functions to these LiveCode versions within the converted FileMaker scripts. These LiveCode function handlers are named slightly different from the original FileMaker functions. Therefore the included FM Functions to LiveCode Mapping.livecode utility stack is used to rename the functions within the converted script and library files. During the conversion process, the contents of the stored/unstored calc library stacks are written to disk as text files (Unstored_Calc_Library.livecodescript, Stored_Calc_Library.livecodescript). This has been done to facilitate the conversion of the function names from FileMaker to LiveCode.

Globals_Library

Globals_Library Overview



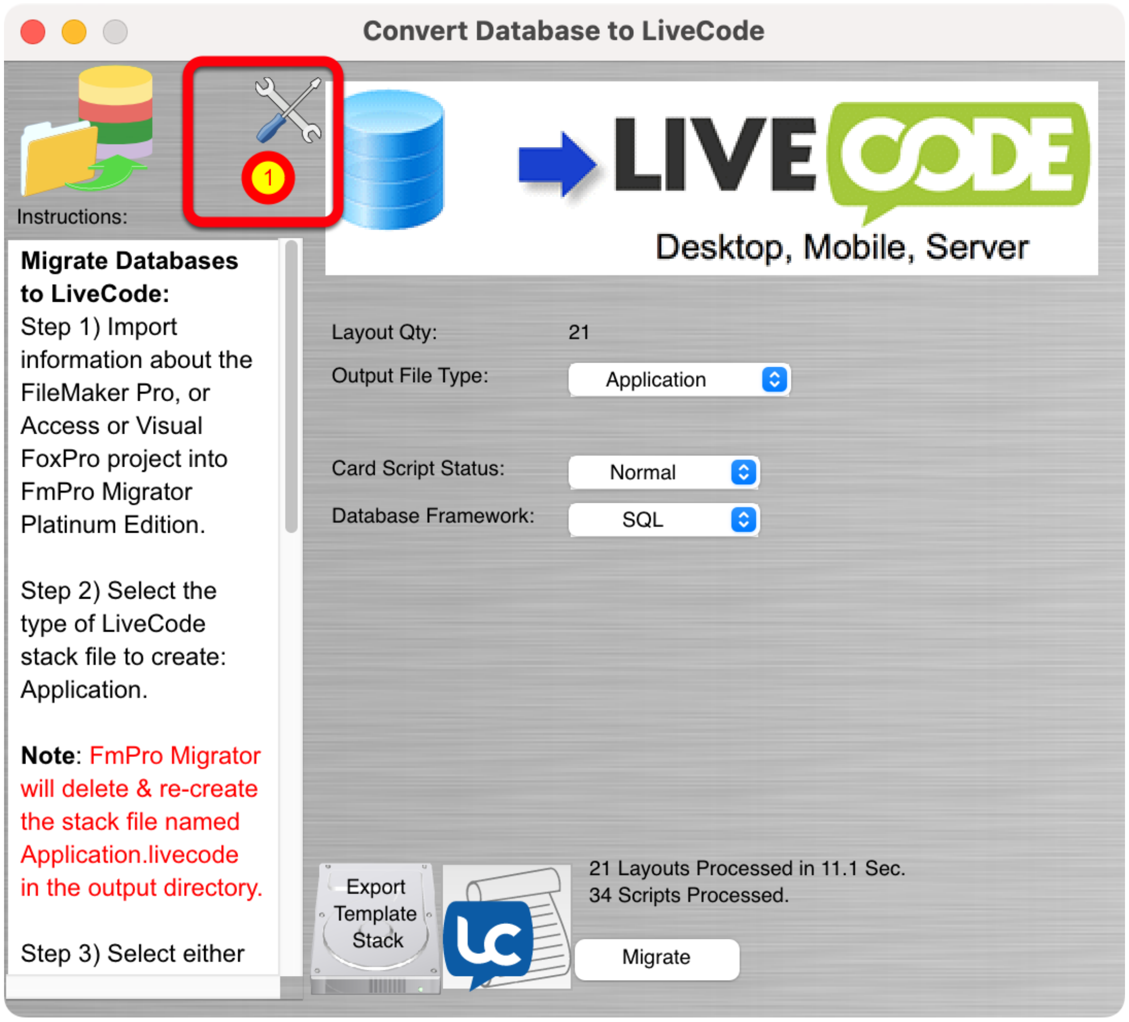
```
on initializeGlobals
   -- Set the values needed in the globals for the app.
   -- For this app it means: gGlobals table from FM

   _gGlobalsInitialize

end initializeGlobals

on _gGlobalsInitialize
   -- The gGlobals array includes FM tables/fields which are used as global variables - initialized during app startup
   global gGlobalsArray

   put empty into gGlobalsArray["tbl_value_list_data_items"]["id"]
   put empty into gGlobalsArray["tbl_value_list_data_items"]["text_label"]
end _gGlobalsInitialize
```

The gGlobals array within the _gGlobalsInitialize handler includes FM tables/fields which are used as global variables. These globals are initialized during app startup to specific known values and can be updated anytime by the application, just like FileMaker global fields.

In a FileMaker database, these fields could be part of an existing database table, mixed with other non-global fields.

Within the converted stack file, these globals are located within a separate array and need referenced separately instead of within existing table fields which might also contain data. This change should be made manually within the converted scripts of the converted stack.

Additional Preferences - LiveCode Conversion Window



Instructions:

Migrate Databases to LiveCode:
Step 1) Import information about the FileMaker Pro, or Access or Visual FoxPro project into FmPro Migrator Platinum Edition.

Step 2) Select the type of LiveCode stack file to create: Application.

Note: FmPro Migrator will delete & re-create the stack file named Application.livecode in the output directory.

Step 3) Select either

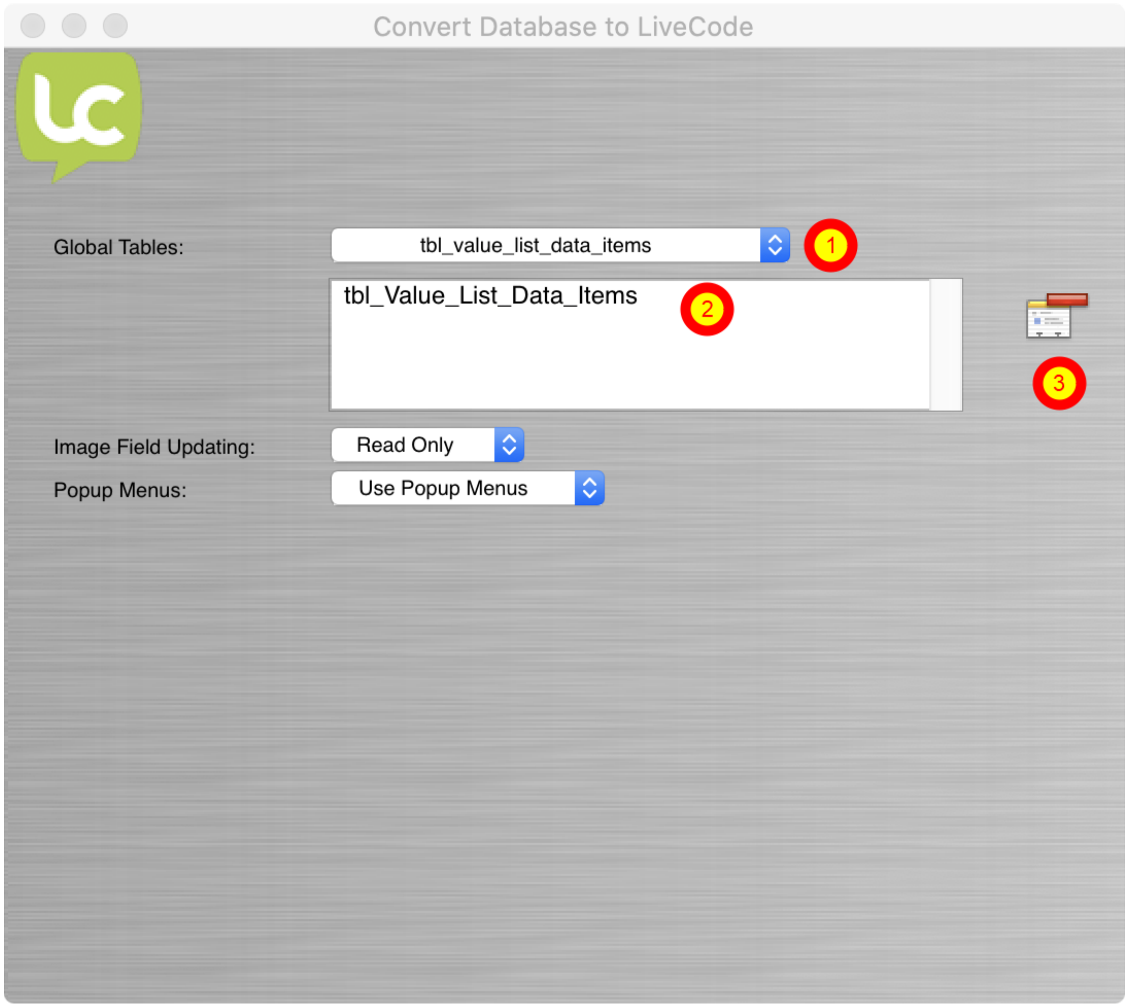
Layout Qty: 21
Output File Type: Application
Card Script Status: Normal
Database Framework: SQL

21 Layouts Processed in 11.1 Sec.
34 Scripts Processed.

Many FileMaker developers use 1 record tables containing only global fields during database development. FmPro Migrator provides a way to specify a global table during the conversion process. The fields of the selected table will be defined as a global array in the Globals_Library.

(1) Click the additional prefs button in the LiveCode Conversion window in FmPro Migrator.

Specifying Global Fields in FmPro Migrator



Global Tables: tbl_value_list_data_items

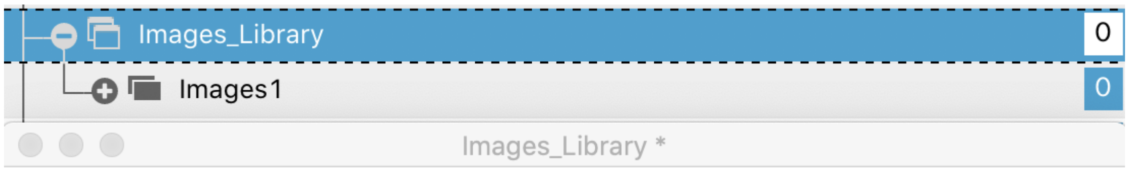
tbl_Value_List_Data_Items

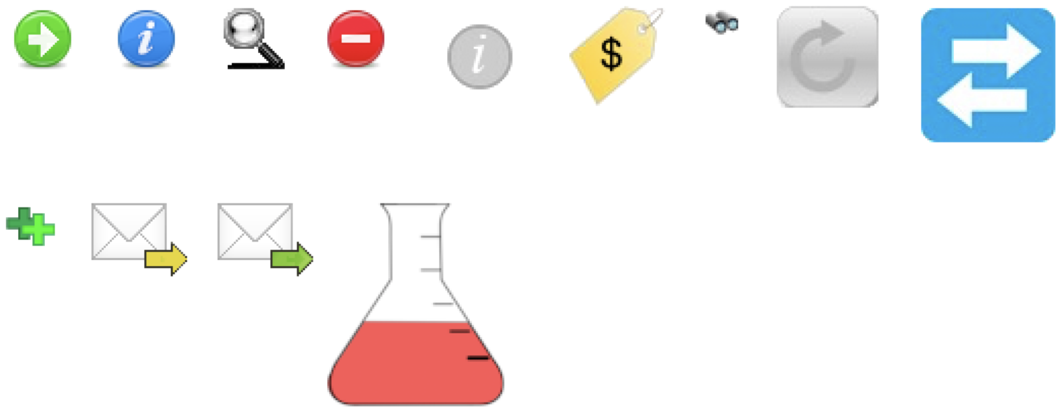
Image Field Updating: Read Only

Popup Menus: Use Popup Menus

Select any table in the solution - it gets added to the list of global tables as it is selected. Click to highlight any item and click the delete button (3) if you added a table by mistake.

Images_Library Overview

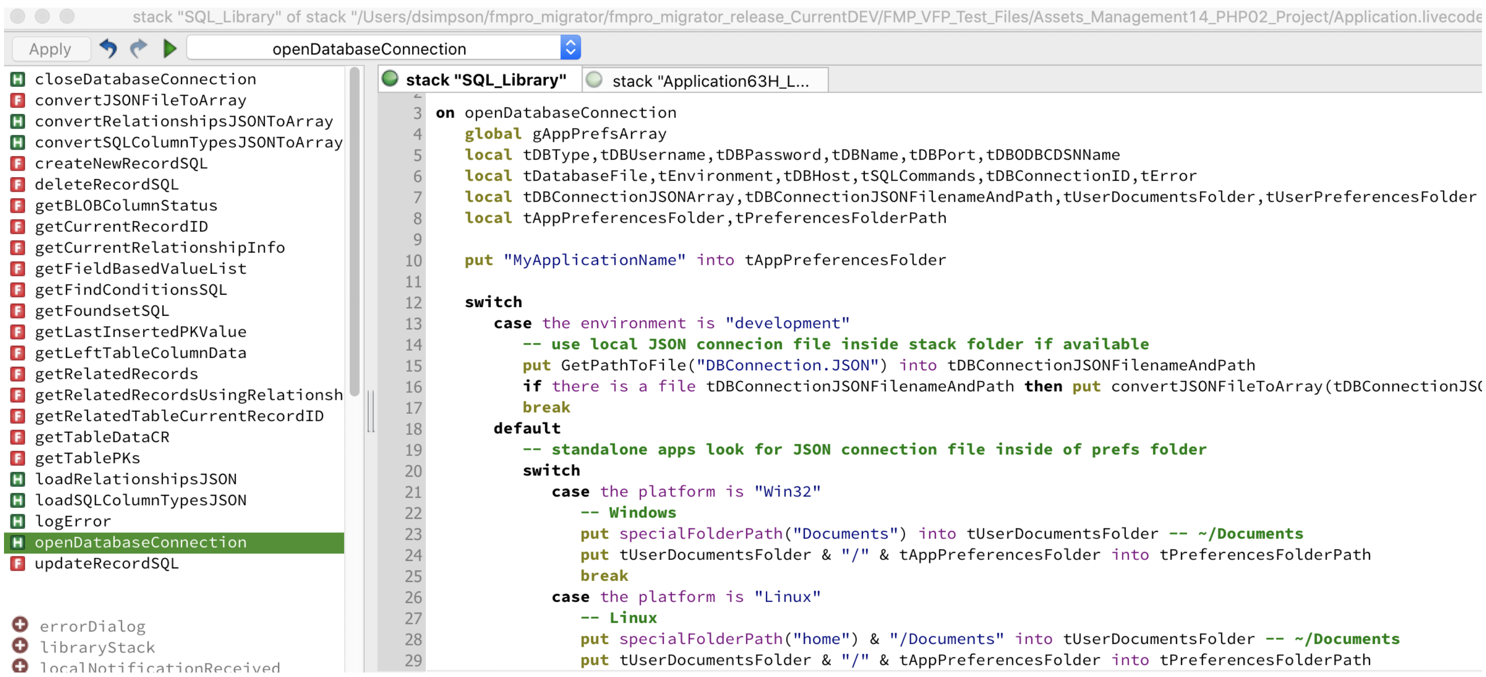




The Images_Library substack provides a convenient place where commonly used images can be centrally stored within the new application stack. This stack contains one card named Images1, which can be expanded to additional cards to support more images. Buttons on other cards of the stack can reference the ID# of images stored in this library. Replacing an image in this substack with another image having the same ID# will effectively replace it everywhere within the application without requiring manual updates on those other cards as long as the size is the same.

SQL_Library

SQL_Library Overview



stack "SQL_Library" of stack "/Users/dsimpson/fmpro_migrator/fmpro_migrator_release_CurrentDEV/FMP_VFP_Test_Files/Assets_Management14_PHP02_Project/Application.livecode

Apply openDatabaseConnection

stack "SQL_Library" stack "Application63H_L...

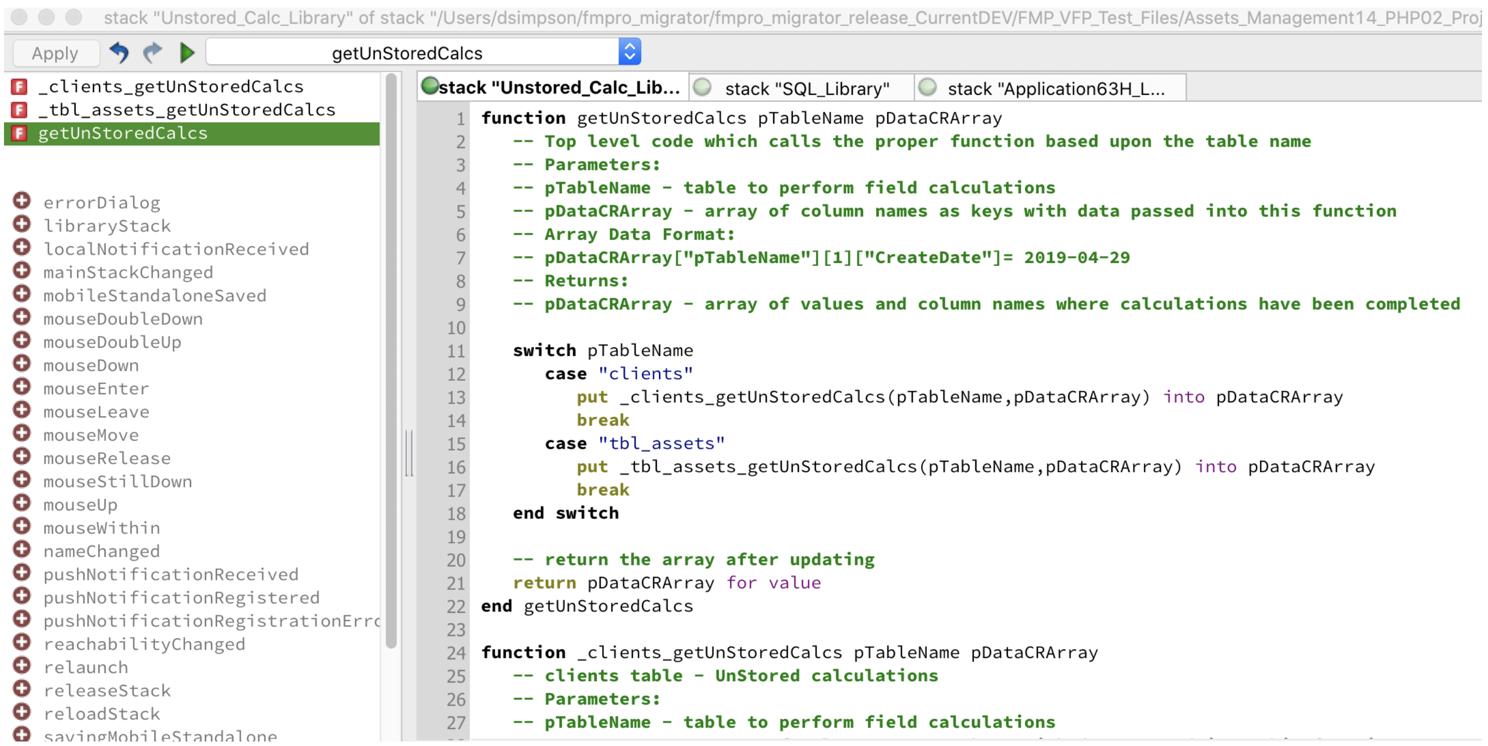
```
H closeDatabaseConnection
F convertJSONFileToArray
H convertRelationshipsJSONToArray
H convertSQLColumnTypesJSONToArray
F createNewRecordSQL
F deleteRecordSQL
F getBLOBColumnStatus
F getCurrentRecordID
F getCurrentRelationshipInfo
F getFieldBasedValueList
F getFindConditionsSQL
F getFoundsetSQL
F getLastInsertedPKValue
F getLeftTableColumnData
F getRelatedRecords
F getRelatedRecordsUsingRelationsh
F getRelatedTableCurrentRecordID
F getTableDataCR
F getTablePKs
H loadRelationshipsJSON
H loadSQLColumnTypesJSON
H logError
H openDatabaseConnection
F updateRecordSQL

errorDialog
libraryStack
localNotificationReceived
```

```
on openDatabaseConnection
   global gAppPrefsArray
   local tDBType,tDBUsername,tDBPassword,tDBName,tDBPort,tDBODBCDSNName
   local tDatabaseFile,tEnvironment,tDBHost,tSQLCommands,tDBConnectionID,tError
   local tDBConnectionJSONArray,tDBConnectionJSONFilenameAndPath,tUserDocumentsFolder,tUserPreferencesFolder
   local tAppPreferencesFolder,tPreferencesFolderPath

   put "MyApplicationName" into tAppPreferencesFolder

   switch
      case the environment is "development"
         -- use local JSON connecion file inside stack folder if available
         put GetPathToFile("DBConnection.JSON") into tDBConnectionJSONFilenameAndPath
         if there is a file tDBConnectionJSONFilenameAndPath then put convertJSONFileToArray(tDBConnectionJS
         break
      default
         -- standalone apps look for JSON connection file inside of prefs folder
         switch
            case the platform is "Win32"
               -- Windows
               put specialFolderPath("Documents") into tUserDocumentsFolder -- ~/Documents
               put tUserDocumentsFolder & "/" & tAppPreferencesFolder into tPreferencesFolderPath
               break
            case the platform is "Linux"
               -- Linux
               put specialFolderPath("home") & "/Documents" into tUserDocumentsFolder -- ~/Documents
               put tUserDocumentsFolder & "/" & tAppPreferencesFolder into tPreferencesFolderPath
```

The SQL_Library provides database independent code to connect to SQL database servers, insert, query, update, delete records, and load relationships from a JSON file.

The openDatabaseConnection handler shown here is checking for the existence of the DBConnection.JSON file containing database connection parameters. If found, the contents of this file are stored in the tDBConnectionJSONArray. When a user logs in, the contents of this array are overridden by any connection info manually entered by the user. This process provides the LiveCode developer with complete control over how much info the user needs to enter when connecting to the database server.

Unstored_Calc_Library Overview

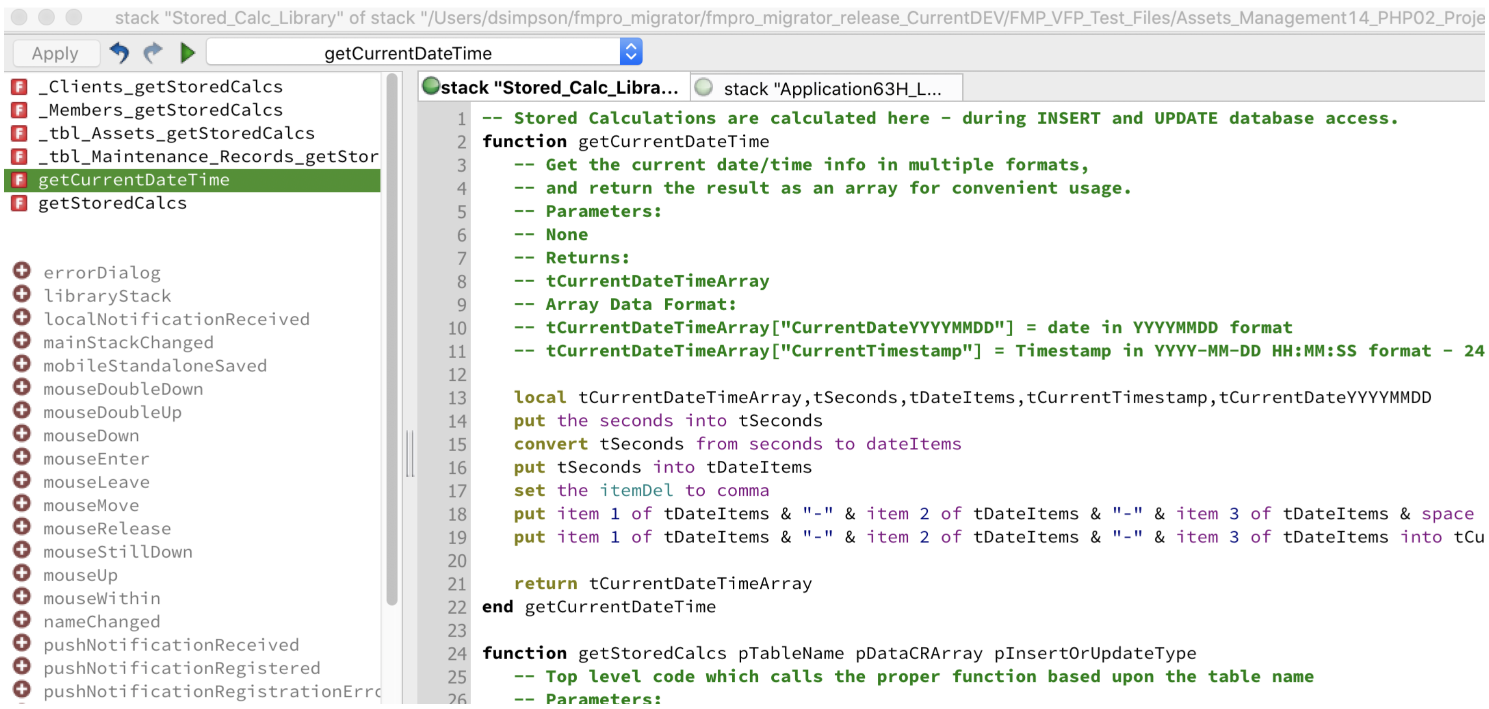

stack "Unstored_Calc_Library" of stack "/Users/dsimpson/fmpro_migrator/fmpro_migrator_release_CurrentDEV/FMP_VFP_Test_Files/Assets_Management14_PHP02_Proj
Apply
getUnStoredCalcs
_clients_getUnStoredCalcs
_tbl_assets_getUnStoredCalcs
getUnStoredCalcs
errorDialog
libraryStack
localNotificationReceived
mainStackChanged
mobileStandaloneSaved
mouseDoubleDown
mouseDoubleUp
mouseDown
mouseEnter
mouseLeave
mouseMove
mouseRelease
mouseStillDown
mouseUp
mouseWithin
nameChanged
pushNotificationReceived
pushNotificationRegistered
pushNotificationRegistrationErrc
reachabilityChanged
relaunch
releaseStack
reloadStack
savingMobileStandalone
stack "Unstored_Calc_Lib...
stack "SQL_Library"
stack "Application63H_L...

```
function getUnStoredCalcs pTableName pDataCRArray
   -- Top level code which calls the proper function based upon the table name
   -- Parameters:
   -- pTableName - table to perform field calculations
   -- pDataCRArray - array of column names as keys with data passed into this function
   -- Array Data Format:
   -- pDataCRArray["pTableName"][1]["CreateDate"]= 2019-04-29
   -- Returns:
   -- pDataCRArray - array of values and column names where calculations have been completed

   switch pTableName
      case "clients"
         put _clients_getUnStoredCalcs(pTableName,pDataCRArray) into pDataCRArray
         break
      case "tbl_assets"
         put _tbl_assets_getUnStoredCalcs(pTableName,pDataCRArray) into pDataCRArray
         break
   end switch

   -- return the array after updating
   return pDataCRArray for value
end getUnStoredCalcs

function _clients_getUnStoredCalcs pTableName pDataCRArray
   -- clients table - UnStored calculations
   -- Parameters:
   -- pTableName - table to perform field calculations
```

Stored field calculations are defined in this library for each table of the database from the original FileMaker database definition. Code which is too complex for automatic conversion is commented to prevent errors. This code runs as records are read from the database by the refreshFields handler on each card. By default, it is commented to improve performance, because these calcs do not necessarily need to run for every converted layout. The developer needs to decide whether it should run, then enable this code as needed.

This script for this library is written to disk as Unstored_Calc_Library.livecodescript in order to facilitate using the FM Functions to LiveCode Mapping.livecode utility to replace the FileMaker functions with the re-written LiveCode versions of these functions in the fmFunctions_Library.

Stored_Calc_Library Overview


stack "Stored_Calc_Library" of stack "/Users/dsimpson/fmpro_migrator/fmpro_migrator_release_CurrentDEV/FMP_VFP_Test_Files/Assets_Management14_PHP02_Proje

```
_Clients_getStoredCalcs
_Members_getStoredCalcs
_tbl_Assets_getStoredCalcs
_tbl_Maintenance_Records_getStor
getCurrentDateTime
getStoredCalcs

errorDialog
libraryStack
localNotificationReceived
mainStackChanged
mobileStandaloneSaved
mouseDoubleDown
mouseDoubleUp
mouseDown
mouseEnter
mouseLeave
mouseMove
mouseRelease
mouseStillDown
mouseUp
mouseWithin
nameChanged
pushNotificationReceived
pushNotificationRegistered
pushNotificationRegistrationErr
```

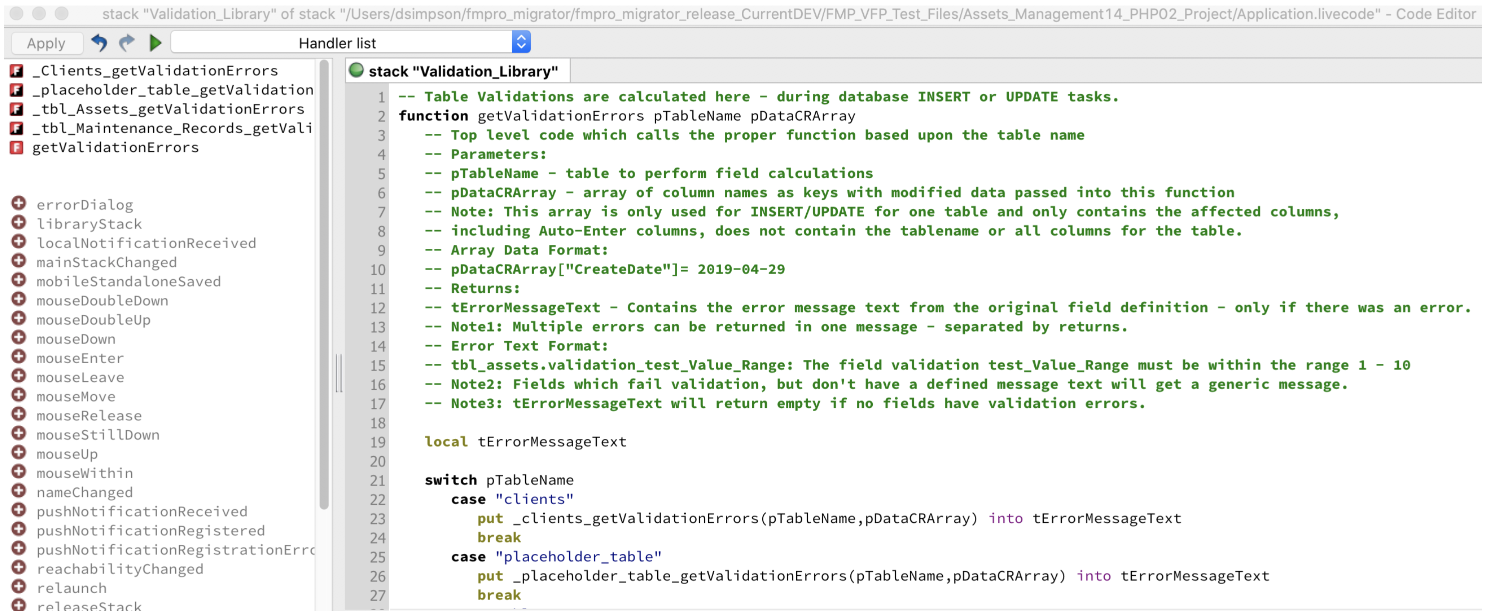
```
-- Stored Calculations are calculated here - during INSERT and UPDATE database access.
function getCurrentDateTime
   -- Get the current date/time info in multiple formats,
   -- and return the result as an array for convenient usage.
   -- Parameters:
   -- None
   -- Returns:
   -- tCurrentDateTimeArray
   -- Array Data Format:
   -- tCurrentDateTimeArray["CurrentDateYYYYMMDD"] = date in YYYYMMDD format
   -- tCurrentDateTimeArray["CurrentTimestamp"] = Timestamp in YYYY-MM-DD HH:MM:SS format - 24

   local tCurrentDateTimeArray,tSeconds,tDateItems,tCurrentTimestamp,tCurrentDateYYYYMMDD
   put the seconds into tSeconds
   convert tSeconds from seconds to dateItems
   put tSeconds into tDateItems
   set the itemDel to comma
   put item 1 of tDateItems & "-" & item 2 of tDateItems & "-" & item 3 of tDateItems & space
   put item 1 of tDateItems & "-" & item 2 of tDateItems & "-" & item 3 of tDateItems into tCu

   return tCurrentDateTimeArray
end getCurrentDateTime

function getStoredCalcs pTableName pDataCRArray pInsertOrUpdateType
   -- Top level code which calls the proper function based upon the table name
   -- Parameters:
```

The Stored_Calc_Library performs calculations for stored field calculations for each table of the database from the original FileMaker database definition. Code which is too complex for automatic conversion is commented to prevent errors. This code runs during record update/insert database tasks.

This script for this library is written to disk as Stored_Calc_Library.livecodescript in order to facilitate using the FM Functions to LiveCode Mapping.livecode utility to replace the FileMaker functions with the re-written LiveCode versions of these functions in the fmFunctions_Library.

Validation_Library

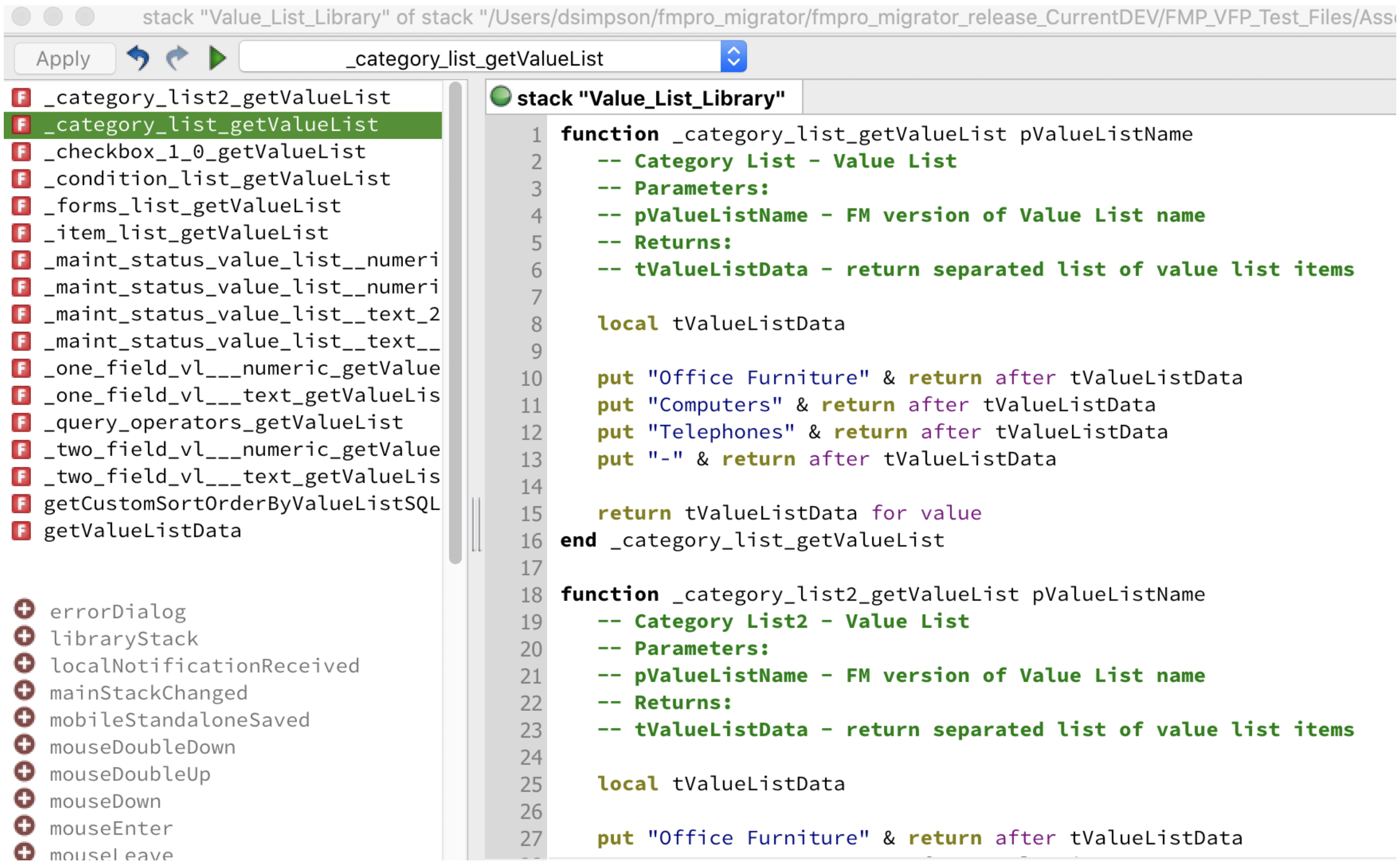
Validation_Library Overview


stack "Validation_Library" of stack "/Users/dsimpson/fmpro_migrator/fmpro_migrator_release_CurrentDEV/FMP_VFP_Test_Files/Assets_Management14_PHP02_Project/Application.livecode" - Code Editor
Apply
Handler list
_Clients_getValidationErrors
_placeholder_table_getValidation
_tbl_Assets_getValidationErrors
_tbl_Maintenance_Records_getVali
getValidationErrors
errorDialog
libraryStack
localNotificationReceived
mainStackChanged
mobileStandaloneSaved
mouseDoubleDown
mouseDoubleUp
mouseDown
mouseEnter
mouseLeave
mouseMove
mouseRelease
mouseStillDown
mouseUp
mouseWithin
nameChanged
pushNotificationReceived
pushNotificationRegistered
pushNotificationRegistrationErro
reachabilityChanged
relaunch
releaseStack
stack "Validation_Library"

```
-- Table Validations are calculated here - during database INSERT or UPDATE tasks.
function getValidationErrors pTableName pDataCRArray
   -- Top level code which calls the proper function based upon the table name
   -- Parameters:
   -- pTableName - table to perform field calculations
   -- pDataCRArray - array of column names as keys with modified data passed into this function
   -- Note: This array is only used for INSERT/UPDATE for one table and only contains the affected columns,
   -- including Auto-Enter columns, does not contain the tablename or all columns for the table.
   -- Array Data Format:
   -- pDataCRArray["CreateDate"]= 2019-04-29
   -- Returns:
   -- tErrorMessageText - Contains the error message text from the original field definition - only if there was an error.
   -- Note1: Multiple errors can be returned in one message - separated by returns.
   -- Error Text Format:
   -- tbl_assets.validation_test_Value_Range: The field validation test_Value_Range must be within the range 1 - 10
   -- Note2: Fields which fail validation, but don't have a defined message text will get a generic message.
   -- Note3: tErrorMessageText will return empty if no fields have validation errors.

   local tErrorMessageText

   switch pTableName
      case "clients"
         put _clients_getValidationErrors(pTableName,pDataCRArray) into tErrorMessageText
         break
      case "placeholder_table"
         put _placeholder_table_getValidationErrors(pTableName,pDataCRArray) into tErrorMessageText
         break
```

The Validation_Library performs field validation handlers during database insert/update tasks for each database table of the converted database.

Value_List_Library Overview



stack "Value_List_Library" of stack "/Users/dsimpson/fmpro_migrator/fmpro_migrator_release_CurrentDEV/FMP_VFP_Test_Files/Ass

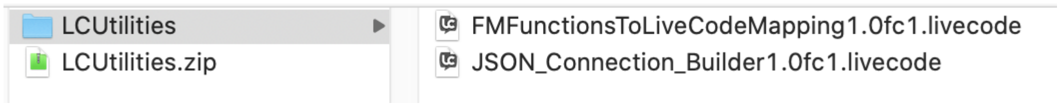
```
_category_list2_getValueList
_category_list_getValueList
_checkbox_1_0_getValueList
_condition_list_getValueList
_forms_list_getValueList
_item_list_getValueList
_maint_status_value_list__numeri
_maint_status_value_list__numeri
_maint_status_value_list__text_2
_maint_status_value_list__text__
_one_field_vl___numeric_getValue
_one_field_vl___text_getValueLis
_query_operators_getValueList
_two_field_vl___numeric_getValue
_two_field_vl___text_getValueLis
getCustomSortOrderByValueListSQL
getValueListData

errorDialog
libraryStack
localNotificationReceived
mainStackChanged
mobileStandaloneSaved
mouseDoubleDown
mouseDoubleUp
mouseDown
mouseEnter
mouseLeave
```

```
function _category_list_getValueList pValueListName
   -- Category List - Value List
   -- Parameters:
   -- pValueListName - FM version of Value List name
   -- Returns:
   -- tValueListData - return separated list of value list items

   local tValueListData

   put "Office Furniture" & return after tValueListData
   put "Computers" & return after tValueListData
   put "Telephones" & return after tValueListData
   put "-" & return after tValueListData

   return tValueListData for value
end _category_list_getValueList

function _category_list2_getValueList pValueListName
   -- Category List2 - Value List
   -- Parameters:
   -- pValueListName - FM version of Value List name
   -- Returns:
   -- tValueListData - return separated list of value list items

   local tValueListData

   put "Office Furniture" & return after tValueListData
```

Custom and field based value lists are defined within the Value_List_Library. Changes made here are reflected throughout the entire application. Value lists from Microsoft Access and Visual FoxPro are also defined inside this library.

Conversion Utilities

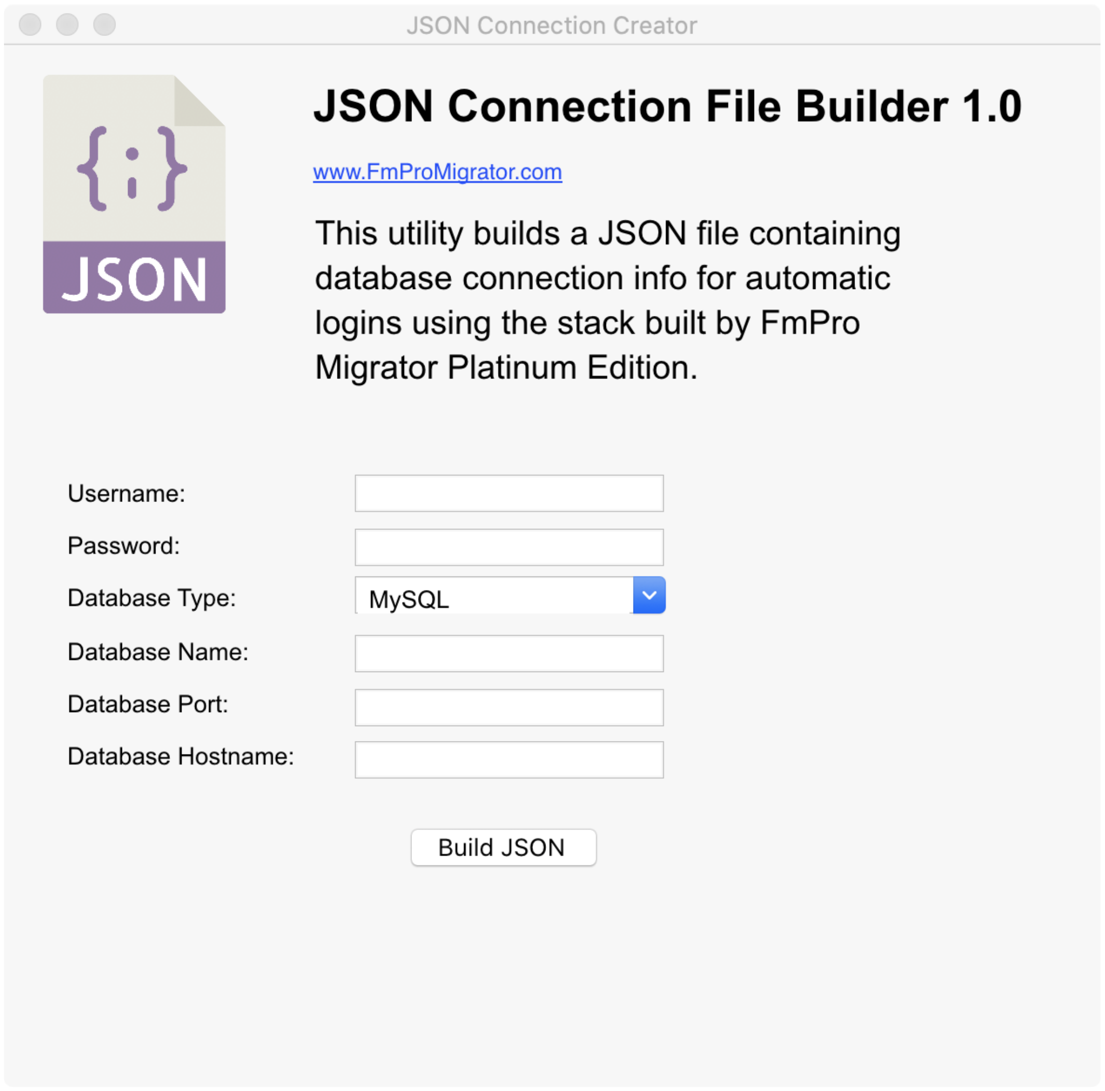
Opening JSON Connection File Builder



LCUtilities
LCUtilities.zip
FMFunctionsToLiveCodeMapping1.0fc1.livecode
JSON_Connection_Builder1.0fc1.livecode

When FmPro Migrator performs a conversion project, it creates the LCUtilities.zip file in the project folder where the converted stack is created. Extract the contents of this zip file and double-click the JSON_Connection_Builder.livecode stack to open it in the LiveCode IDE.

JSON Connection File Builder



JSON Connection Creator

JSON Connection File Builder 1.0

www.FmProMigrator.com

This utility builds a JSON file containing database connection info for automatic logins using the stack built by FmPro Migrator Platinum Edition.

Username:	
Password:	
Database Type:	MySQL
Database Name:	
Database Port:	
Database Hostname:	

The JSON Connection File Builder builds a database connection JSON file named DBConnection.JSON.
This JSON connection file has 2 main uses:

1) During development, you can quickly log the application under development into your test server without having to manually enter connection details.
2) For production usage, you can create connection files having the database type, name, port, hostname info pre-filled for users to log into the database server. Fields which are missing from the DBConnection.JSON file will be filled in by the user during login.

Password Encryption - JSON Connection File Builder

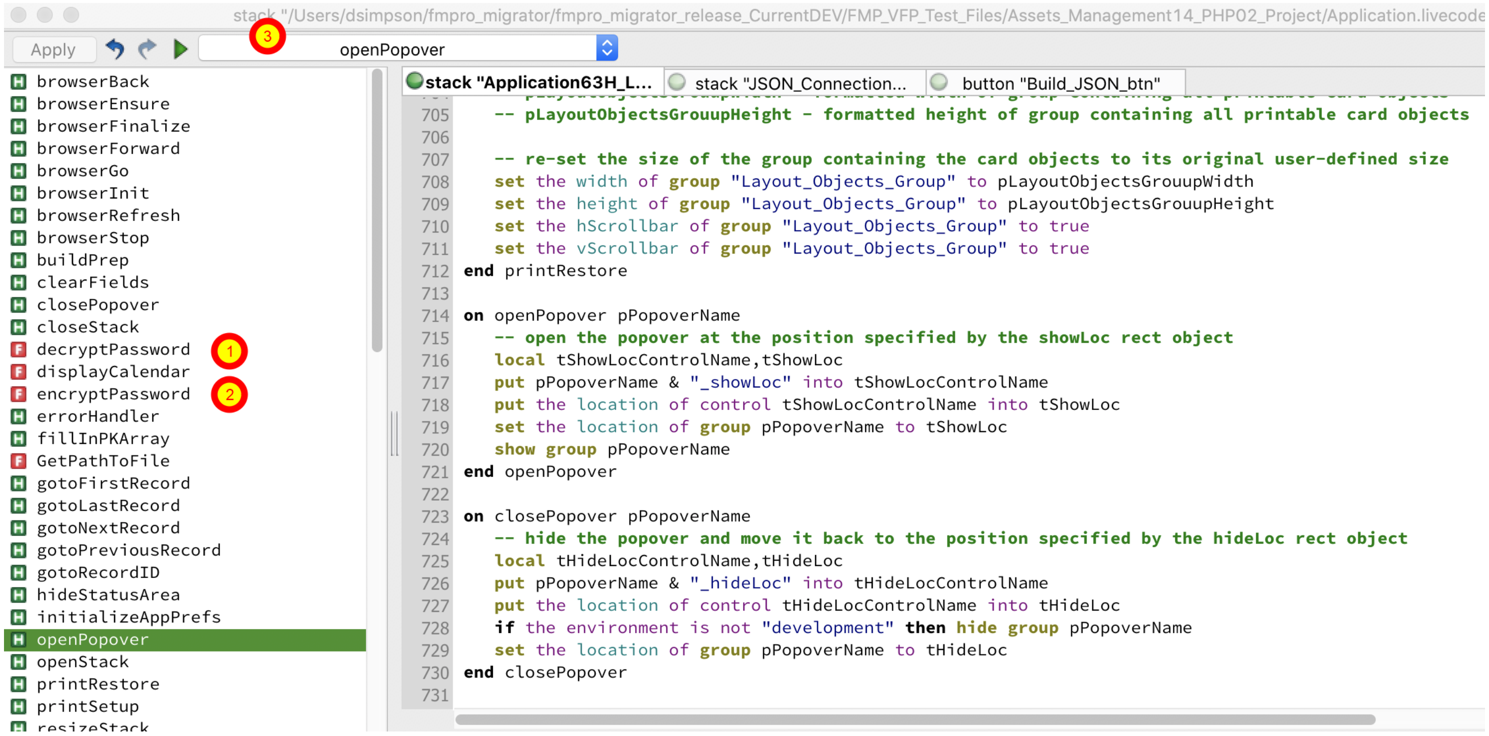
```
function decryptPassword pPasswordEncrypted
   -- decrypt a single password
   local tPasswordCleartext

   decrypt pPasswordEncrypted using "aes-256-cbc" with password "[redacted]"
   put it into tPasswordCleartext
   return tPasswordCleartext
end decryptPassword

function encryptPassword pPasswordCleartext
   -- encrypt a single password
   local tPasswordEncrypted

   encrypt pPasswordCleartext using "aes-256-cbc" with password "[redacted]"
   put it into tPasswordEncrypted
   return tPasswordEncrypted
end encryptPassword
```

The JSON Connection File Builder utility encrypts the contents of the password field before writing the JSON file. The encryptPassword function in the stack performs this task. These two handlers correspond to identical handlers located in the application stack.

Note: It is recommended that you change the encryption/decryption password in both locations before distributing your production application.

Usage Ideas: It is likely that this JSON Connection File Builder utility will be used during development to simplify the login process during testing. But it is hoped that by distributing this stack and its source code, it could give developers ideas for turning it into a more extensive utility for production use. For instance, a more extensive tool could be developed by LiveCode developers to automate the on-boarding process for new customers.

Password Encryption/Decryption - Application Stack


stack "/Users/dsimpson/fmpro_migrator/fmpro_migrator_release_CurrentDEV/FMP_VFP_Test_Files/Assets_Management14_PHP02_Project/Application.livecode
Apply
3
openPopover
stack "Application63H_L...
stack "JSON_Connection...
button "Build_JSON_btn"
1
2

```
browserBack
browserEnsure
browserFinalize
browserForward
browserGo
browserInit
browserRefresh
browserStop
buildPrep
clearFields
closePopover
closeStack
decryptPassword
displayCalendar
encryptPassword
errorHandler
fillInPKArray
GetPathToFile
gotoFirstRecord
gotoLastRecord
gotoNextRecord
gotoPreviousRecord
gotoRecordID
hideStatusArea
initializeAppPrefs
openPopover
openStack
printRestore
printSetup
resizeStack
```

```
   -- pLayoutObjectsGrouupHeight - formatted height of group containing all printable card objects

   -- re-set the size of the group containing the card objects to its original user-defined size
   set the width of group "Layout_Objects_Group" to pLayoutObjectsGrouupWidth
   set the height of group "Layout_Objects_Group" to pLayoutObjectsGrouupHeight
   set the hScrollbar of group "Layout_Objects_Group" to true
   set the vScrollbar of group "Layout_Objects_Group" to true
end printRestore

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
```

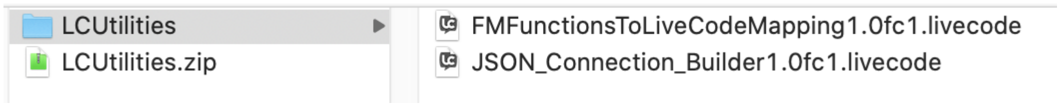
The encryptPassword/decryptPassword handlers are located at the top-level of the Application.livecode stack as shown in the screenshot above.

DBConnection.JSON File Creation

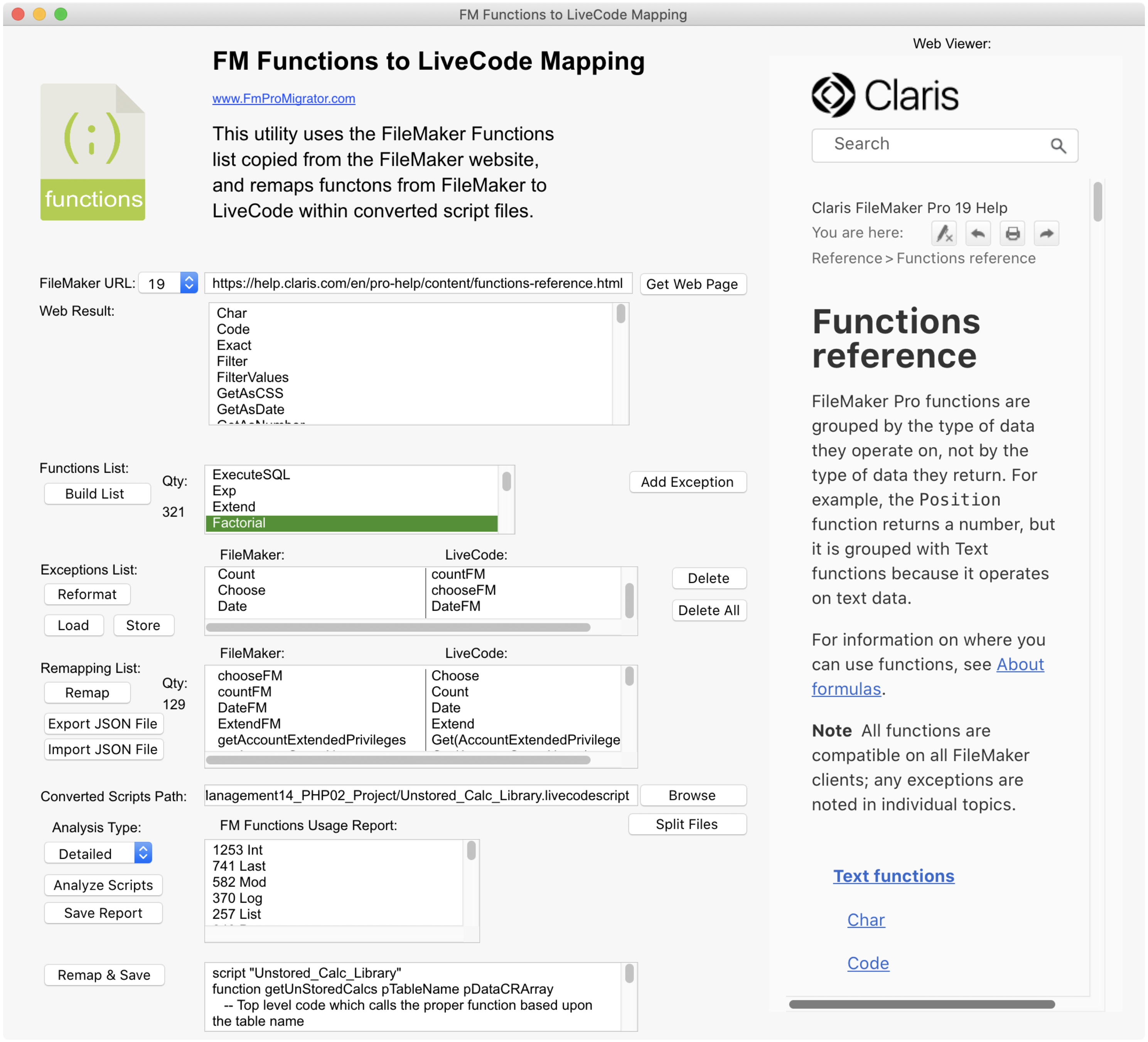
```
on mouseUp
   -- build a JSON file using the fields on this card.
   -- Output: DBConnection.JSON located in the same directory as this stack

   local tDBConnectionJSONFilenameAndPath,tDBConnectionJSONArray,tDBConnectionJSONData
   local tDBType,tDBUsername,tDBPassword,tDBName,tDBHost,tDBPort,tJSONFilename

   put "DBConnection.JSON" into tJSONFilename

   -- gather info from fields
   put field "Username_fld" of card "configuration" into tDBUsername
   put field "Password_fld" of card "configuration" into tDBPassword
   put the label of button "DBType_menu" of card "configuration" into tDBType
   put field "Database_Name_fld" of card "configuration" into tDBName
   put field "Database_Port_fld" of card "configuration" into tDBPort
   put field "Database_Hostname_fld" of card "configuration" into tDBHost

   -- assemble info into array
   put tDBType into tDBConnectionJSONArray["DBType"]
   put tDBUsername into tDBConnectionJSONArray["DBUsername"]
   put encryptPassword(tDBPassword) into tDBConnectionJSONArray["DBPassword"]
   put tDBName into tDBConnectionJSONArray["DBName"]
   put tDBPort into tDBConnectionJSONArray["DBPort"]
   put tDBHost into tDBConnectionJSONArray["DBHost"]

   -- convert array into JSON data
   put ArrayToJSON(tDBConnectionJSONArray,,1) into tDBConnectionJSONData
```

The code which creates the DBConnection.JSON file is straightforward. The contents of the fields are gathered into variables, encrypted (if necessary) and placed into an array. The array is converted into JSON text and written to disk in the location specified by the user.

Opening the FM Functions to LiveCode Remapping Utility


LCUtilities
LCUtilities.zip
FMFunctionsToLiveCodeMapping1.0fc1.livecode
JSON_Connection_Builder1.0fc1.livecode

When FmPro Migrator performs a conversion project, it creates the LCUtilities.zip file in the project folder where the converted stack is created. Extract the contents of this zip file and double-click the FMFunctionsToLiveCodeMapping.livecode stack to open it in the LiveCode IDE.



FM Functions to LiveCode Mapping

FM Functions to LiveCode Mapping

www.FmProMigrator.com

This utility uses the FileMaker Functions list copied from the FileMaker website, and remaps functons from FileMaker to LiveCode within converted script files.

FileMaker URL: 19 https://help.claris.com/en/pro-help/content/functions-reference.html [Get Web Page]

Web Result:

```
Char
Code
Exact
Filter
FilterValues
GetAsCSS
GetAsDate
```

Functions List: [Build List] Qty: 321

```
ExecuteSQL
Exp
Extend
Factorial
```

[Add Exception]

Exceptions List: [Reformat] [Load] [Store] [Delete] [Delete All]

FileMaker:	LiveCode:
Count	countFM
Choose	chooseFM
Date	DateFM

Remapping List: [Remap] Qty: 129 [Export JSON File] [Import JSON File]

FileMaker:	LiveCode:
chooseFM	Choose
countFM	Count
DateFM	Date
ExtendFM	Extend
getAccountExtendedPrivileges	Get(AccountExtendedPrivilege

Converted Scripts Path: lanagement14_PHP02_Project/Unstored_Calc_Library.livecodescript [Browse]

Analysis Type: Detailed [Analyze Scripts] [Save Report]

FM Functions Usage Report: [Split Files]

```
1253 Int
741 Last
582 Mod
370 Log
257 List
```

[Remap & Save]

```
script "Unstored_Calc_Library"
function getUnStoredCalcs pTableName pDataCRArray
   -- Top level code which calls the proper function based upon
the table name
```

Web Viewer:

Claris

Search

Claris FileMaker Pro 19 Help

You are here:

Reference > Functions reference

Functions reference

FileMaker Pro functions are grouped by the type of data they operate on, not by the type of data they return. For example, the Position function returns a number, but it is grouped with Text functions because it operates on text data.

For information on where you can use functions, see About formulas.

Note All functions are compatible on all FileMaker clients; any exceptions are noted in individual topics.

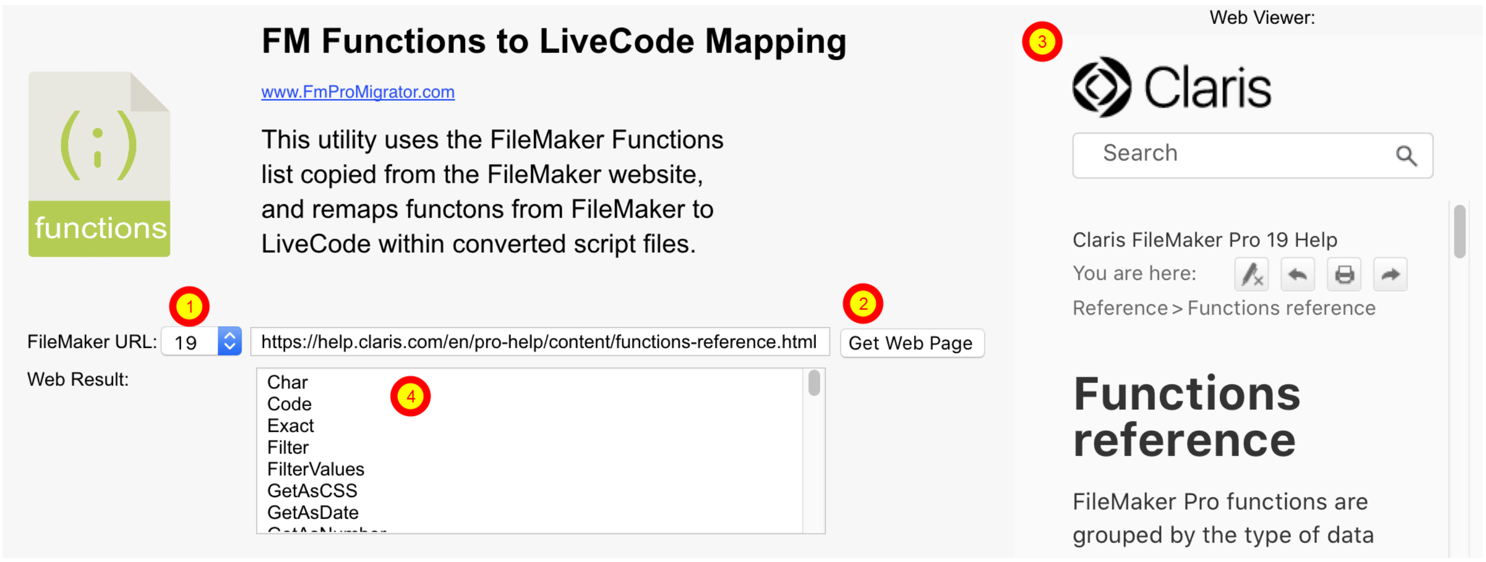
Text functions

Char

Code

The FMFunctionsToLiveCodeMapping.livecode stack is provided to automate the conversion of FileMaker Pro functions into equivalent LiveCode functions located within the fmFunctions_Library substack. This utility is provided as an unlocked stack so that the developer can make changes if necessary.

Getting the List of FileMaker Functions


FM Functions to LiveCode Mapping

www.FmProMigrator.com

This utility uses the FileMaker Functions list copied from the FileMaker website, and remaps functons from FileMaker to LiveCode within converted script files.

FileMaker URL: 19 https://help.claris.com/en/pro-help/content/functions-reference.html

Web Result:

Char
Code
Exact
Filter
FilterValues
GetAsCSS
GetAsDate

Web Viewer:

Claris FileMaker Pro 19 Help
You are here:
Reference > Functions reference

Functions reference

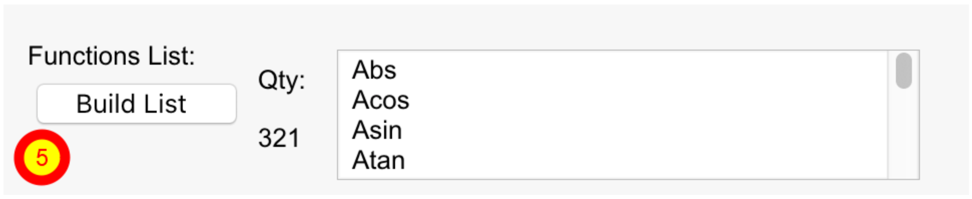
FileMaker Pro functions are grouped by the type of data

This utility already contains a list of the latest FileMaker Pro functions stored within the stack, so you might not need to update the list, and could skip this section.

But if you wanted to update the functions list, (1) select the FileMaker version menu which will populate the FileMaker URL field with the proper web page. (2) Click the Get Web Page button. The LiveCode browser widget (3) will display the contents of the specified URL. Once the page has loaded into the browser widget, a callback message will be sent to retrieve the text from the browser and populate the text of the functions into the (4) Web Result field.

<u>**Note:**</u> FileMaker versions 16 - 18 at this time seem to return the same list of functions. This appears to be due to all of the functions being available on the webpage but unneeded functions are omitted from being viewed if they don't apply to the selected FileMaker version. However this automated process retrieves all of the text of the webpage, so all functions are seen.

Building the Functions List


Functions List:

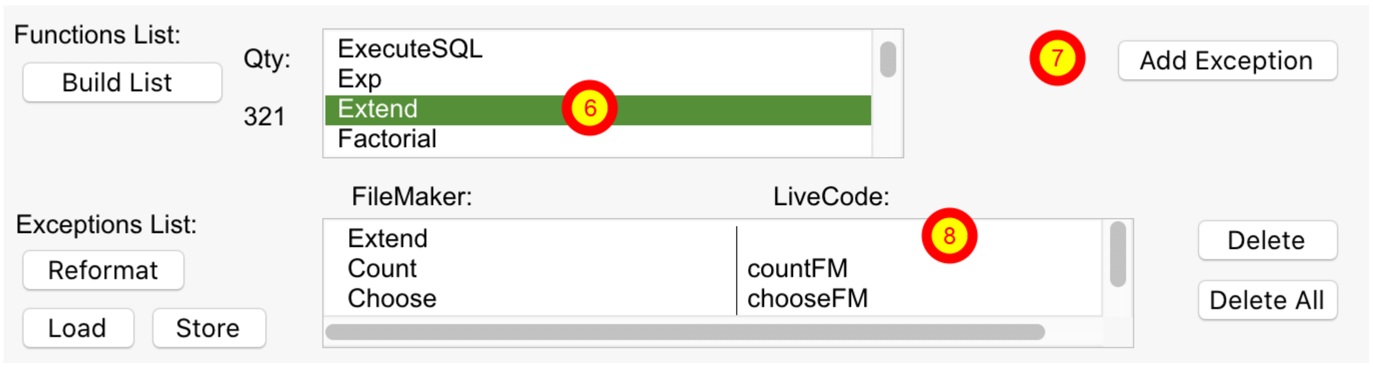
Qty: 321

Abs
Acos
Asin
Atan

The contents of the Web Result field are intended to be correct, but it should be edited manually if necessary prior to proceeding to the next step. After any manual changes have been made to the Web Result field, click the Build List button.

After clicking the Build List button, the functions will be copied from the Web Results field, sorted and counted when placed into the Functions List field. This will be the official list of functions moving forward.

Adding Exceptions to the Functions List - #1

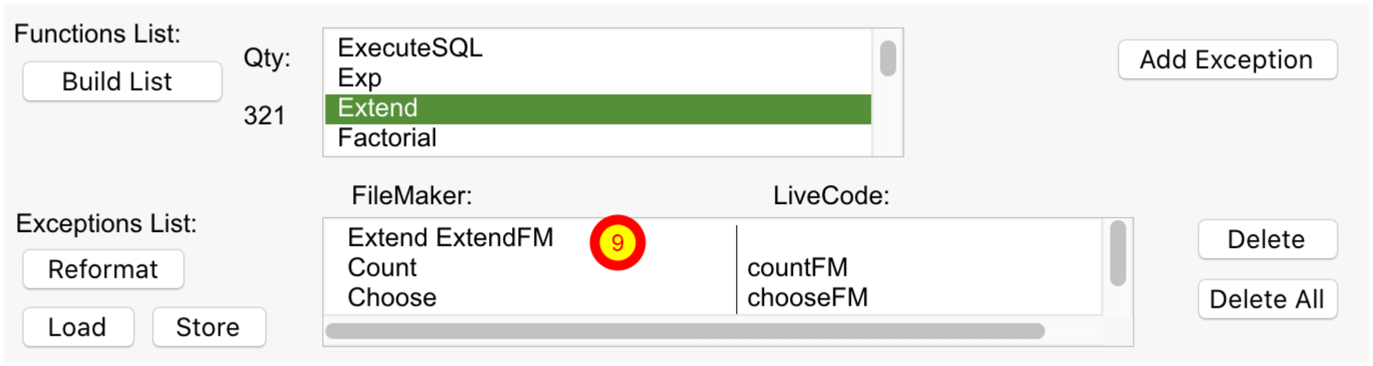


Many FileMaker functions are very close in name and functionality to their equivalent LiveCode functions. The FileMaker Get() functions are remapped automatically because the LiveCode function names have been designed to match the same naming format.

Changes can be made to the rest of the functions by adding exceptions to the list to name a new LiveCode function to replace the FileMaker function.

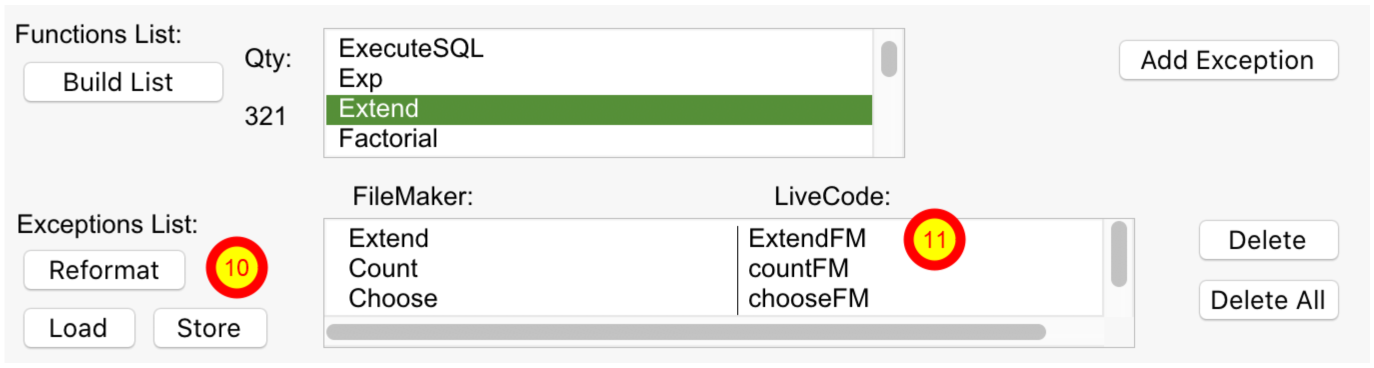
To make this type of addition, (6) click a function in the Functions List field. (7) Click the Add Exception button. Anew entry will be added to the Exceptions List. (8) Click column #2 of the Exceptions List and type the name of the new LiveCode function.

Adding Exceptions to the Functions List - #2



For this example, Extend might be replaced with ExtendFM. Add a space after the Extend function name, (9) then type the new LiveCode function name.

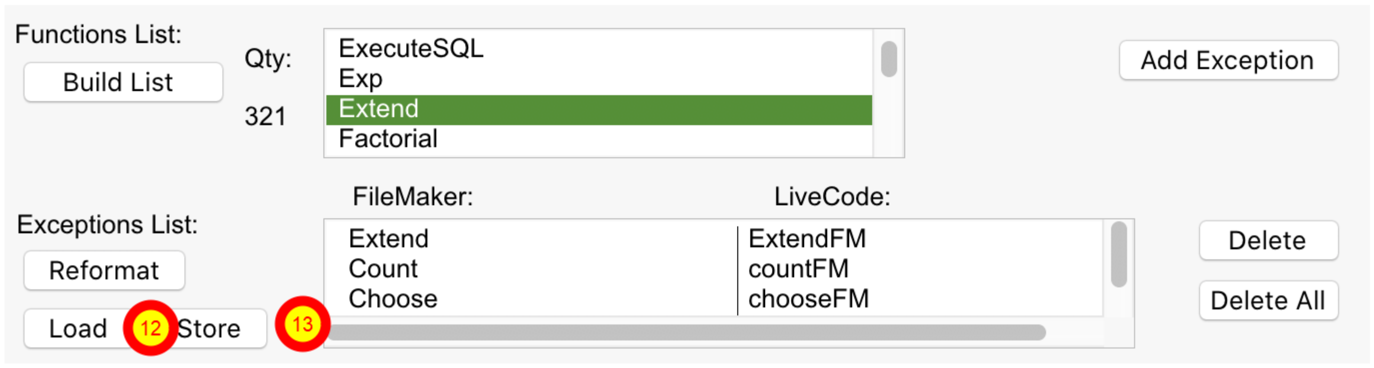
Adding Exceptions to the Functions List - #3



After typing the new LiveCode function name, (10) click the Reformat button, the new LiveCode function name will be moved over to column #2.

Note; If there aren't any functions listed in the Exceptions List, click the Load button to populate this field with the internally stored copy of exceptions.

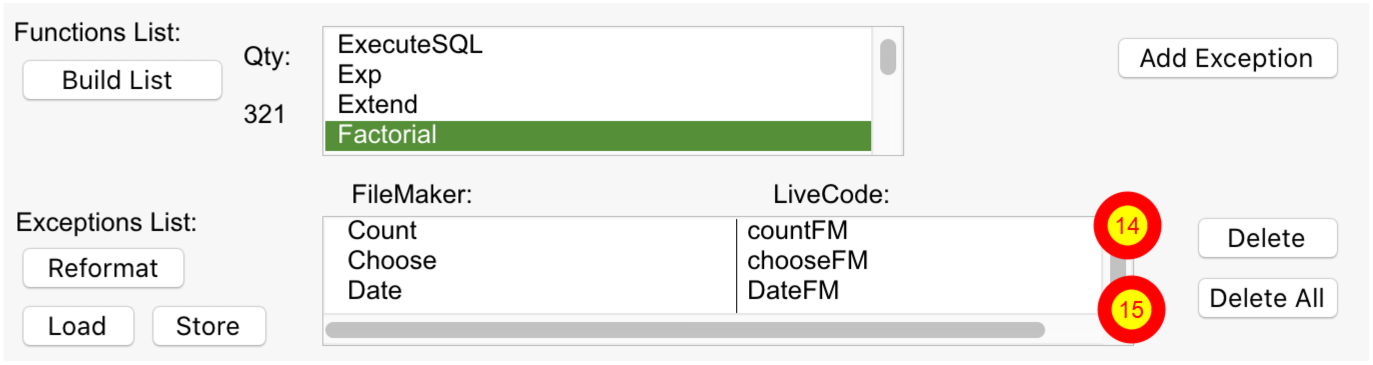
Loading/Saving Exceptions to the Functions List - #4



The (12) Load button is used to load the previously stored copy of the Exceptions List from the FunctionsExceptionsList custom property of the FM Functions to LiveCode Remapping.livecode stack.

After modifying the Exceptions List, (13) click the Store button to save the results back into the FunctionsExceptionsList custom property.

Deleting Exceptions to the Functions List - #5



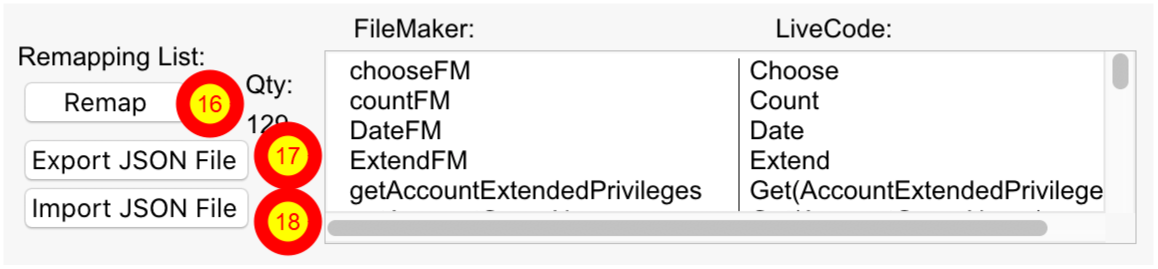
FileMaker:	LiveCode:
Count	countFM
Choose	chooseFM
Date	DateFM

To delete an exception, click the row of the Exceptions List, (14) then click the Delete button.

It is also possible to click the (15) Delete All button to delete all of the exceptions.

After modifying the Exceptions List, (13) click the Store button to save the results back into the FunctionsExceptionsList custom property.

Remapping List & Save JSON File



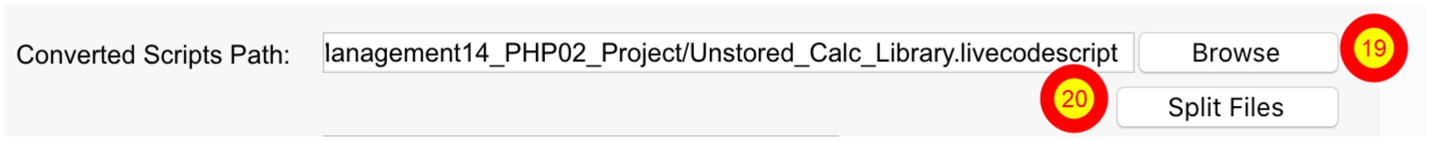
FileMaker:	LiveCode:
chooseFM	Choose
countFM	Count
DateFM	Date
ExtendFM	Extend
getAccountExtendedPrivileges	Get(AccountExtendedPrivilege

Now that the Exceptions List has been created, (16) click the Remap button. The Remapping List field will be populated with all of the function names which can be remapped to LiveCode functions and the qty of functions will be displayed below the Qty label.

Click the (17) Export JSON File button to save the contents of the Remapping List field as a JSON file.

Click the (18) Import JSON File button to load previously saved Remapping List field contents into the field.

Splitting Converted Script Files


Converted Scripts Path:
lanagement14_PHP02_Project/Unstored_Calc_Library.livecodescript
Browse
19
20
Split Files

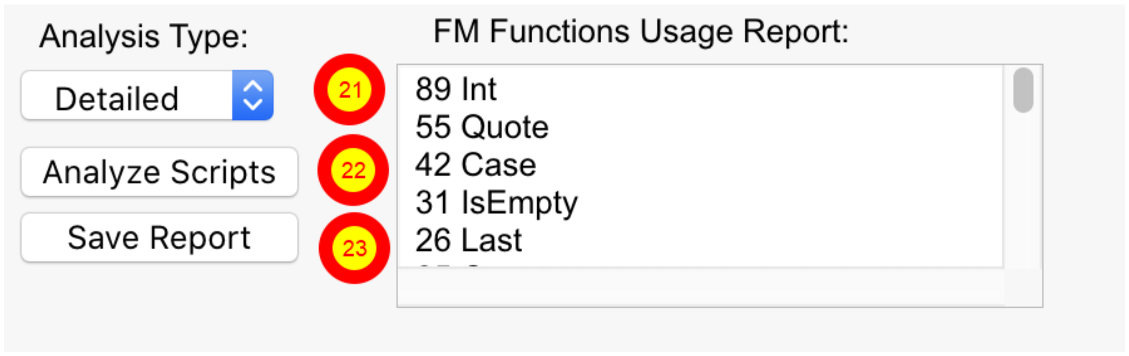
Some of the converted scripts files can become large in size. It is easier to manage the converted code by splitting it into smaller more manageable pieces. To split files into smaller pieces:

(19) Click the Browse button to select a converted script file. Files with .txt and .livecodescript can be selected.

If the file is large in size (1MB or more), (20) click the Split Files button to split the file into smaller parts having about 10,000 lines of code each. Each new file will be given a name ending with: _Part1, _Part2 etc.

Once a file has been split, select the 1st of the pieces to continue remapping of the functions.

Analyze Scripts



```
89 Int
55 Quote
42 Case
31 IsEmpty
26 Last
```

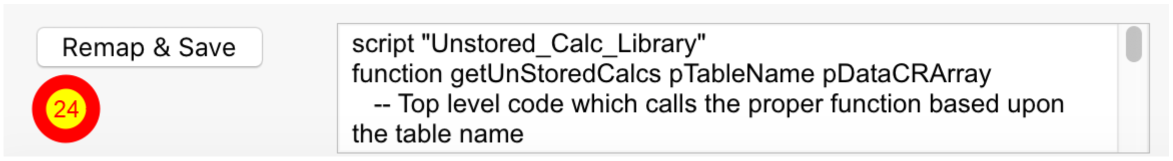
It is helpful to know how many instances of a specific function exist within script files in order to prioritize the writing of new LiveCode replacement functions.

(21) Select a type of analysis [Detailed or Existence].
The Detailed analysis shows how many times a specific function was used, but it takes longer to run.
Selecting the Existence option just determines if a specific function has been used within a script.

(22) Click the Analyze Scripts button to perform the analysis.

(23) Click the Save Report button to save the results of the Usage Report to a text file.

Remap & Save



```
script "Unstored_Calc_Library"
function getUnStoredCalcs pTableName pDataCRArray
   -- Top level code which calls the proper function based upon
the table name
```

(24) Click the Remap & Save button to remap the FileMaker functions into LiveCode functions. The resulting file will be automatically saved using the original filename plus the text "_Remapped" as the new filename.

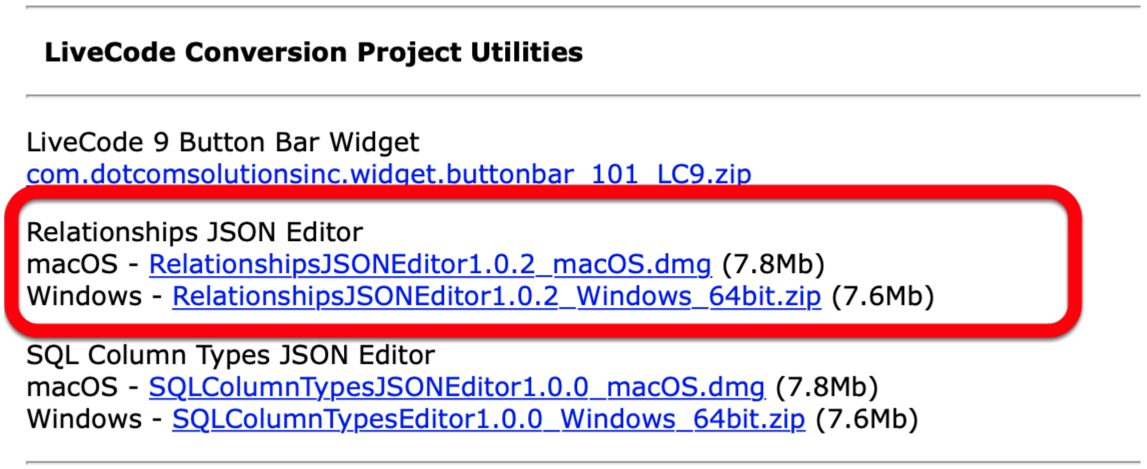
The resulting text file is now ready for development within LiveCode. As part of the development process, it will be a good idea to plan exactly where you want the new scripts to be stored within the converted LiveCode application stack.

Relationships JSON Editor

When performing an automated LiveCode conversion project, FmPro Migrator generates the Relationships.JSON file from the relationships imported from within the original database. This file is saved to disk and is also imported into the RelationshipsJSONProp custom property of the SQL_Library substack created by FmPro Migrator.

The Relationship JSON Editor utility serves as a graphical tool for editing and saving the relationships JSON data as the LiveCode app continues to be updated.

Downloading the Relationships JSON Editor



LiveCode Conversion Project Utilities

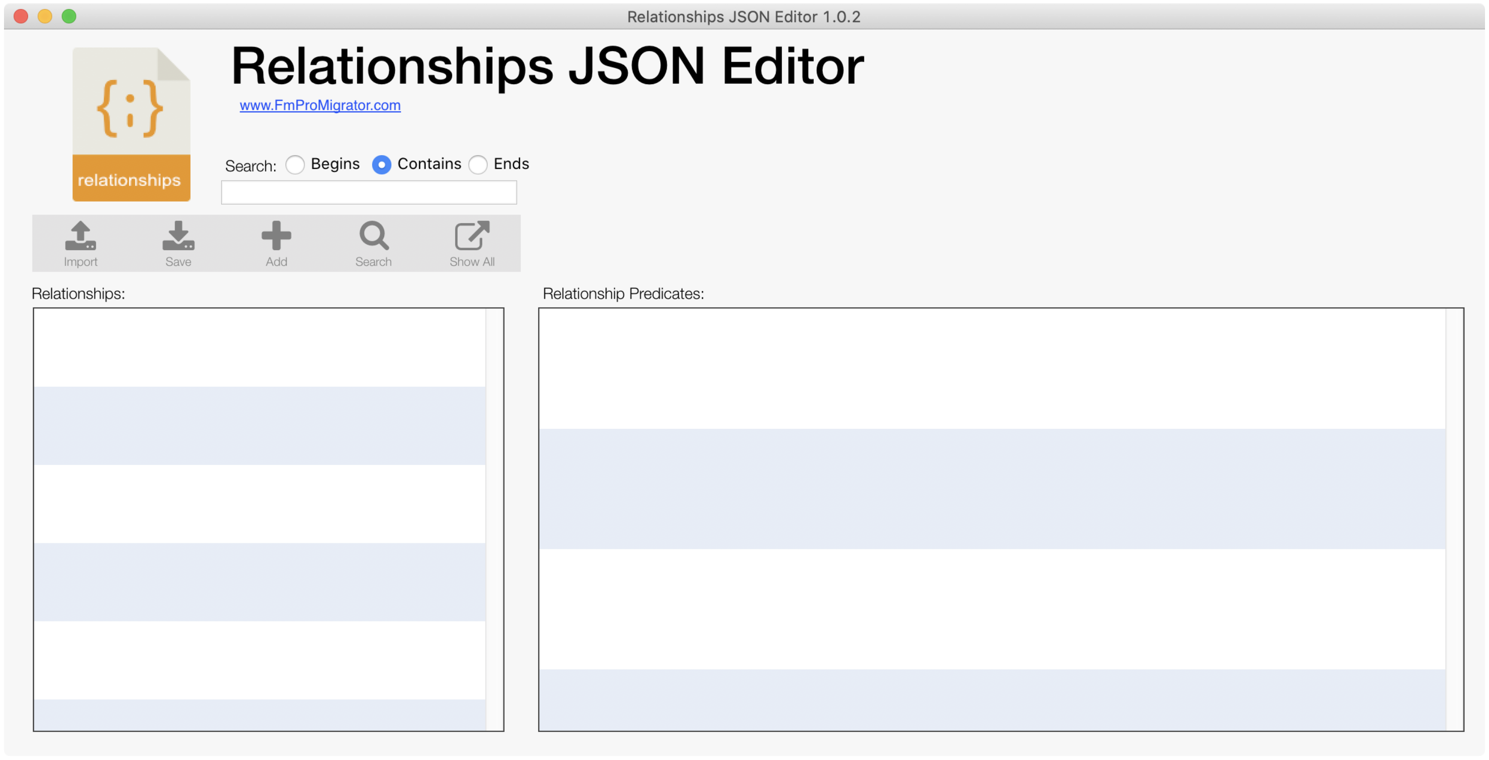
LiveCode 9 Button Bar Widget
com.dotcomsolutionsinc.widget.buttonbar_101_LC9.zip

Relationships JSON Editor
macOS - RelationshipsJSONEditor1.0.2_macOS.dmg (7.8Mb)
Windows - RelationshipsJSONEditor1.0.2_Windows_64bit.zip (7.6Mb)

SQL Column Types JSON Editor
macOS - SQLColumnTypesJSONEditor1.0.0_macOS.dmg (7.8Mb)
Windows - SQLColumnTypesEditor1.0.0_Windows_64bit.zip (7.6Mb)

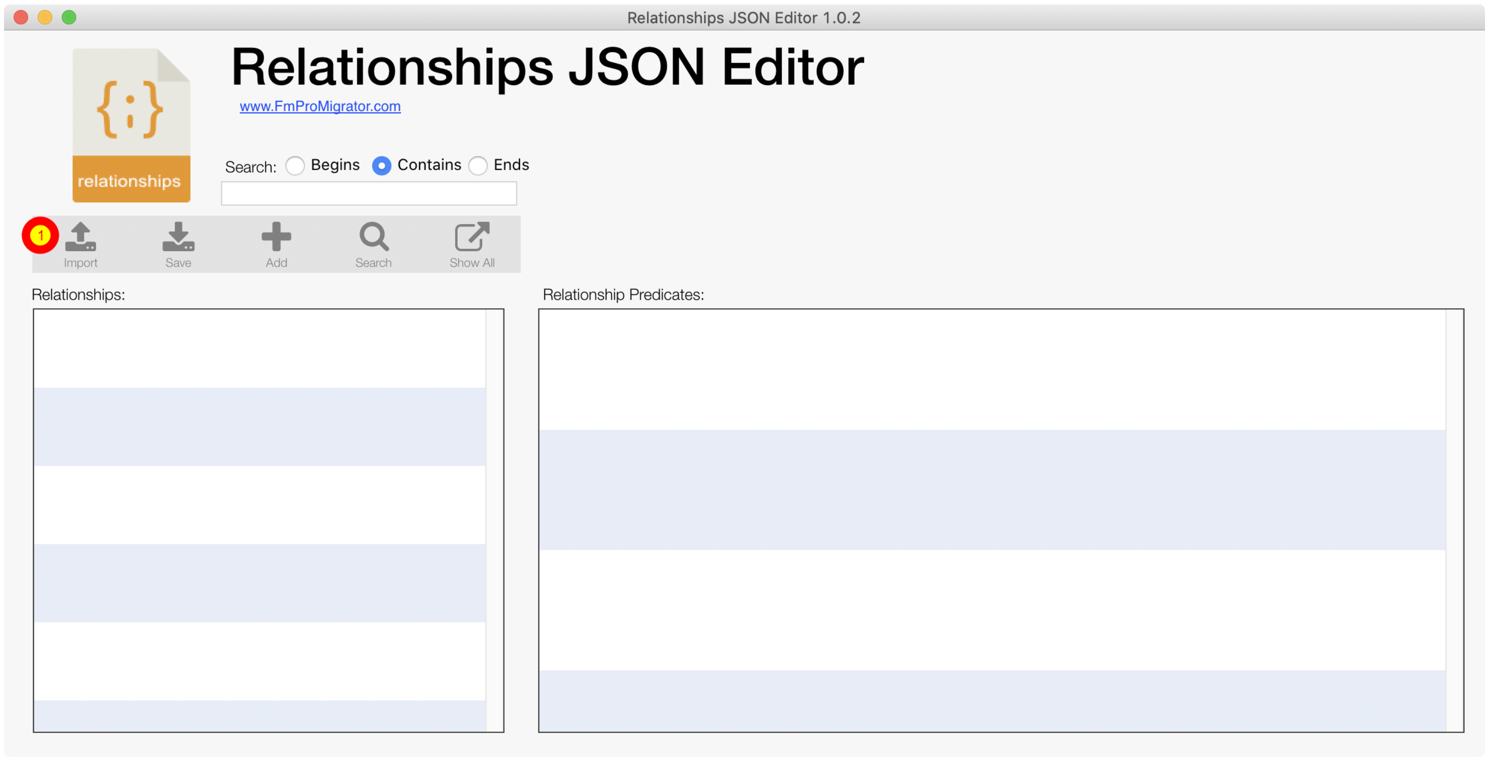
The Relationships JSON Editor is available on the same web page where the fully functional FmPro Migrator was downloaded (not the Trial version). This URL is provided with your email purchase receipt.

Relationships JSON Editor Overview



The starting point for using the Relationships JSON Editor is the button bar near the top of the window.

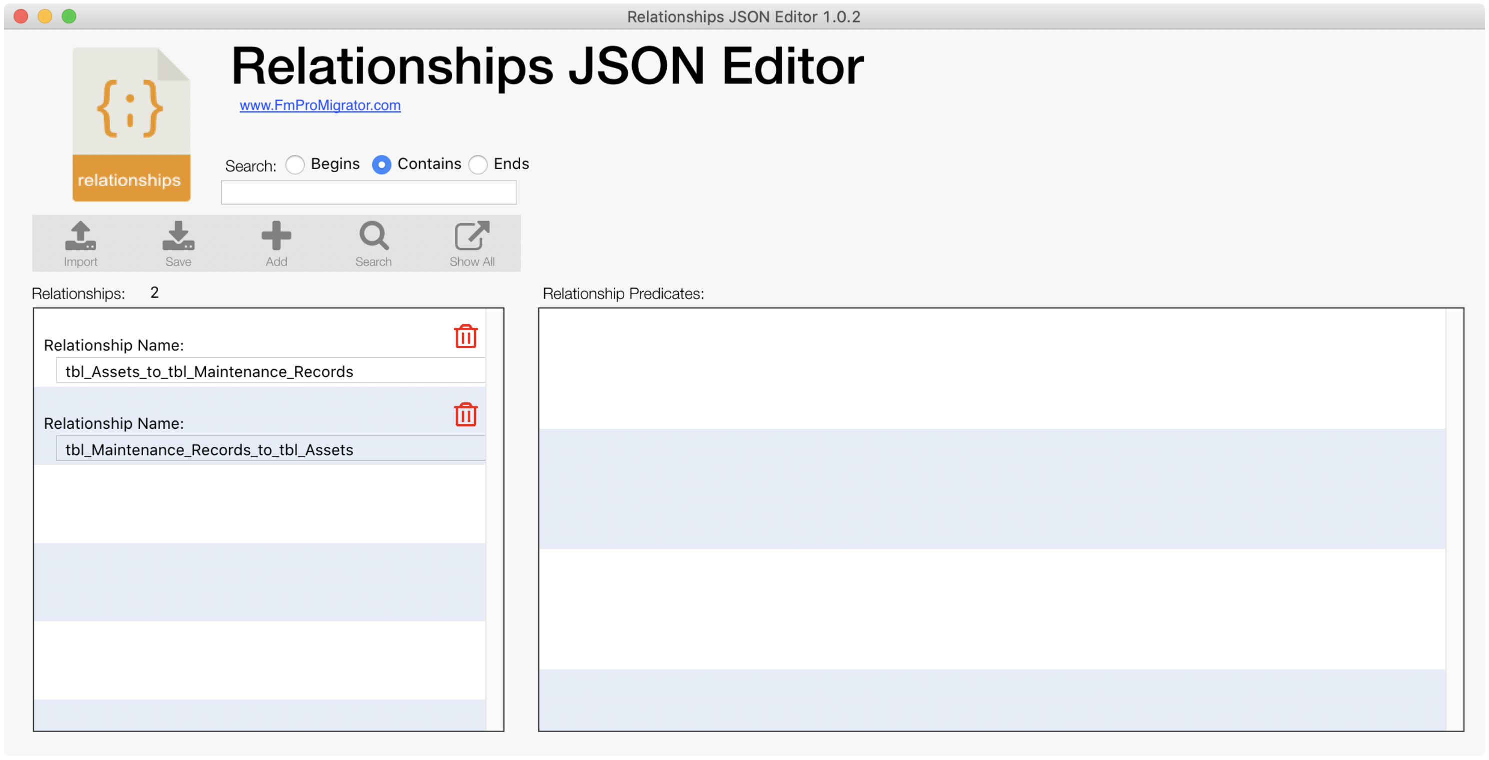
Importing the Relationships.JSON File



Click (1) the Import button to import the Relationships.JSON file.

<u>Note</u>: No data validation is performed when reading the JSON file. If the a random file is imported by accident, no data will be displayed in the Relationships data grid on the left side, but rows of empty data will be displayed.

Import Results



Relationships JSON Editor

www.FmProMigrator.com

Search: Begins [✓ Contains] Ends

Relationships: 2

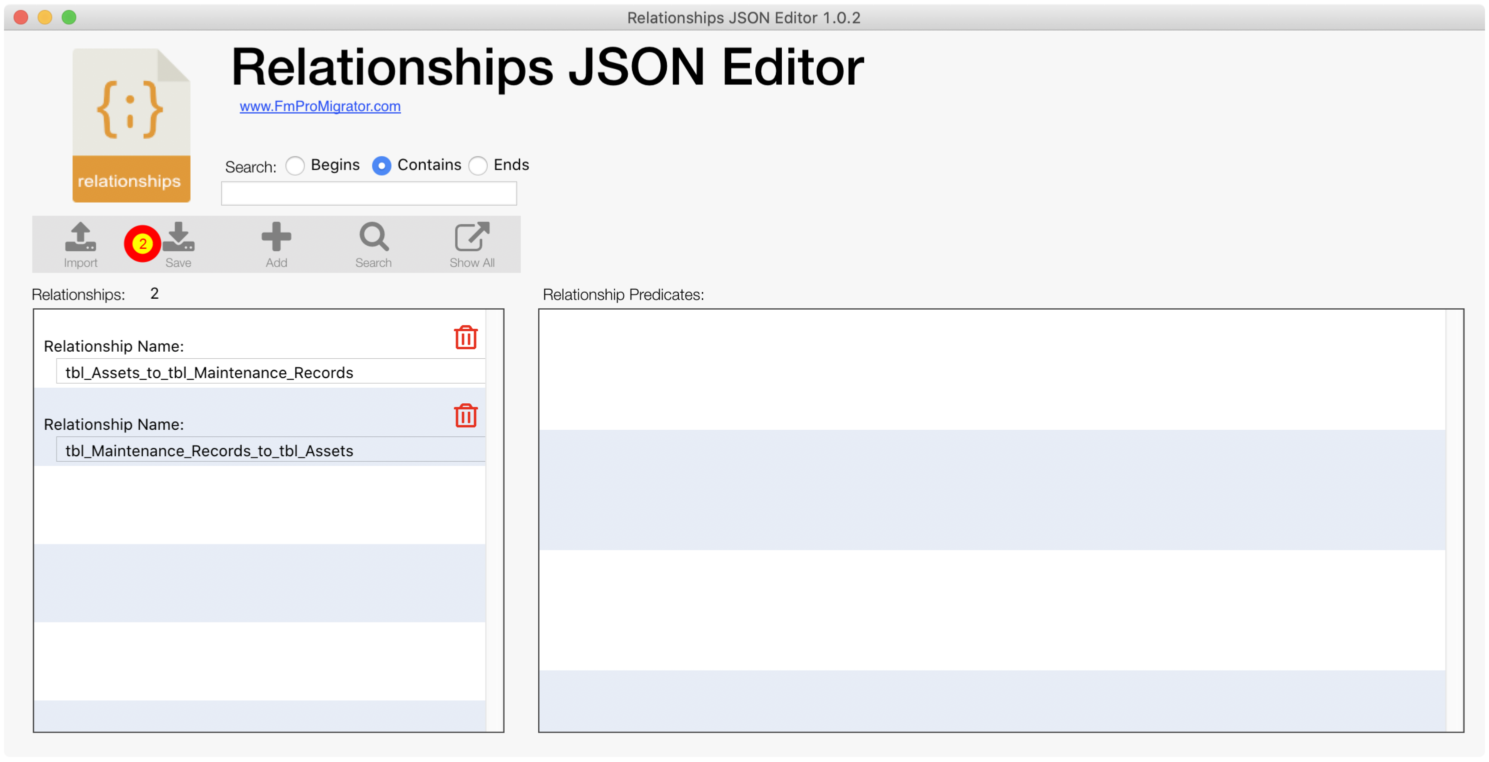
Relationship Predicates:

Relationship Name:
tbl_Assets_to_tbl_Maintenance_Records

Relationship Name:
tbl_Maintenance_Records_to_tbl_Assets

Once the Relationships.JSON file has been imported, the names of each relationship will be shown in the Relationships data grid on the left side.
The count of relationships is displayed at the top of the grid.

Saving the Relationships.JSON File



Relationships: 2

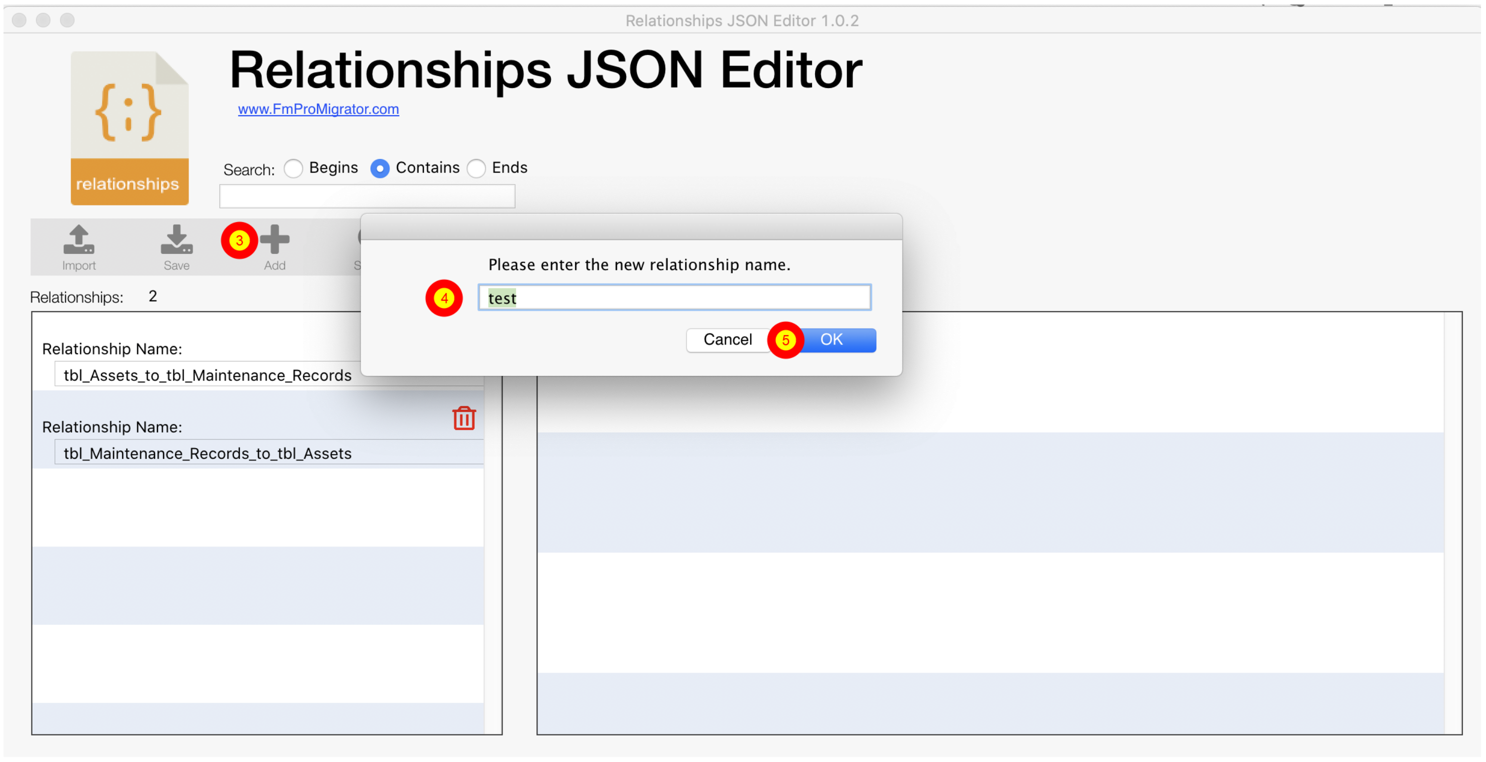
Relationship Name: tbl_Assets_to_tbl_Maintenance_Records

Relationship Name: tbl_Maintenance_Records_to_tbl_Assets

Click the (2) Save button to save the relationships (including predicates) if editing has been done. All editing occurs in memory, and the data only gets saved by clicking the save button and entering a filename. Since the file is a text file, it can be copied to change management systems as needed.

After the Relationships.JSON file has been updated, it should be re-imported back into the SQ_Library by clicking the loadRelationshipsJSON button on the 1st card of the SQL_Library stack.

Add New Relationship


Relationships JSON Editor 1.0.2

Relationships JSON Editor

www.FmProMigrator.com

Search: Begins [✓ Contains] Ends

Relationships: 2

Relationship Name:
tbl_Assets_to_tbl_Maintenance_Records

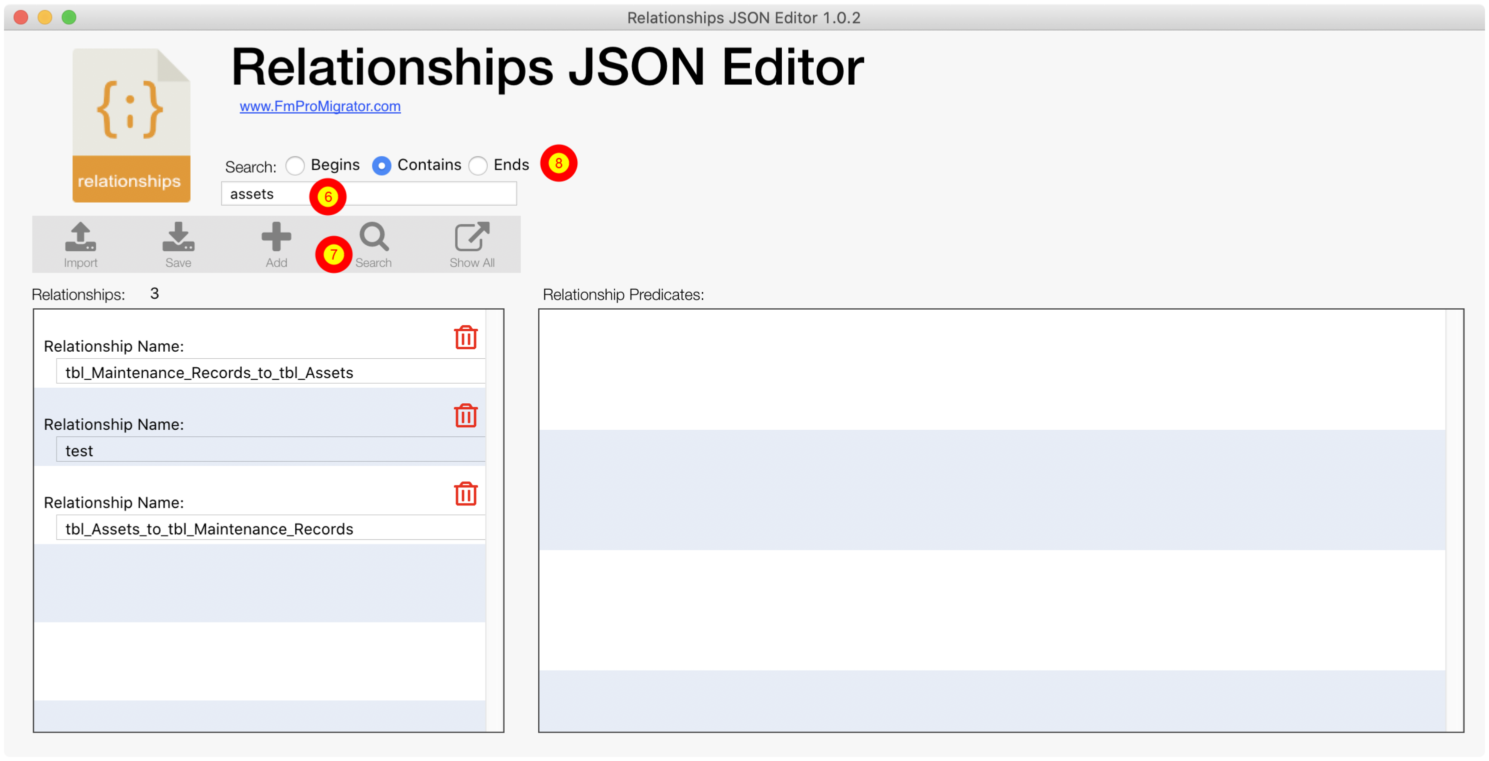
Relationship Name:
tbl_Maintenance_Records_to_tbl_Assets

Please enter the new relationship name.

test

Click the (3) Add button to add a new relationship, (4) enter the new relationship name in the dialog box then (5) click the Ok button. The new relationship with 0 predicates will be added to the list of relationships.

Searching Relationship Names



Search: Begins [✓ Contains] Ends

assets

Relationships: 3

Relationship Predicates:

Relationship Name: tbl_Maintenance_Records_to_tbl_Assets

Relationship Name: test

Relationship Name: tbl_Assets_to_tbl_Maintenance_Records

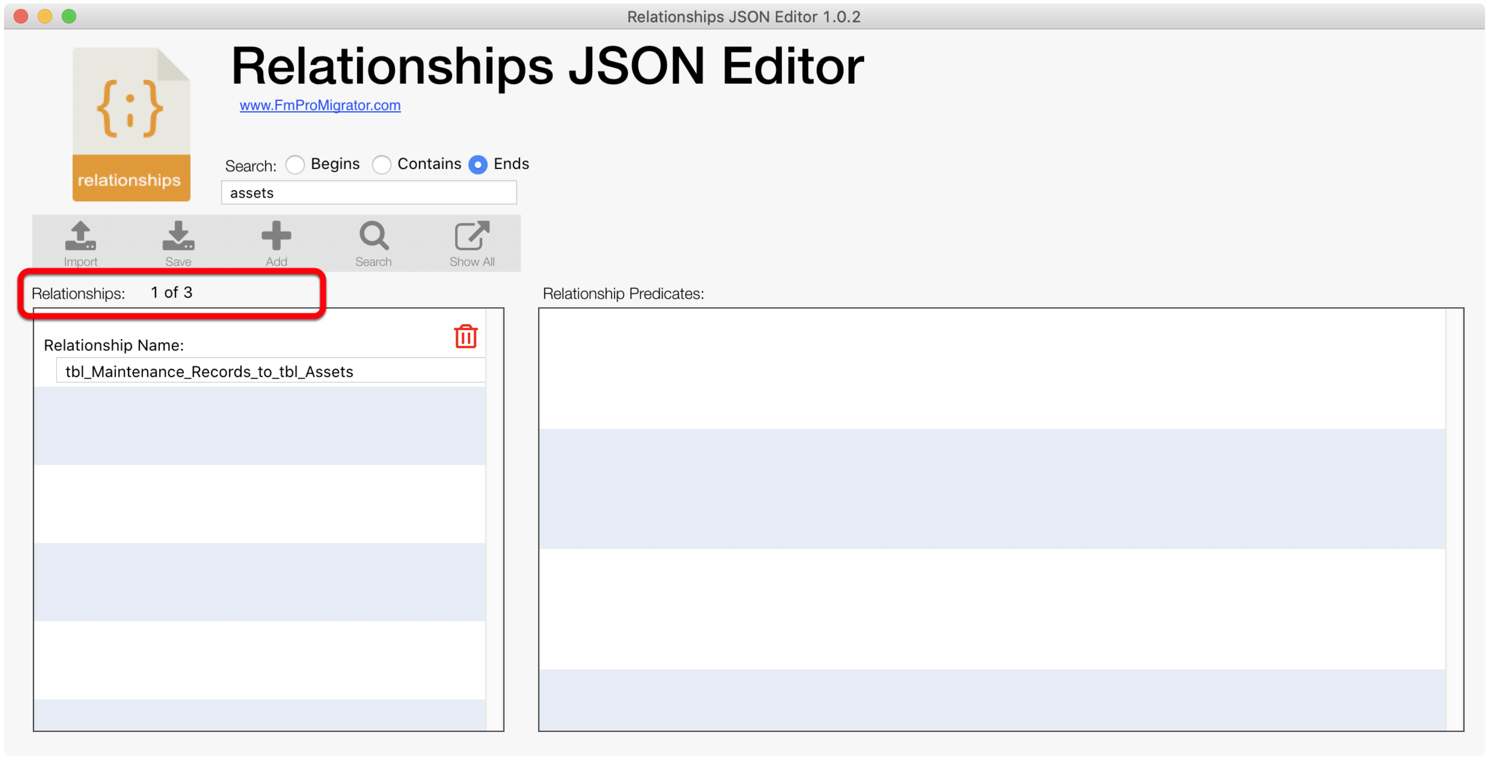
To search thru the relationship names, enter search text (6) within the search field, (7) click the Search button or (8) click the search type radio buttons. The search will be performed and only the matching records will be displayed in the Relationships grid.

To perform a search, any of the following actions will start the search:
1] Type search text in the search field, then tab out of the field or click the background.
2] Click the Search button.
3] Click on any of the search type buttons Begins, Contains, Ends above the search field.

(8) The search type radio buttons perform the search as follows:
Begins: Searches the relationship names and matches the ones which begin with the search text.
Contains: Searches the relationship names and matches the ones which contain the search text.
Ends: Searches the relationship names and matches the ones which end with the search text.

During testing, even JSON files having over 1000 relationships return results in less than 1 second on modern machines.

Search Results


Relationships JSON Editor 1.0.2

relationships

Relationships JSON Editor

www.FmProMigrator.com

Search: Begins Contains [✓ Ends]

assets

Relationships: 1 of 3

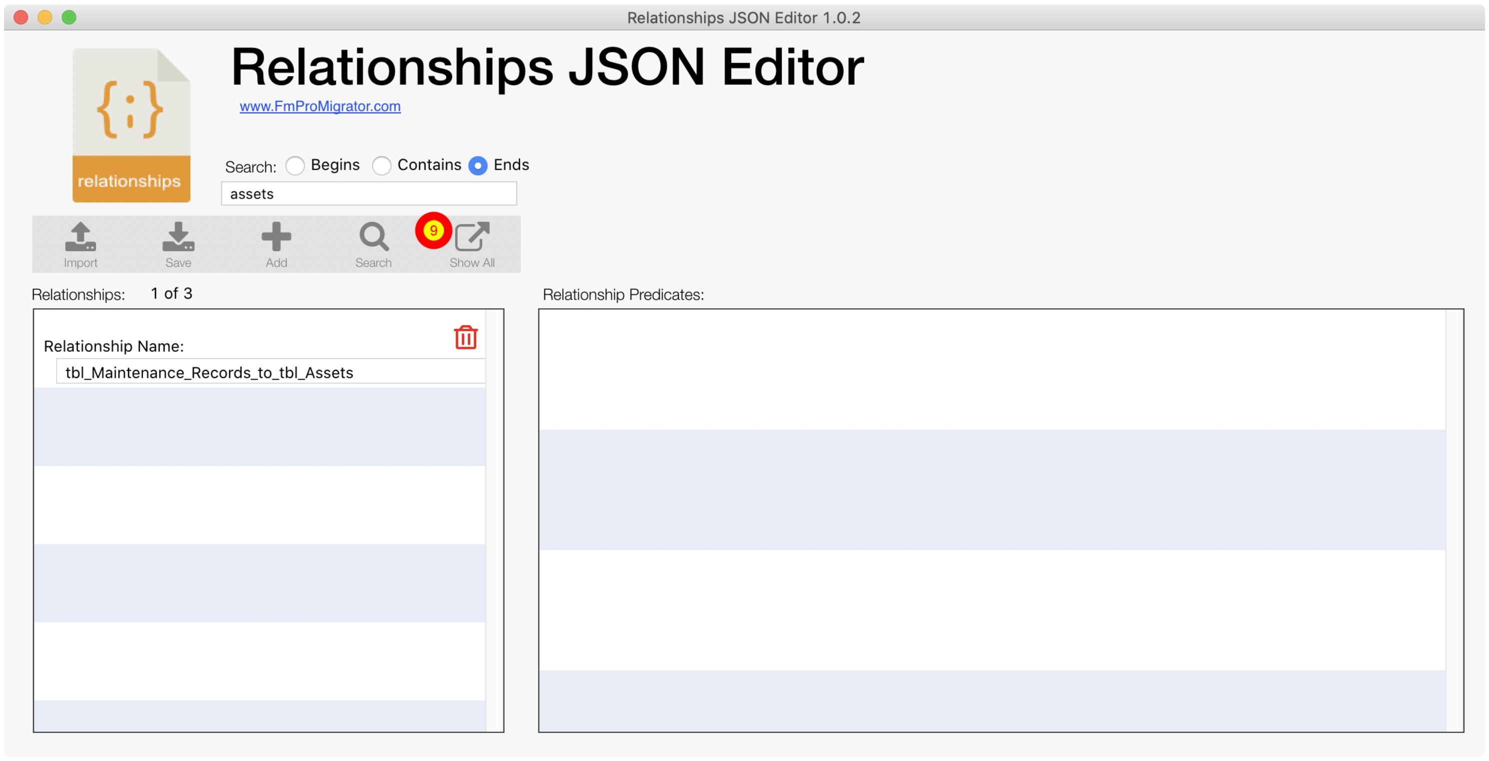
Relationship Predicates:

Relationship Name:

tbl_Maintenance_Records_to_tbl_Assets

Searching for relationships displays a search results label showing the number of found records, followed by the total number of relationships being searched.

Show All - Search Results



Relationships JSON Editor

www.FmProMigrator.com

Search: ( ) Begins ( ) Contains [✓ Ends]

assets

Relationships: 1 of 3

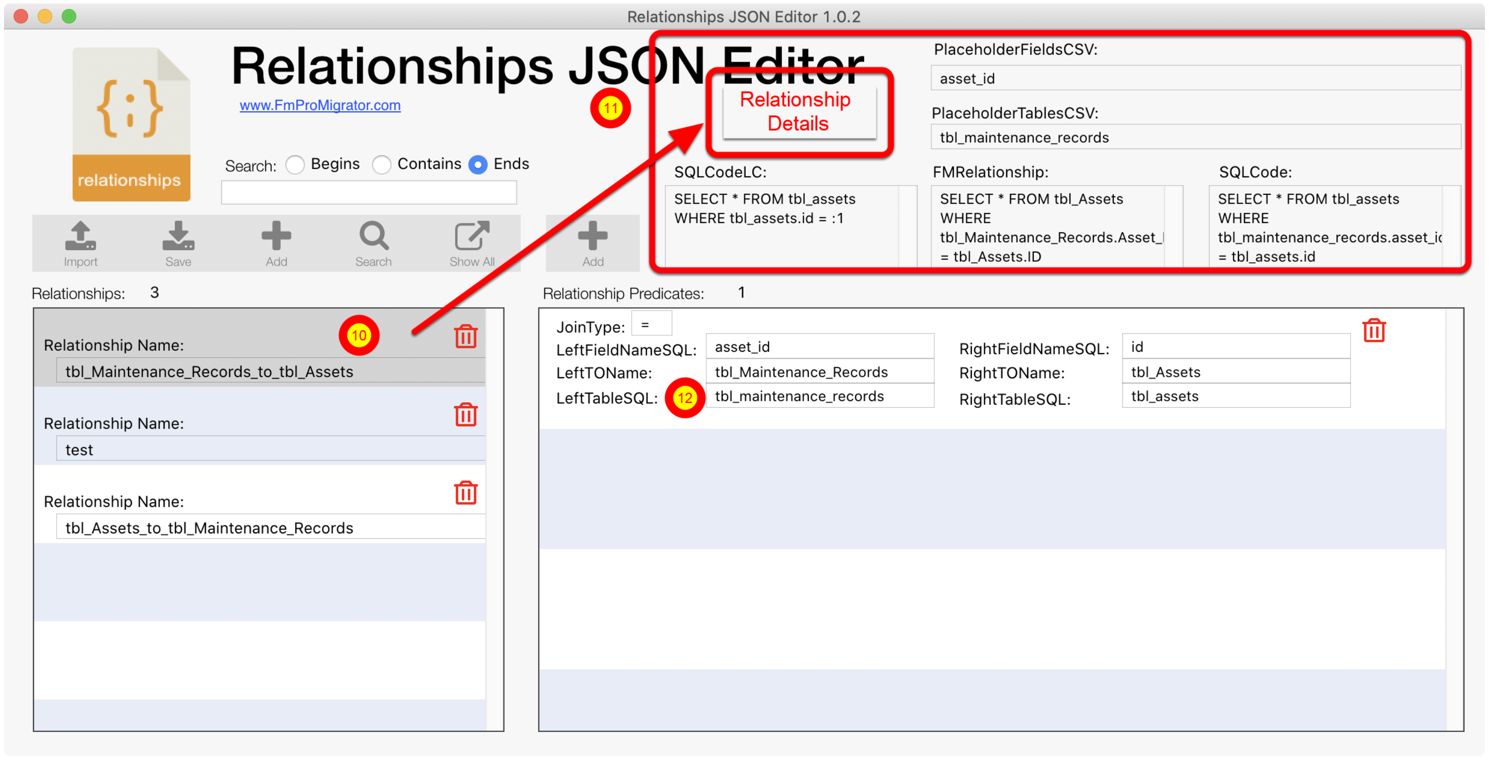
Relationship Name:

tbl_Maintenance_Records_to_tbl_Assets

Relationship Predicates:

Clicking the (9) Show All button results in the display of all relationships.

Editing Relationship Information


Relationships JSON Editor 1.0.2

Relationships JSON Editor

www.FmProMigrator.com

Search: ( ) Begins ( ) Contains [✓ Ends]

Relationships: 3

Relationship Name: tbl_Maintenance_Records_to_tbl_Assets
Relationship Name: test
Relationship Name: tbl_Assets_to_tbl_Maintenance_Records

PlaceholderFieldsCSV: asset_id

PlaceholderTablesCSV: tbl_maintenance_records

SQLCodeLC:
```
SELECT * FROM tbl_assets
WHERE tbl_assets.id = :1
```

FMRelationship:
```
SELECT * FROM tbl_Assets
WHERE
tbl_Maintenance_Records.Asset_I
= tbl_Assets.ID
```

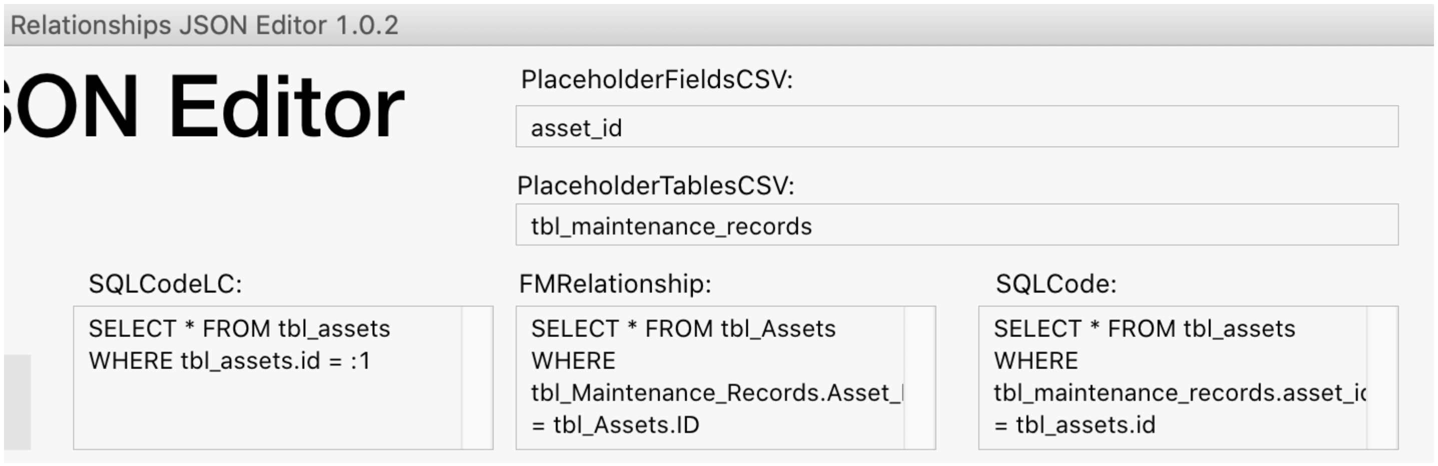
SQLCode:
```
SELECT * FROM tbl_assets
WHERE
tbl_maintenance_records.asset_ic
= tbl_assets.id
```

Relationship Predicates: 1

JoinType:	=		
LeftFieldNameSQL:	asset_id	RightFieldNameSQL:	id
LeftTOName:	tbl_Maintenance_Records	RightTOName:	tbl_Assets
LeftTableSQL:	tbl_maintenance_records	RightTableSQL:	tbl_assets

Clicking (10) any row of the list of relationships grid will display detailed info for the relationship (11), along with a list of predicates (12) for that relationship in the Relationship Predicates grid on the right.

Editing Relationship Information - Relationship Details


Relationships JSON Editor 1.0.2

SON Editor

PlaceholderFieldsCSV: asset_id

PlaceholderTablesCSV: tbl_maintenance_records

SQLCodeLC:
```
SELECT * FROM tbl_assets
WHERE tbl_assets.id = :1
```

FMRelationship:
```
SELECT * FROM tbl_Assets
WHERE
tbl_Maintenance_Records.Asset_I
= tbl_Assets.ID
```

SQLCode:
```
SELECT * FROM tbl_assets
WHERE
tbl_maintenance_records.asset_ic
= tbl_assets.id
```

The contents of the relationship details fields are used directly by the generated LiveCode application to run queries on related tables, either for single row data or multiple rows of data to fill data grids in a manner similar to the original portal on a form/layout. The info displayed in these fields has been generated automatically by FmPro Migrator from the original database.

Manual changes to the predicates will require equivalent changes to these fields.

These fields include:
PlaceholderFieldsCSV - A CSV list of placeholder field names values is used to query the columns of the SQL database.

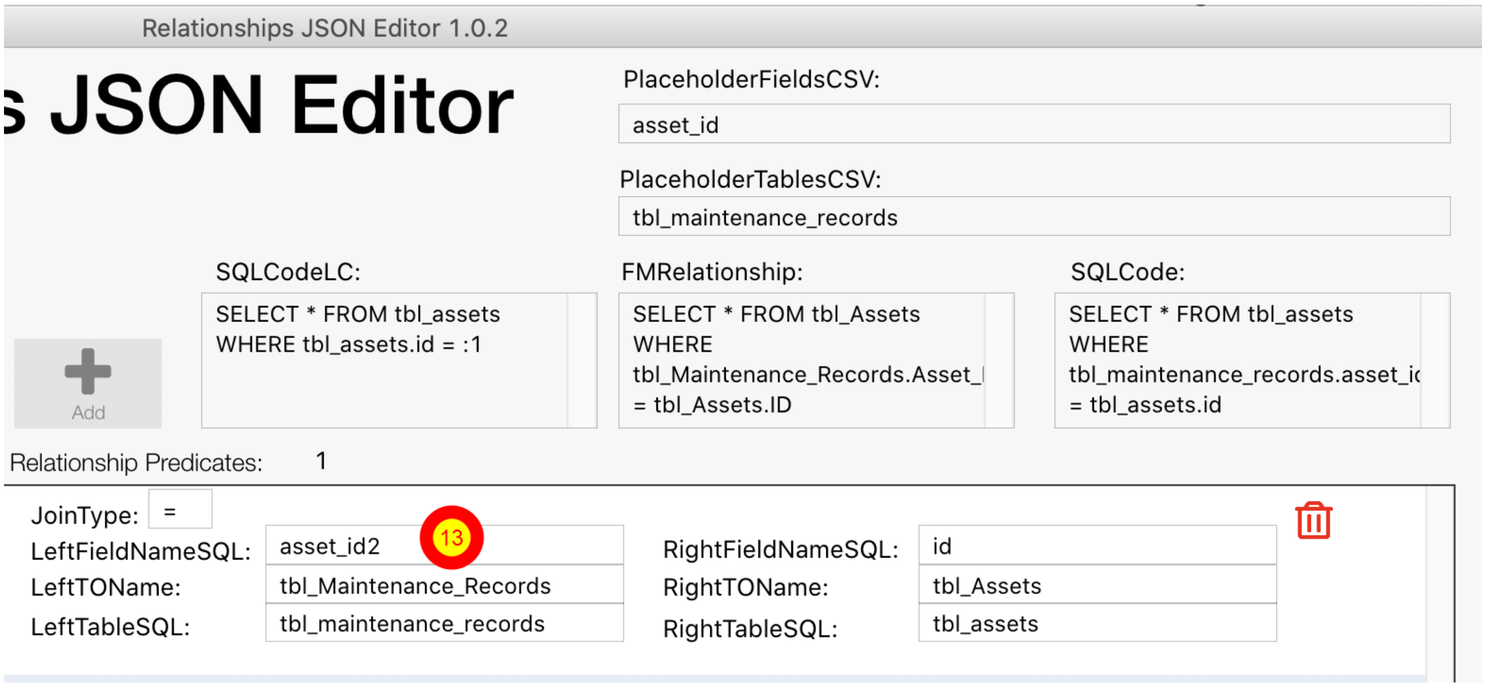
PlaceholderTablesCSV - A CSV list of placeholder table names is used when querying the related table(s) in the SQL database.

SQLCodeLC - This is the SQL code executed by LiveCode, where placeholder fields have been replaced with placeholders (:1, :2 etc) passed into the query hander.

FMRelationship - This is a SQL representation of the original query as it might appear in a FileMaker database, using the original table occurrence names.

SQLCode - The FileMaker table occurrence names have been replaced with the underlying SQL database table in the database server.

Editing Relationship Predicates



PlaceholderFieldsCSV: asset_id

PlaceholderTablesCSV: tbl_maintenance_records

SQLCodeLC:
```
SELECT * FROM tbl_assets
WHERE tbl_assets.id = :1
```

FMRelationship:
```
SELECT * FROM tbl_Assets
WHERE
tbl_Maintenance_Records.Asset_I
= tbl_Assets.ID
```

SQLCode:
```
SELECT * FROM tbl_assets
WHERE
tbl_maintenance_records.asset_ic
= tbl_assets.id
```

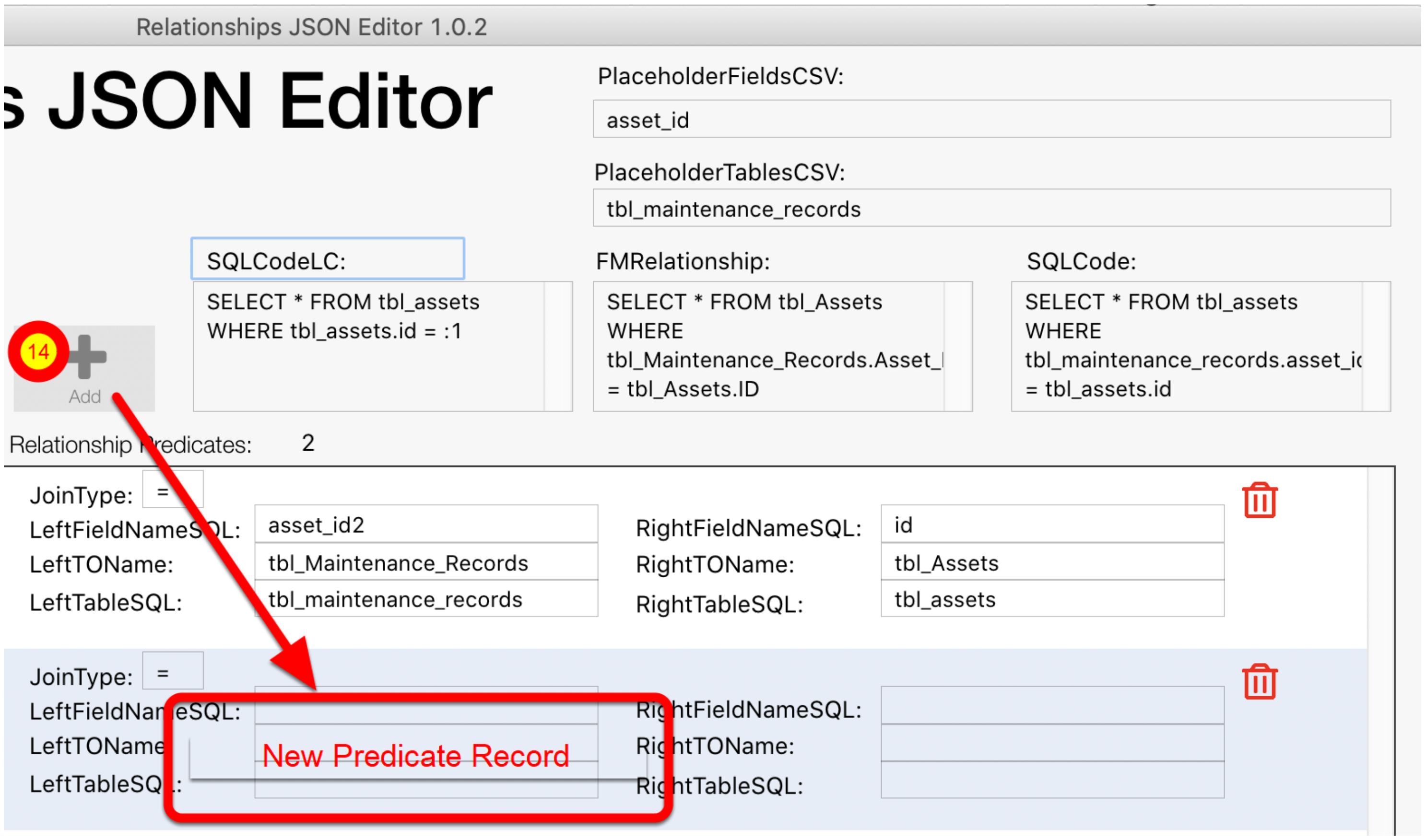
Relationship Predicates: 1

JoinType: =

LeftFieldNameSQL:	asset_id2	RightFieldNameSQL:	id
LeftTOName:	tbl_Maintenance_Records	RightTOName:	tbl_Assets
LeftTableSQL:	tbl_maintenance_records	RightTableSQL:	tbl_assets

Click on any field (13) in the Relationship Predicates grid to edit its field contents. Changes are saved into memory automatically, but saving to disk only occurs when clicking the top-level Save button.

Adding a Relationship Predicate


Relationships JSON Editor 1.0.2

s JSON Editor

PlaceholderFieldsCSV: asset_id

PlaceholderTablesCSV: tbl_maintenance_records

SQLCodeLC:

```
SELECT * FROM tbl_assets
WHERE tbl_assets.id = :1
```

FMRelationship:

```
SELECT * FROM tbl_Assets
WHERE
tbl_Maintenance_Records.Asset_I
= tbl_Assets.ID
```

SQLCode:

```
SELECT * FROM tbl_assets
WHERE
tbl_maintenance_records.asset_id
= tbl_assets.id
```

Relationship Predicates: 2

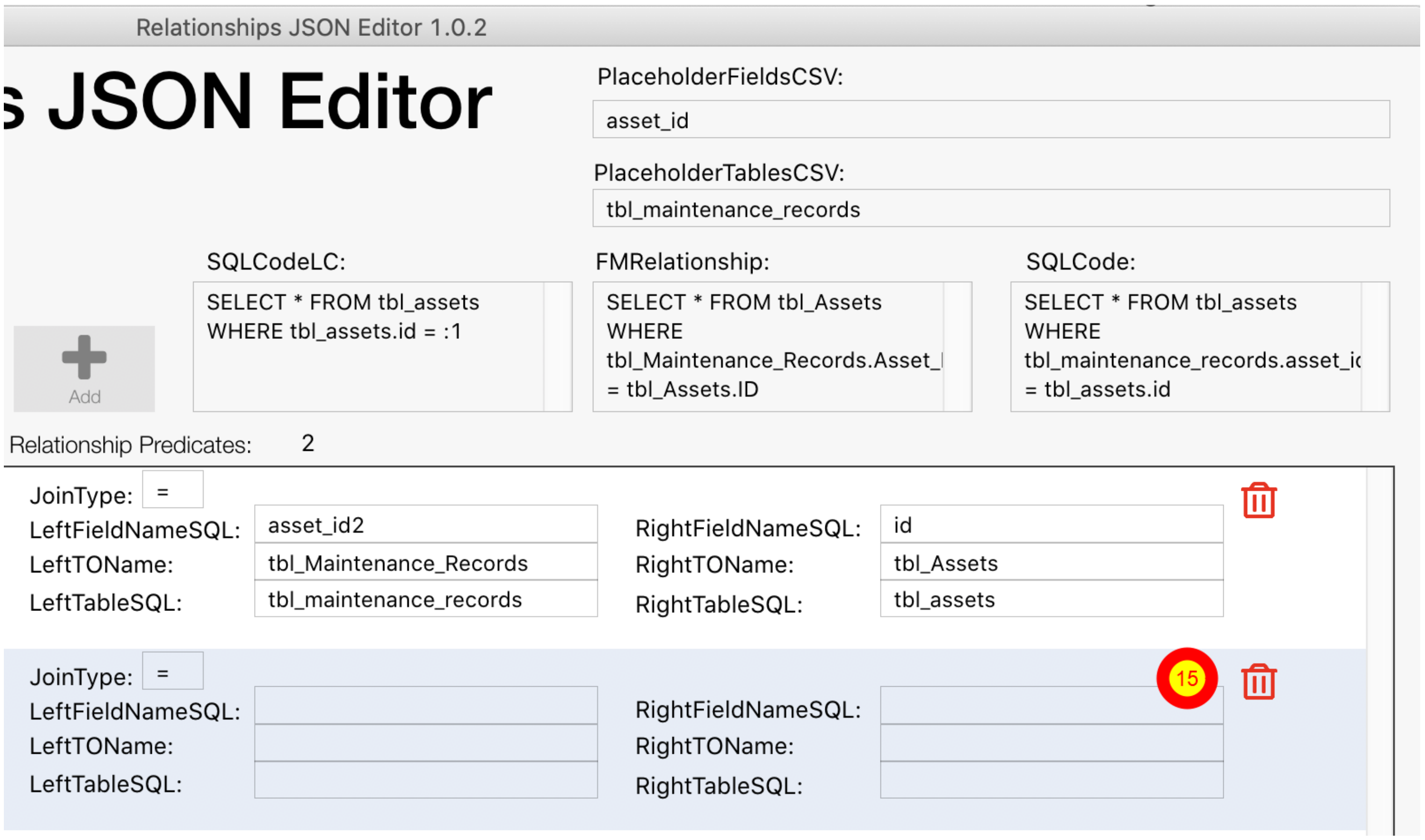
JoinType:	=		
LeftFieldNameSQL:	asset_id2	RightFieldNameSQL:	id
LeftTOName:	tbl_Maintenance_Records	RightTOName:	tbl_Assets
LeftTableSQL:	tbl_maintenance_records	RightTableSQL:	tbl_assets

JoinType:	=		
LeftFieldNameSQL:		RightFieldNameSQL:	
LeftTOName:		RightTOName:	
LeftTableSQL:		RightTableSQL:	

Click (14) the Add button above the Relationship Predicates grid to add a new empty predicate record. The record will be created immediately and added to the grid. Click in each of the fields to edit the info for the relationship.

Note: After adding the predicate, you must manually make the same changes to the top-level relationship fields above the Relationship Predicate grid.

Deleting a Relationship Predicate



PlaceholderFieldsCSV: asset_id

PlaceholderTablesCSV: tbl_maintenance_records

SQLCodeLC:
```
SELECT * FROM tbl_assets
WHERE tbl_assets.id = :1
```

FMRelationship:
```
SELECT * FROM tbl_Assets
WHERE
tbl_Maintenance_Records.Asset_I
= tbl_Assets.ID
```

SQLCode:
```
SELECT * FROM tbl_assets
WHERE
tbl_maintenance_records.asset_ic
= tbl_assets.id
```

Relationship Predicates: 2

JoinType:	=		
LeftFieldNameSQL:	asset_id2	RightFieldNameSQL:	id
LeftTOName:	tbl_Maintenance_Records	RightTOName:	tbl_Assets
LeftTableSQL:	tbl_maintenance_records	RightTableSQL:	tbl_assets

JoinType:	=		
LeftFieldNameSQL:		RightFieldNameSQL:	
LeftTOName:		RightTOName:	
LeftTableSQL:		RightTableSQL:	

Click the (15) trash can icon at the right side of the record to delete an individual relationship predicate.

SQL Column Types JSON Editor

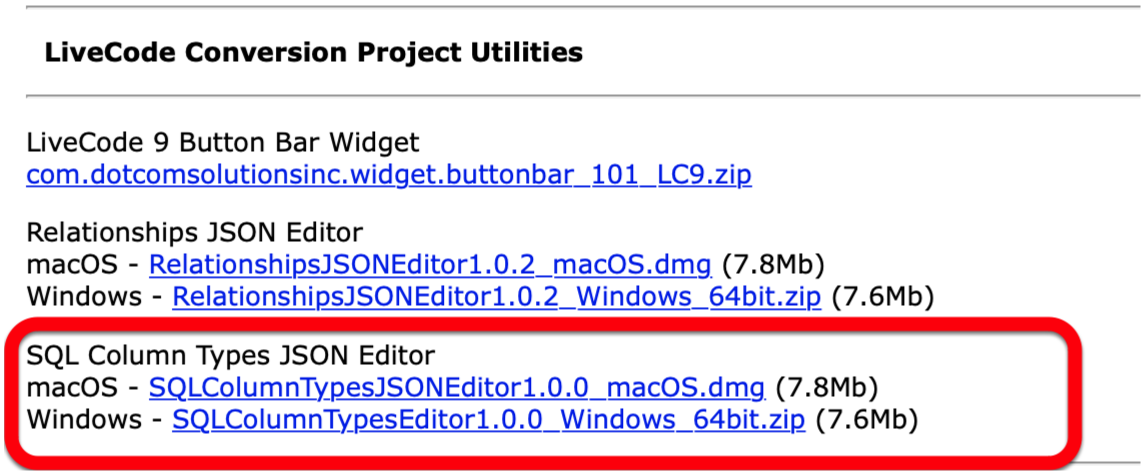
When performing an automated LiveCode conversion project, FmPro Migrator generates the SQLColumnTypes.JSON file from the tables/fields imported from within the original database. This file is saved to disk and is also imported into the SQLColumnTypesJSONProp custom property of the SQL_Library substack created by FmPro Migrator.

When a database connection has been made, this JSON data is converted into the gSQLColumnTypesArray. The getBLOBColumnStatus() function in the SQL_Library uses this array to determine whether the SQL database column contains BLOB data when reading/writing data in the database.

This test is performed in order to avoid performing textEncode/textDecode on the data for BLOB columns.
The test is being done on the original database field type, looking for the type = "PICT" due to the consistency of always checking the FileMaker database type, since there are many database types used for various destination databases.

The SQL Column Types JSON Editor utility serves as a graphical tool for editing and saving the SQL Column Types JSON data as the LiveCode app continues to be updated.

Downloading the SQL Column Types JSON Editor



LiveCode Conversion Project Utilities

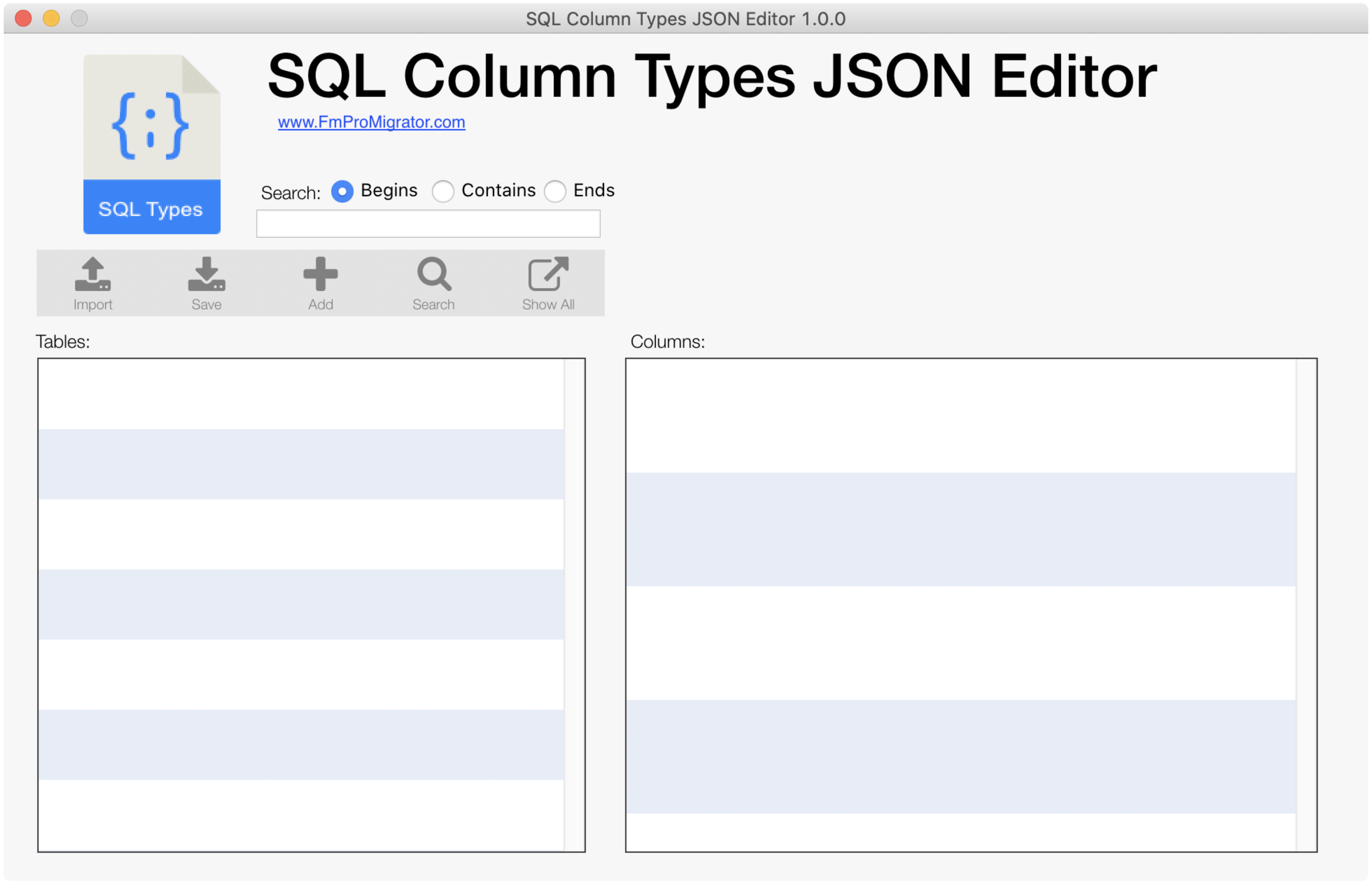
LiveCode 9 Button Bar Widget
com.dotcomsolutionsinc.widget.buttonbar_101_LC9.zip

Relationships JSON Editor
macOS - RelationshipsJSONEditor1.0.2_macOS.dmg (7.8Mb)
Windows - RelationshipsJSONEditor1.0.2_Windows_64bit.zip (7.6Mb)

SQL Column Types JSON Editor
macOS - SQLColumnTypesJSONEditor1.0.0_macOS.dmg (7.8Mb)
Windows - SQLColumnTypesEditor1.0.0_Windows_64bit.zip (7.6Mb)

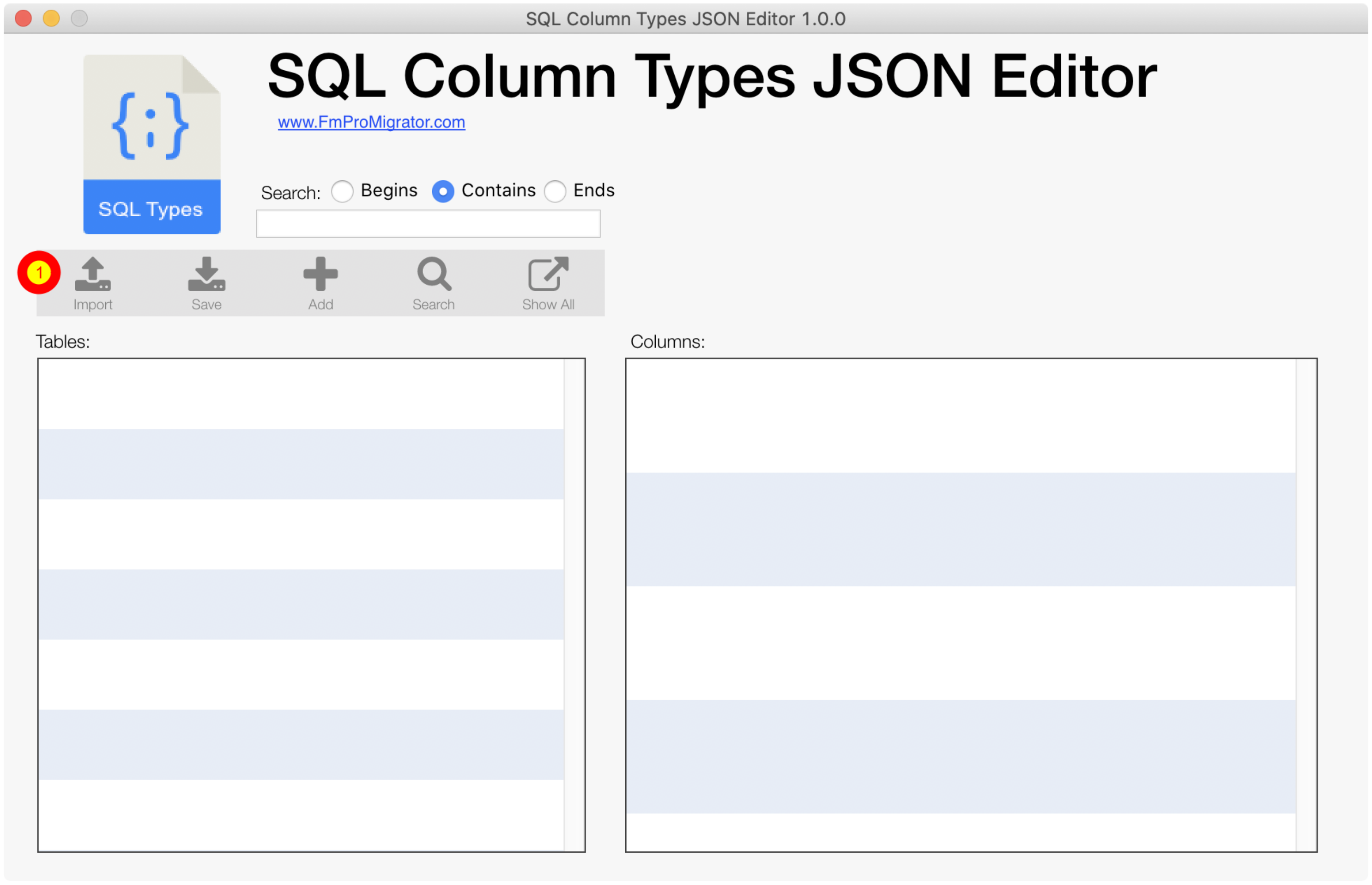
The SQL Column Types JSON Editor is available on the same web page where FmPro Migrator was downloaded (not the Trial version). This URL is provided with your email purchase receipt.

SQL Column Types JSON Editor Overview


SQL Column Types JSON Editor 1.0.0
SQL Types
SQL Column Types JSON Editor
www.FmProMigrator.com
Search: [✓ Begins] Contains Ends
Import
Save
Add
Search
Show All
Tables:
Columns:

The starting point for using the SQL Column Types JSON Editor is the button bar near the top of the window.

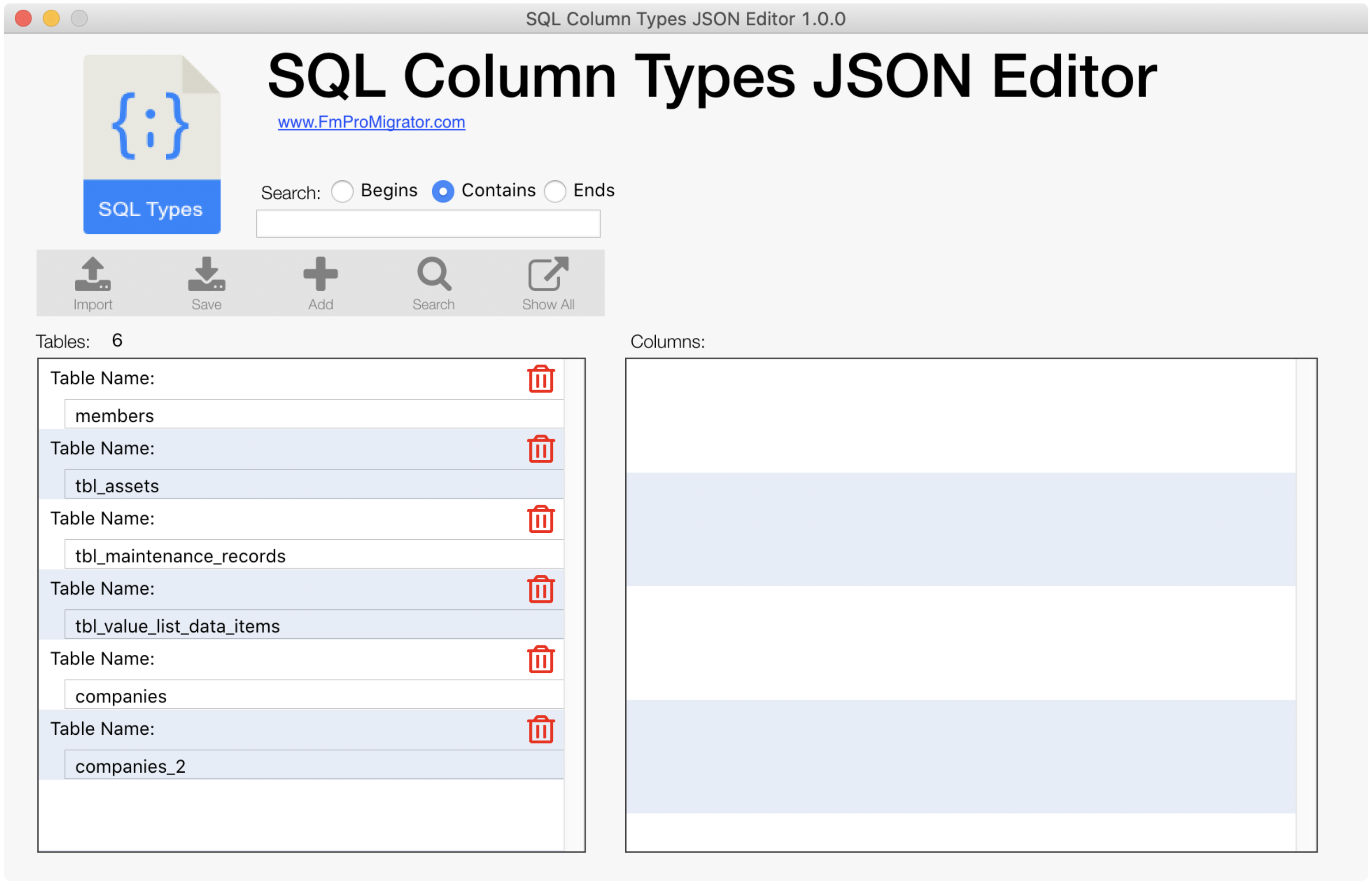
Importing the SQLColumnTypes.JSON File



Click (1) the Import button to import the SQLColumnTypes.JSON file.

Note: No data validation is performed when reading the JSON file. If the a random file is imported by accident, no data will be displayed in the Tables data grid on the left side, but rows of empty data will be displayed.

Import Results


SQL Column Types JSON Editor 1.0.0

SQL Column Types JSON Editor

www.FmProMigrator.com

Search: Begins [✓ Contains] Ends

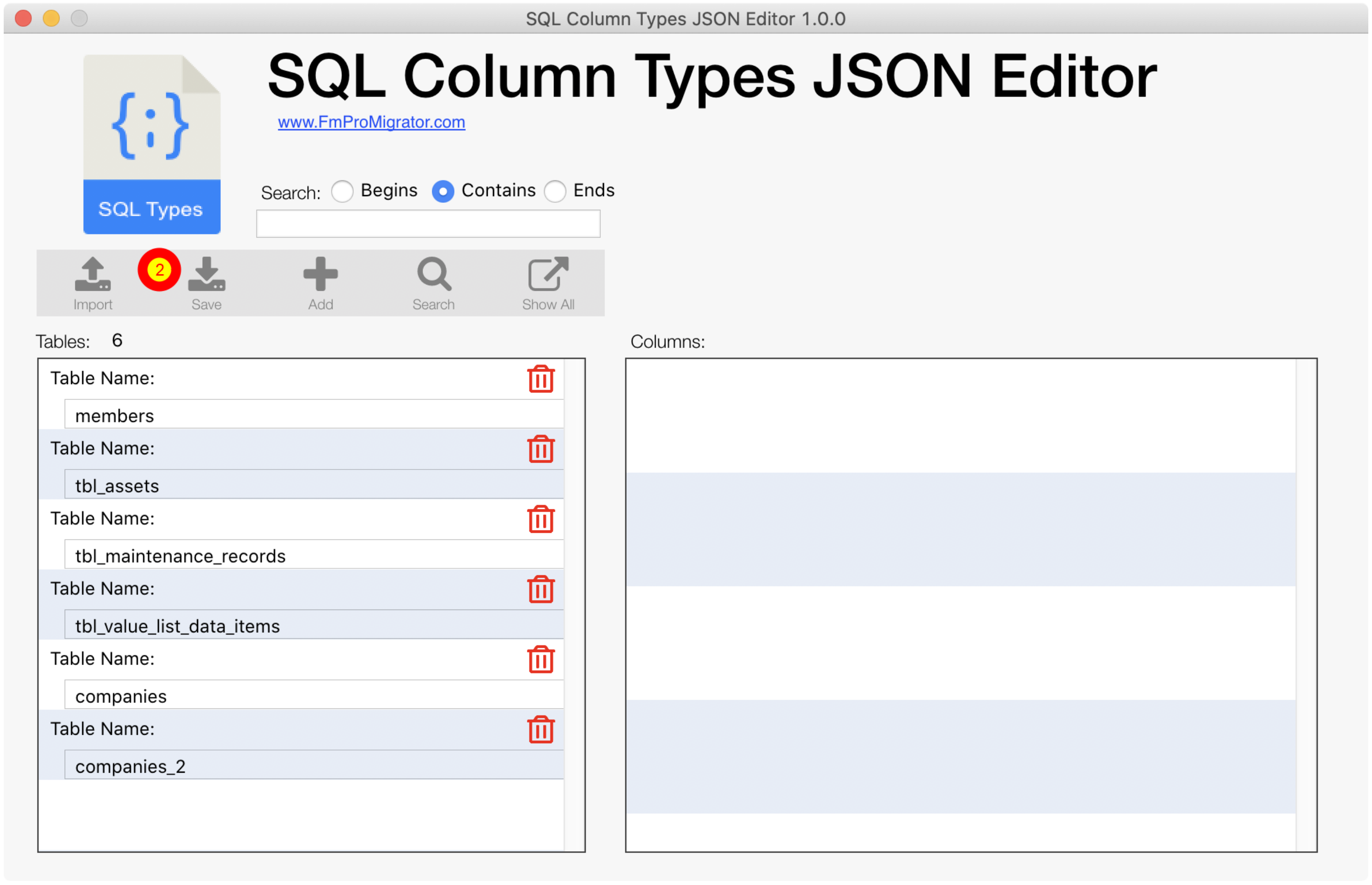
Tables: 6

Table Name: members
Table Name: tbl_assets
Table Name: tbl_maintenance_records
Table Name: tbl_value_list_data_items
Table Name: companies
Table Name: companies_2

Columns:

Once the SQLColumnTypes.JSON file has been imported, the names of each table will be shown in the Tables data grid on the left side.
The count of tables is displayed above the top of the grid.

Saving the Relationships.JSON File


SQL Column Types JSON Editor 1.0.0

SQL Column Types JSON Editor

www.FmProMigrator.com

Search: ( ) Begins [✓ Contains] ( ) Ends

Tables: 6

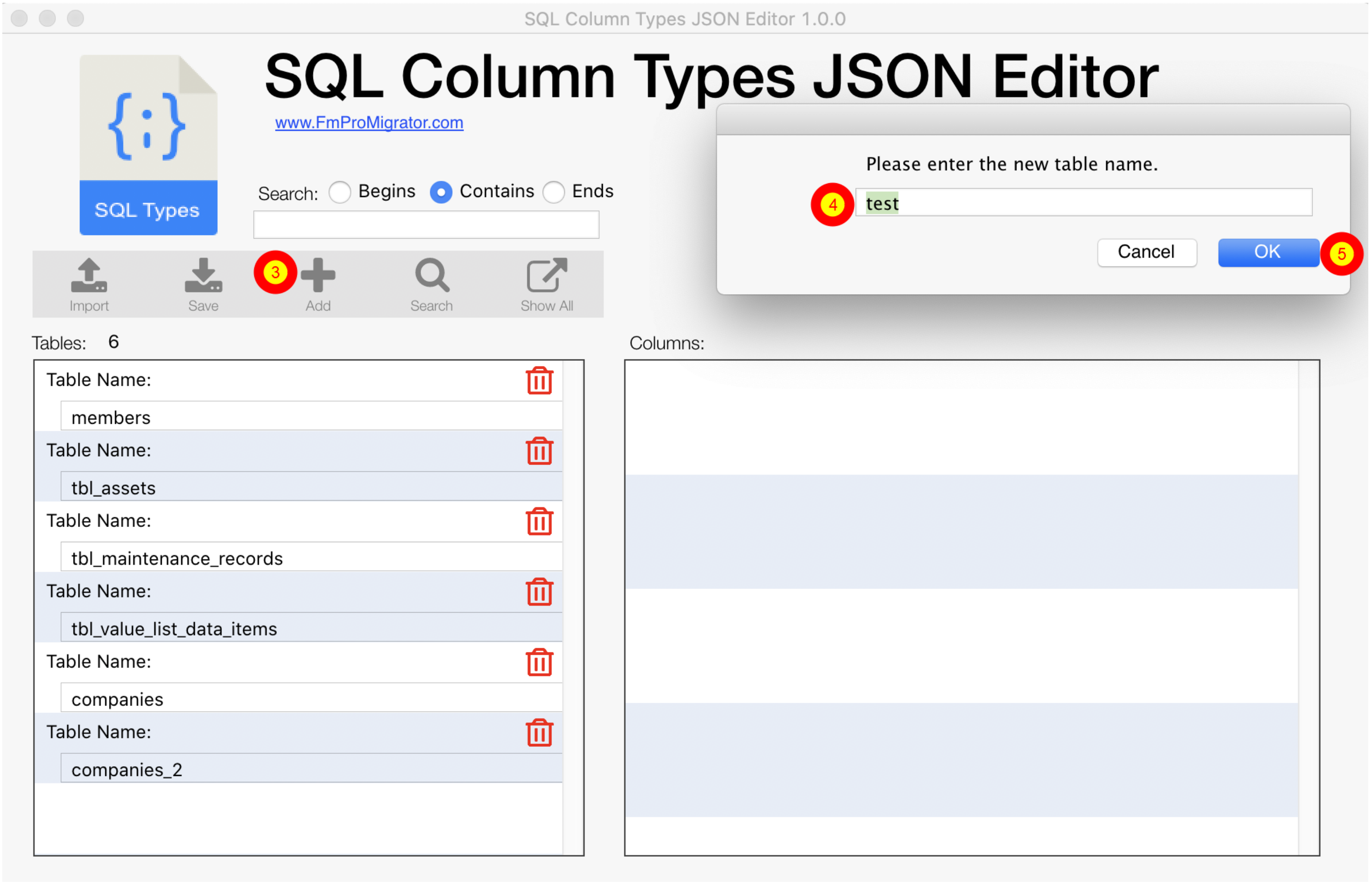
Columns:

Table Name:	members
Table Name:	tbl_assets
Table Name:	tbl_maintenance_records
Table Name:	tbl_value_list_data_items
Table Name:	companies
Table Name:	companies_2

Click the (2) Save button to save the tables (including column info) if editing has been done. All editing occurs in memory, and the data only gets saved by clicking the save button and entering a filename. Since the file is a text file, it can be copied to change management systems as needed.

After the SQLColumnTypes.JSON file has been updated, it should be re-imported back into the SQ_Library by clicking the loadSQLColumnsJSON button on the 1st card of the SQL_Library stack.

Add New Table



Please enter the new table name.

test

Tables: 6

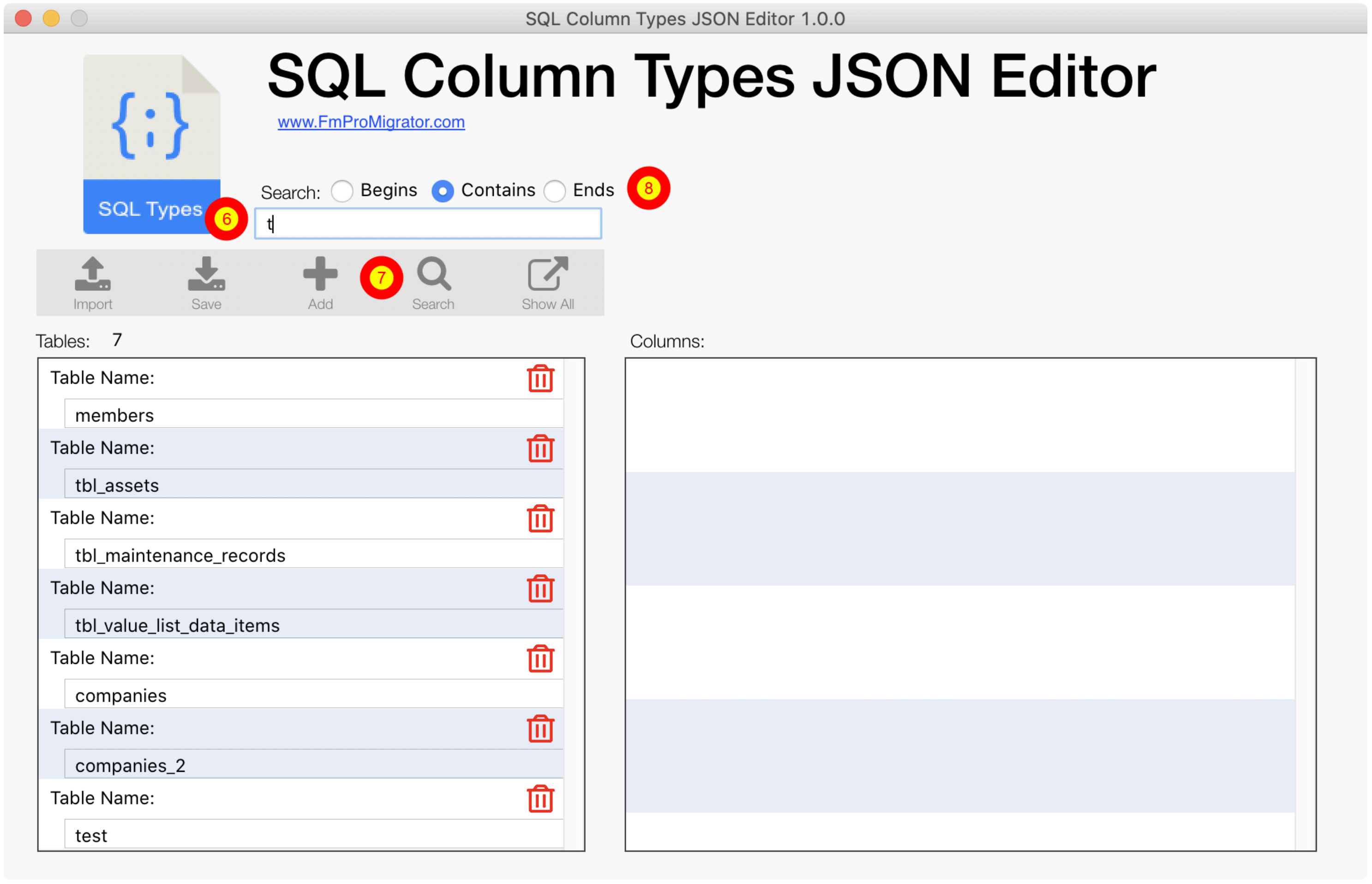
Columns:

Table Name:	members
Table Name:	tbl_assets
Table Name:	tbl_maintenance_records
Table Name:	tbl_value_list_data_items
Table Name:	companies
Table Name:	companies_2

Click the (3) Add button to add a new table, (4) enter the new table name in the dialog box then (5) click the Ok button. The new table with 0 columns will be added to the list of tables.

Note: Table names cannot be edited here, because the table name is effectively serving as a primary key for the data. It is possible to add and delete table names, then add columns individually.
Alternatively, it would be possible to edit the table name within the SQLColumnTypes.JSON before importing. The same change would need to be made within the SQL database being used to store the data.

Searching Table Names


SQL Column Types JSON Editor 1.0.0

SQL Column Types JSON Editor

www.FmProMigrator.com

Search: ( ) Begins (● Contains) ( ) Ends

t

Tables: 7

Columns:

Table Name: members
Table Name: tbl_assets
Table Name: tbl_maintenance_records
Table Name: tbl_value_list_data_items
Table Name: companies
Table Name: companies_2
Table Name: test

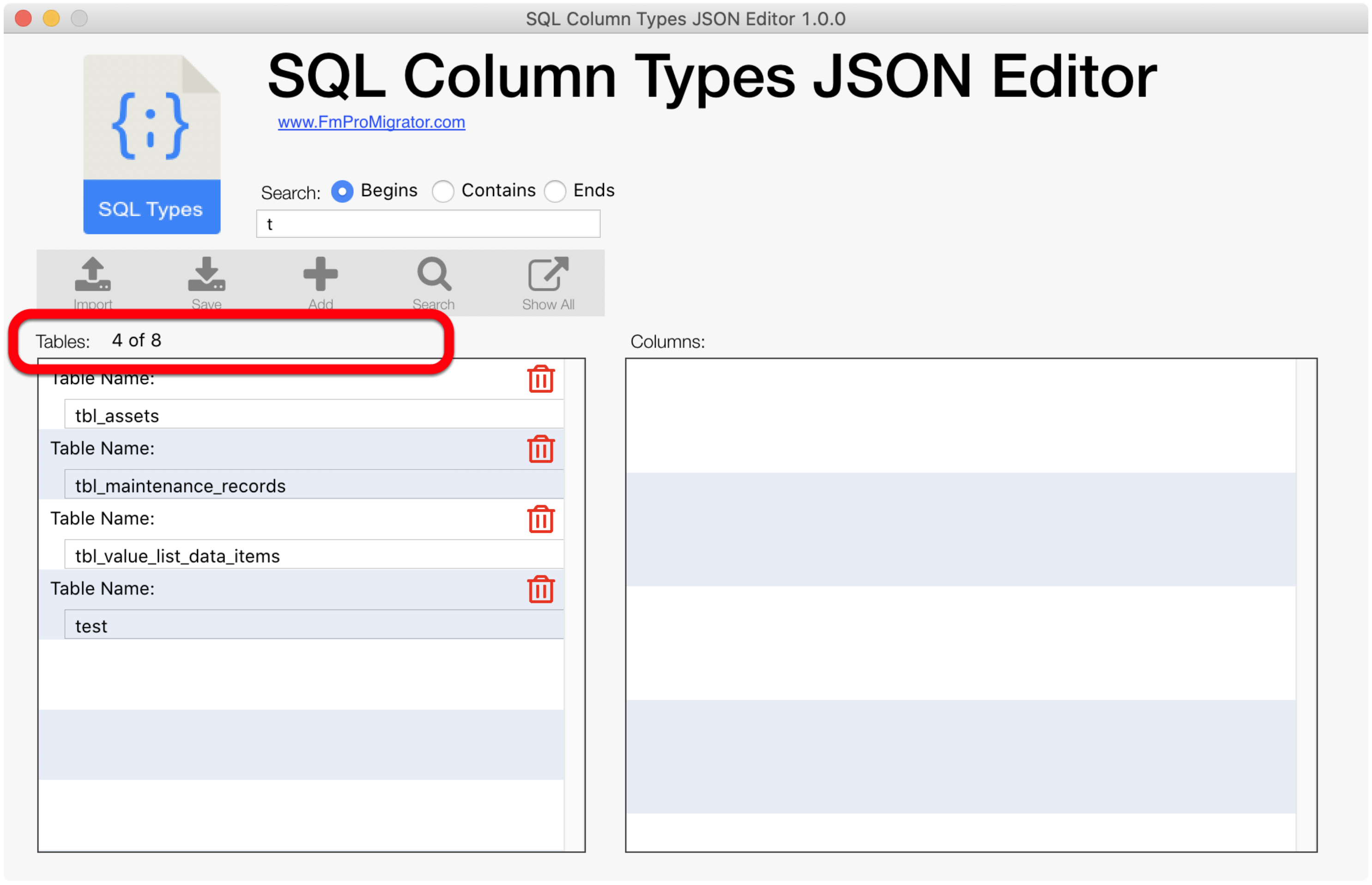
To search thru the table names, enter search text (6) within the search field, (7) click the Search button or (8) click the search type radio buttons. The search will be performed and only the matching records will be displayed in the Tables grid.

To perform a search, any of the following actions will start the search:
1] Type search text in the search field, then tab out of the field or click the background.
2] Click the Search button.
3] Click on any of the search type buttons Begins, Contains, Ends above the search field.

(8) The search type radio buttons perform the search as follows:
Begins: Searches the table names and matches the ones which begin with the search text.
Contains: Searches the table names and matches the ones which contain the search text.
Ends: Searches the table names and matches the ones which end with the search text.

During testing, even JSON files having over 1000 items return results in less than 1 second on modern machines.

Search Results



SQL Column Types JSON Editor

www.FmProMigrator.com

Search: [✓ Begins] Contains Ends

t

Tables: 4 of 8

Columns:

Table Name: tbl_assets

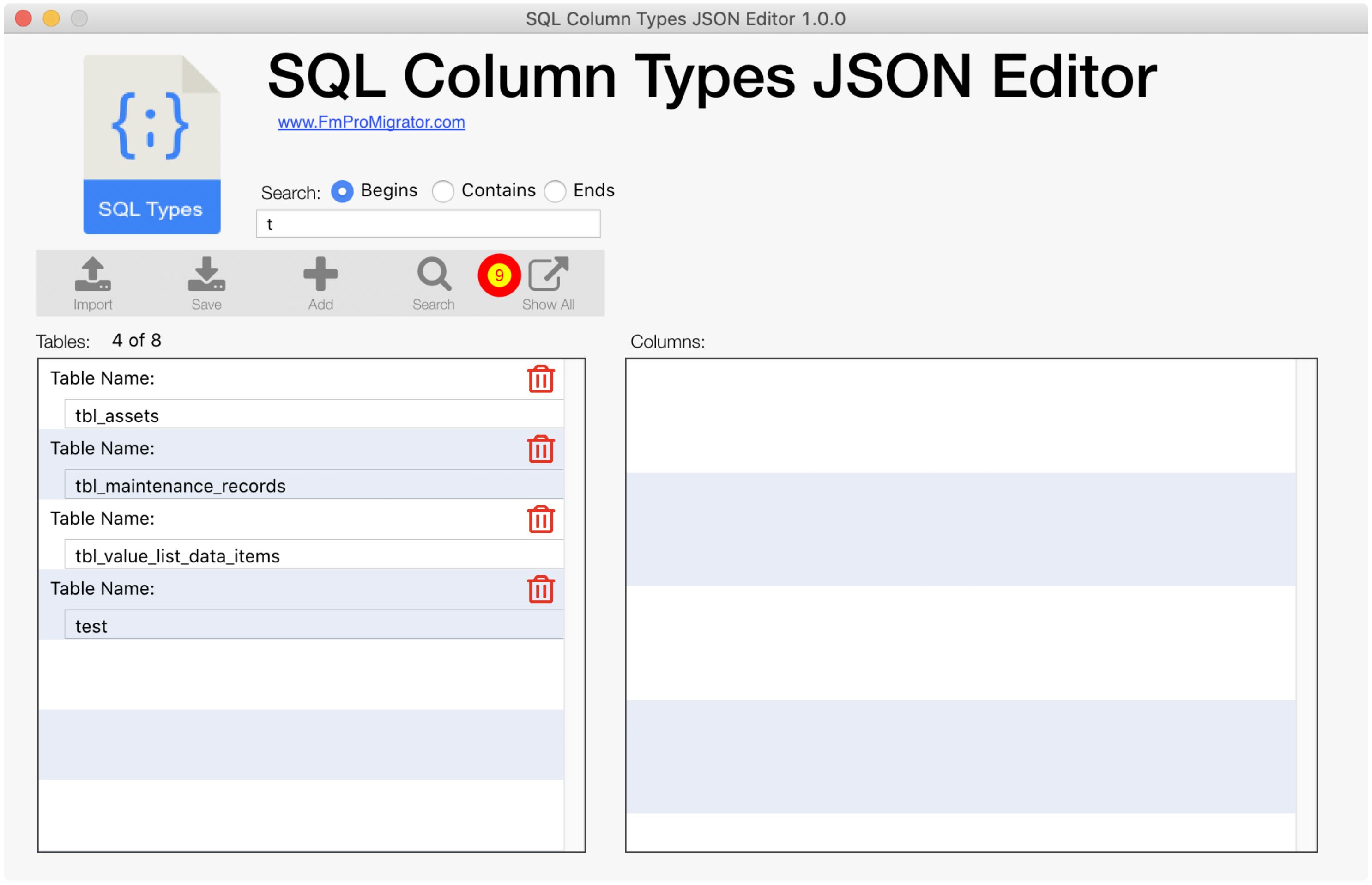
Table Name: tbl_maintenance_records

Table Name: tbl_value_list_data_items

Table Name: test

Searching for tables displays a search results label showing the number of found records, followed by the total number of tables being searched.

Show All - Search Results



SQL Column Types JSON Editor

www.FmProMigrator.com

Search: [✓ Begins] Contains Ends

t

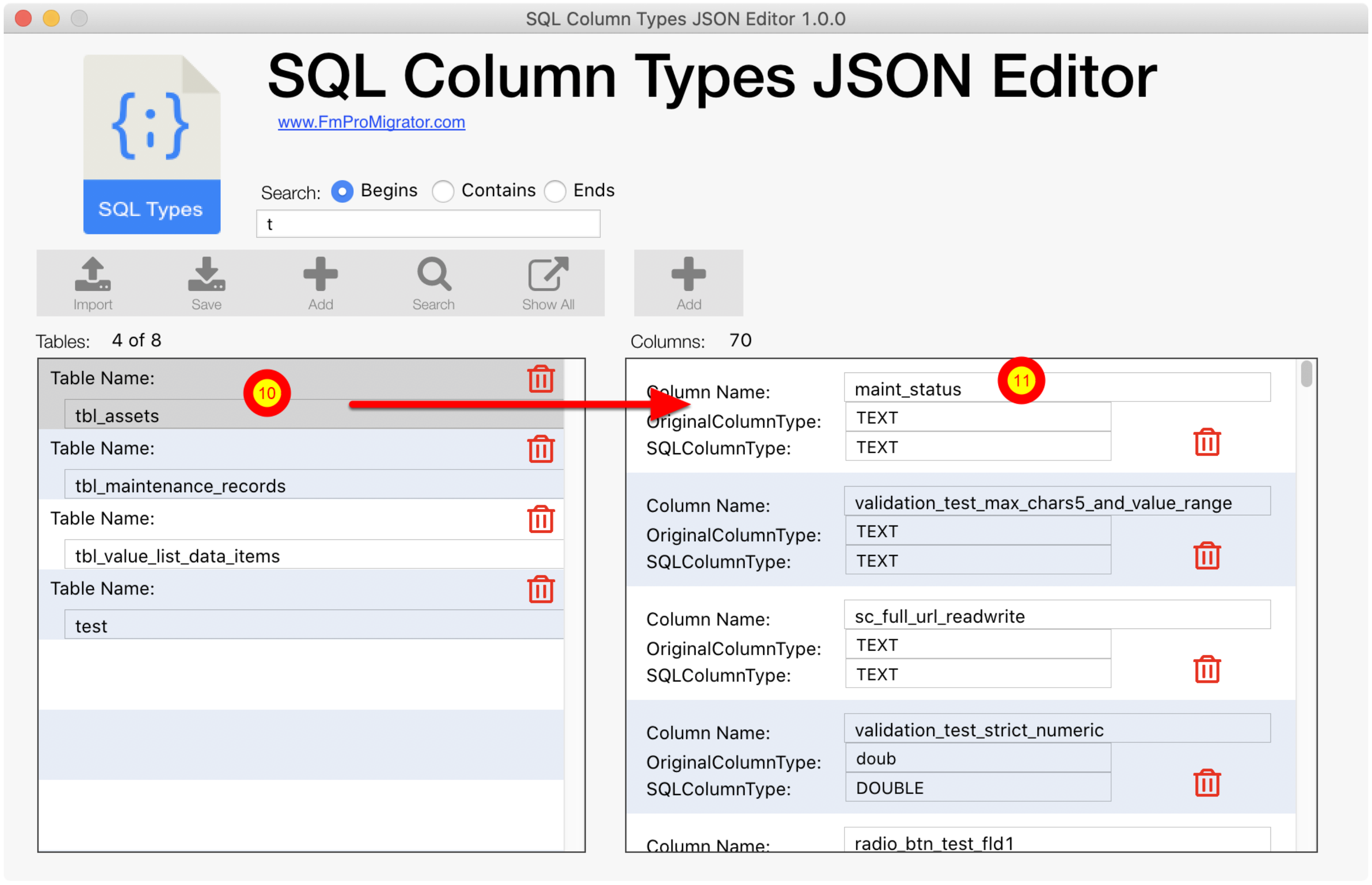
Tables: 4 of 8

Columns:

Table Name: tbl_assets
Table Name: tbl_maintenance_records
Table Name: tbl_value_list_data_items
Table Name: test

Clicking the (9) Show All button results in the display of all tables.

Editing Table Column Information


SQL Column Types JSON Editor 1.0.0
SQL Column Types JSON Editor
www.FmProMigrator.com
SQL Types
Search: [✓ Begins] Contains Ends
t
Import
Save
Add
Search
Show All
Add
10
11

Tables: 4 of 8

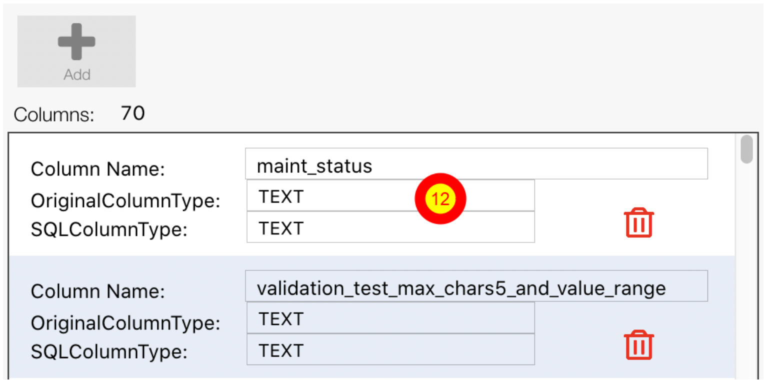
Table Name:	tbl_assets
Table Name:	tbl_maintenance_records
Table Name:	tbl_value_list_data_items
Table Name:	test

Columns: 70

Column Name	OriginalColumnType	SQLColumnType
maint_status	TEXT	TEXT
validation_test_max_chars5_and_value_range	TEXT	TEXT
sc_full_url_readwrite	TEXT	TEXT
validation_test_strict_numeric	doub	DOUBLE
radio_btn_test_fld1		

Clicking (10) any row of the list of tables grid will display detailed about for the columns in the table (11), in the Columns grid on the right.

Editing Table Column Info



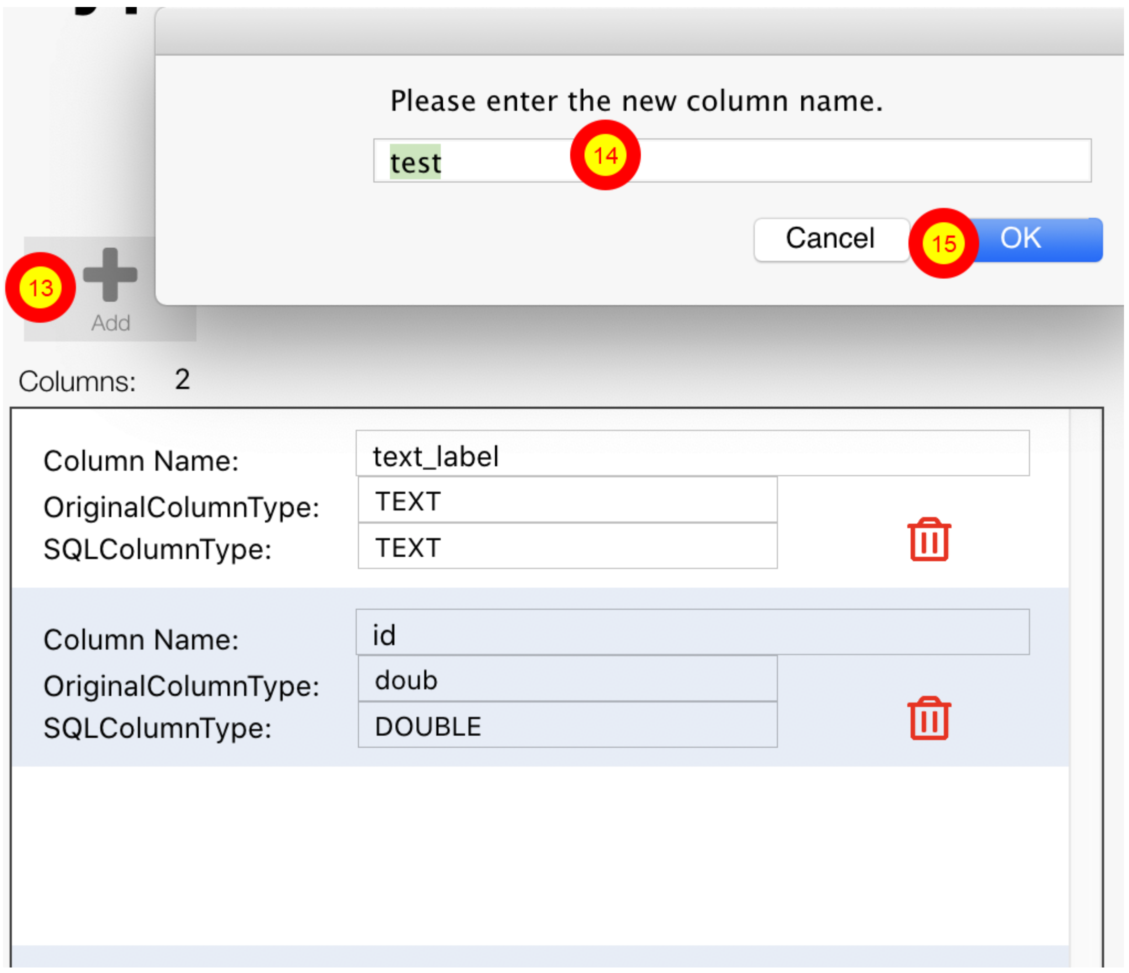
Columns: 70

Column Name	OriginalColumnType	SQLColumnType
maint_status	TEXT	TEXT
validation_test_max_chars5_and_value_range	TEXT	TEXT

Click on any field (12) in the Columns grid to edit its field contents. Changes are saved into memory automatically, but saving to disk only occurs when clicking the top-level Save button.

<u>Note</u>: The Column Name effectively serves as the primary key for the columns info, so the name of a column cannot be edited. If this type of change needs to be made, add a new column with a new name and delete the original one.

Adding a Column

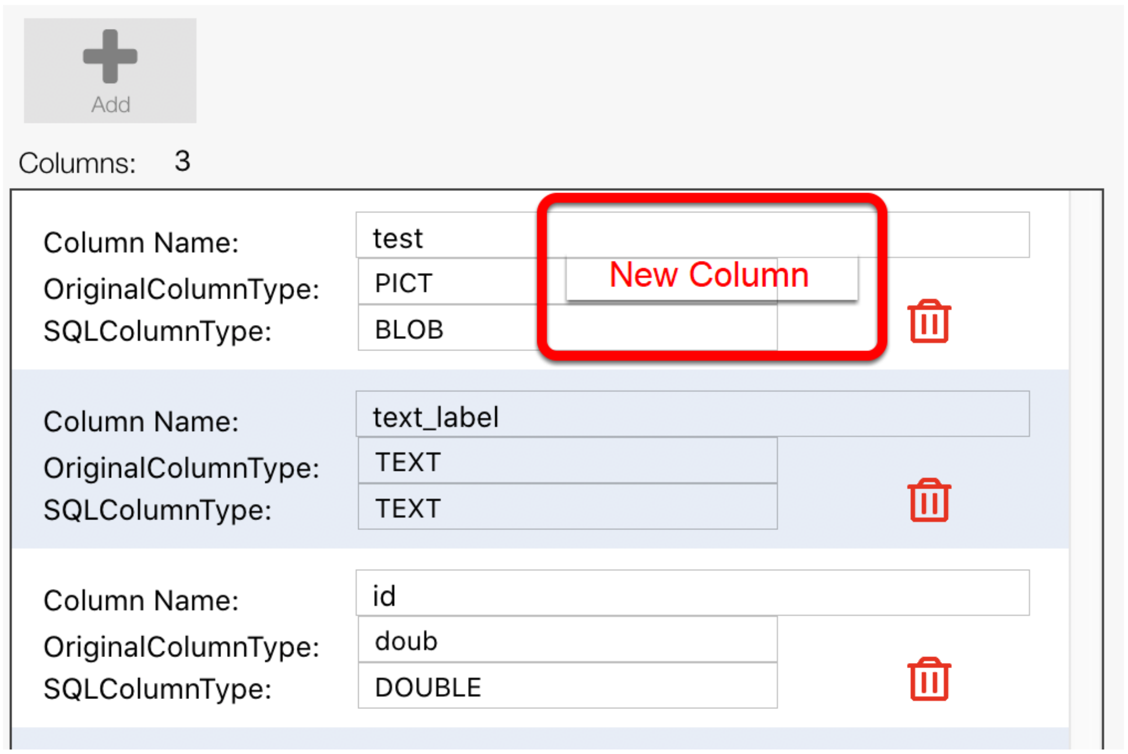


Columns: 2

Column Name:	text_label
OriginalColumnType:	TEXT
SQLColumnType:	TEXT
Column Name:	id
OriginalColumnType:	doub
SQLColumnType:	DOUBLE

Click (13) the Add button above the Columns grid (14) enter a name, then (15) click the Ok button to add a new column record having default values. The record will be created immediately and added to the grid.

New Table Column



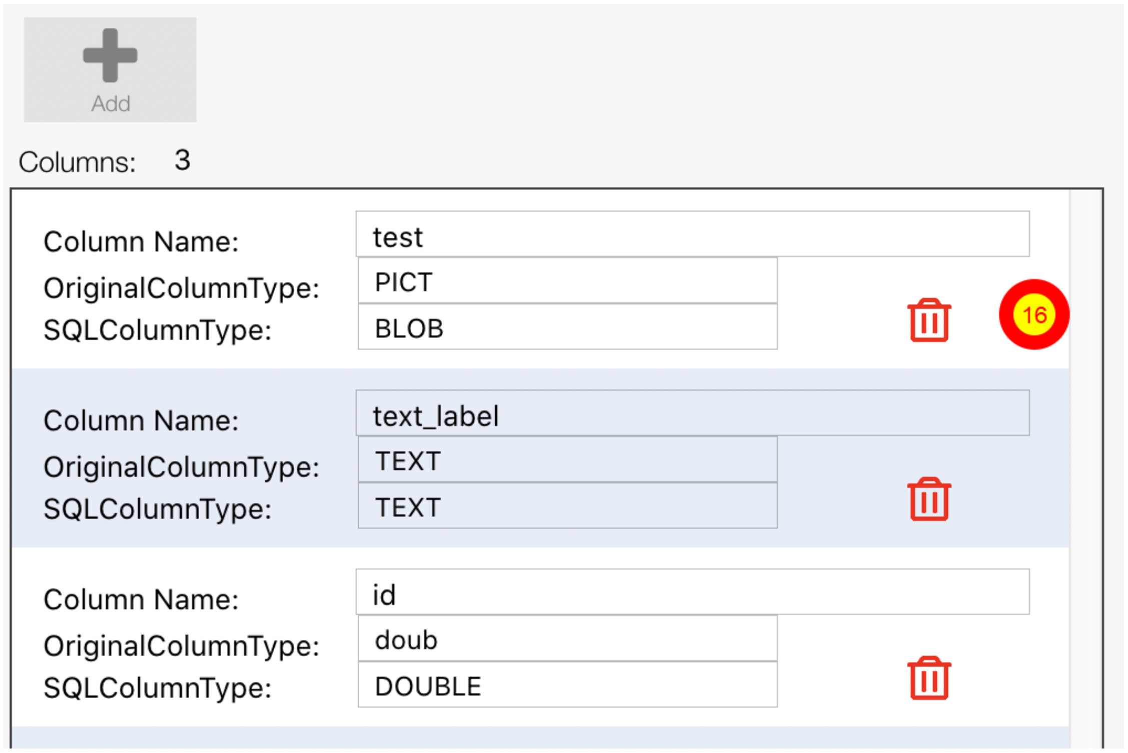
Columns: 3

Column Name:	test
OriginalColumnType:	PICT
SQLColumnType:	BLOB
Column Name:	text_label
OriginalColumnType:	TEXT
SQLColumnType:	TEXT
Column Name:	id
OriginalColumnType:	doub
SQLColumnType:	DOUBLE

Click in each of the fields to edit the info for the column.

<u>**Note**</u>: As shown in this screenshot, when dealing with images, the OriginalColumnType field should contain the text "PICT", which usually translates to the "BLOB" column type for a SQL database. The code in the SQL Library is only looking at the OriginalColumnType text of "PICT" to determine whether a database column contains binary data.

Deleting a Table Column



Columns: 3

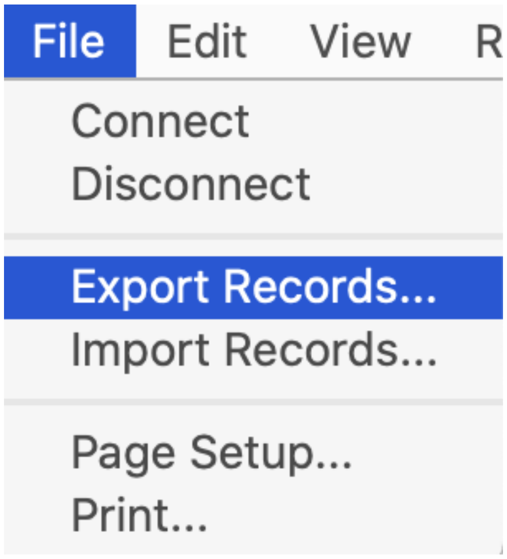
Column Name:	test
OriginalColumnType:	PICT
SQLColumnType:	BLOB
Column Name:	text_label
OriginalColumnType:	TEXT
SQLColumnType:	TEXT
Column Name:	id
OriginalColumnType:	doub
SQLColumnType:	DOUBLE

Click the (16) trash can icon at the right side of the record to delete an individual table column.

Using the Converted Application

Export Records

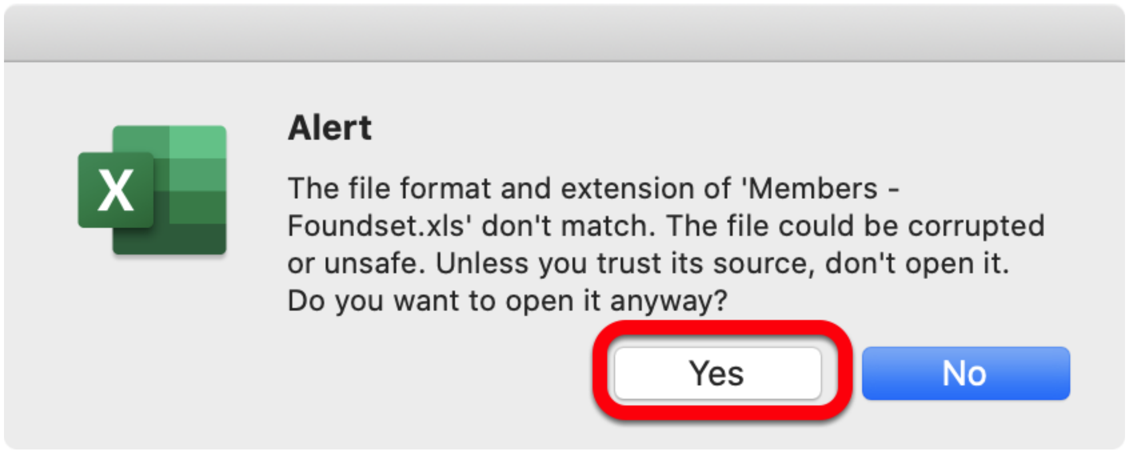
File -> Export Records... Menu



The Export Records... menu item in the File menu exports records from the current card and database table as TAB delimited data into a file having an .XLS extension. The file will then be opened automatically in Excel after it has been saved.

Data will be exported from database columns for the table associated with the layout/form - not related tables. BLOB columns aren't supported for data import/export purposes.

Excel Warning Message

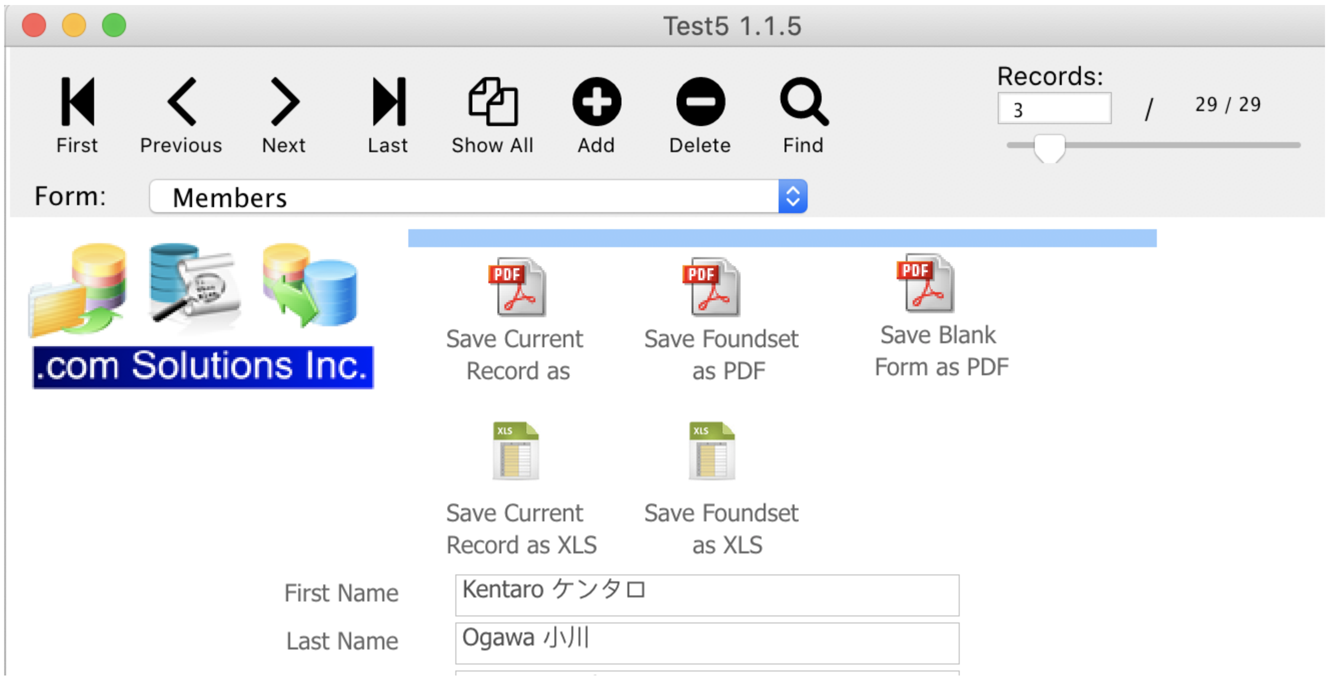


Alert

The file format and extension of 'Members - Foundset.xls' don't match. The file could be corrupted or unsafe. Unless you trust its source, don't open it. Do you want to open it anyway?

Excel warns that the file is not really a spreadsheet file, but it knows how to open it due to the contents consisting of .TAB delimited data. Just click the Yes button to open the file. Then the file should be saved using the latest Excel file format (i.e. .xlsx) for instance.

Exported UTF Characters

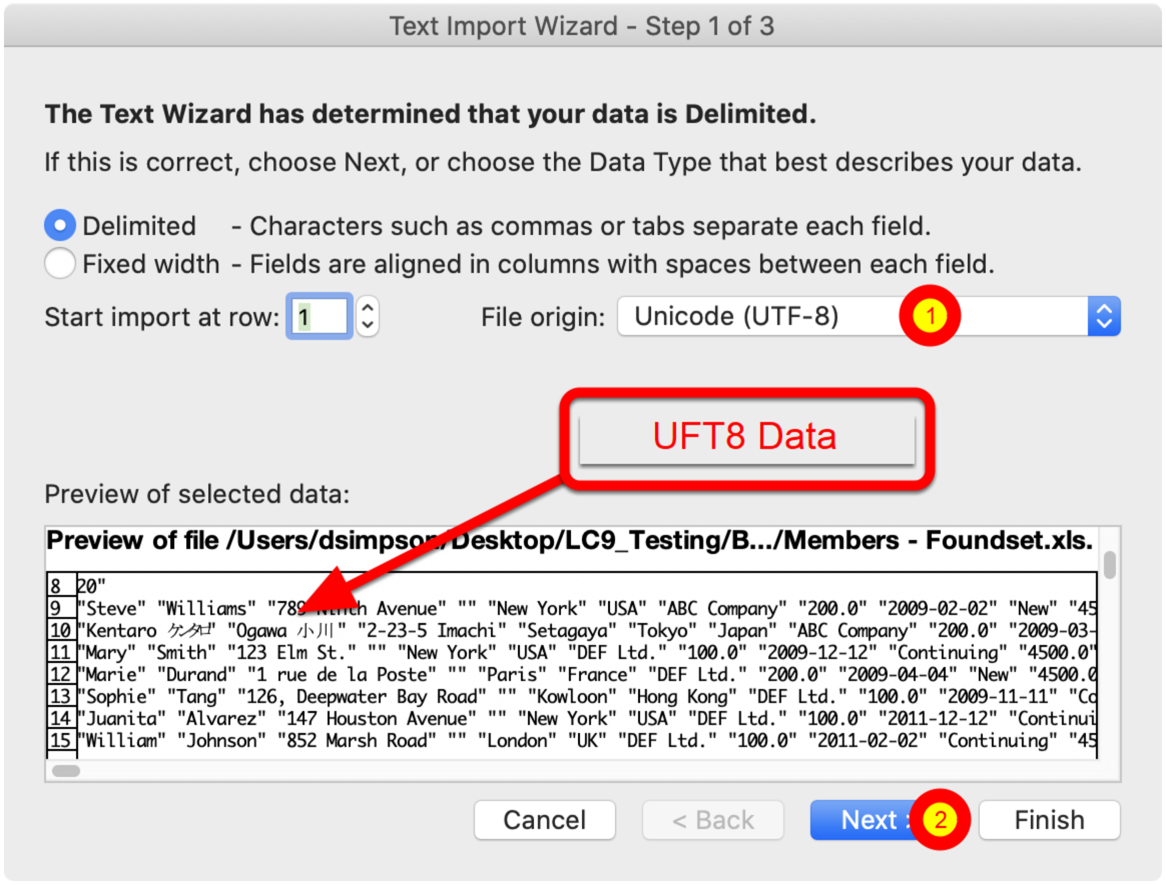


First Name: Kentaro ケンタロ

Last Name: Ogawa 小川

Exported data containing UTF-8 characters requires a slightly different opening process in order for Excel to properly recognize the file encoding. By default, Excel imports data using the native character set of the operating system, which causes UTF-8 characters to be improperly read.

Importing UTF-8 Character Data



Text Import Wizard - Step 1 of 3

The Text Wizard has determined that your data is Delimited.

If this is correct, choose Next, or choose the Data Type that best describes your data.

[✓ Delimited] - Characters such as commas or tabs separate each field.
Fixed width - Fields are aligned in columns with spaces between each field.

Start import at row: 1   File origin: Unicode (UTF-8)

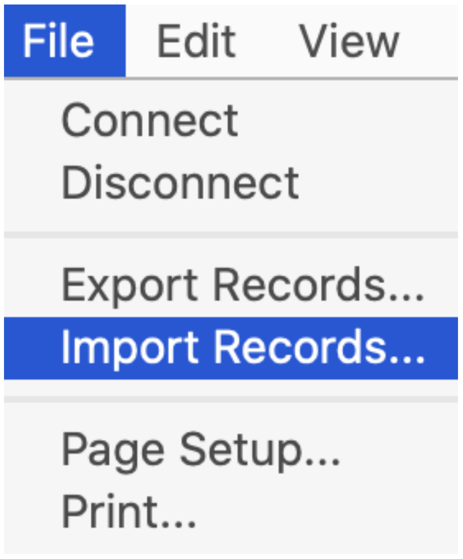
Preview of selected data:

Preview of file /Users/dsimpson/Desktop/LC9_Testing/B.../Members - Foundset.xls.

```
8  20"
9  "Steve" "Williams" "789 Ninth Avenue" "" "New York" "USA" "ABC Company" "200.0" "2009-02-02" "New" "45
10 "Kentaro ケンタロ" "Ogawa 小川" "2-23-5 Imachi" "Setagaya" "Tokyo" "Japan" "ABC Company" "200.0" "2009-03-
11 "Mary" "Smith" "123 Elm St." "" "New York" "USA" "DEF Ltd." "100.0" "2009-12-12" "Continuing" "4500.0"
12 "Marie" "Durand" "1 rue de la Poste" "" "Paris" "France" "DEF Ltd." "200.0" "2009-04-04" "New" "4500.0
13 "Sophie" "Tang" "126, Deepwater Bay Road" "" "Kowloon" "Hong Kong" "DEF Ltd." "100.0" "2009-11-11" "Co
14 "Juanita" "Alvarez" "147 Houston Avenue" "" "New York" "USA" "DEF Ltd." "100.0" "2011-12-12" "Continui
15 "William" "Johnson" "852 Marsh Road" "" "London" "UK" "DEF Ltd." "100.0" "2011-02-02" "Continuing" "45
```

Select Open from the File menu within Excel to open the file, then select (1) the UTF-8 encoding from the File origin menu.
As soon as UTF-8 is selected from the menu, the Preview field will show that the data has been decoded properly, so you can click the (2) Finish button.

File -> Import Records... Menu

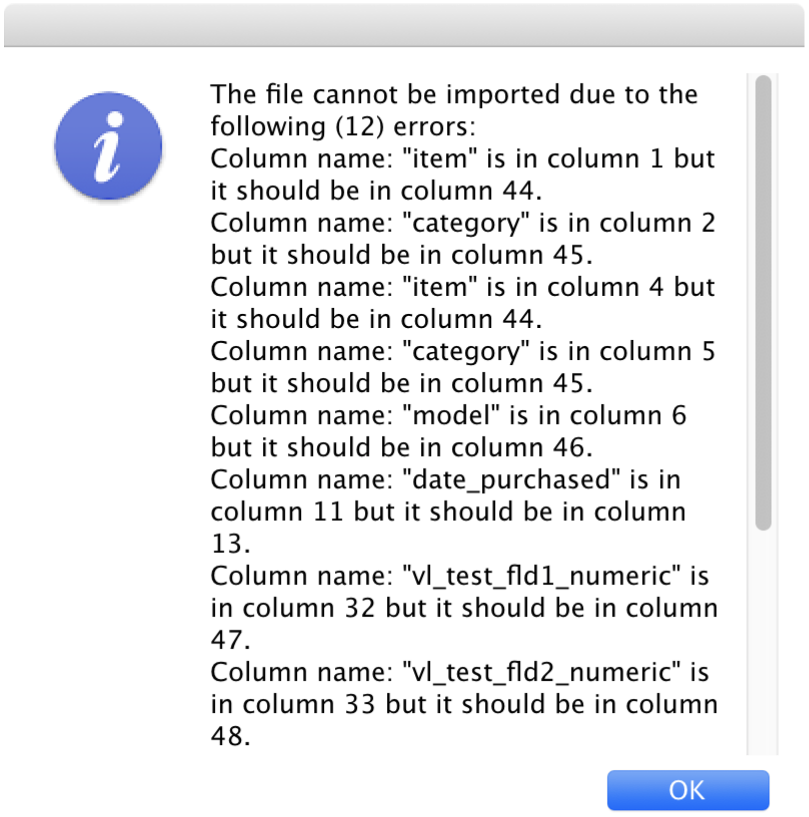


The Import Records... menu imports TAB delimited text files having the .TAB or .TXT file extension. The column names are checked to make sure they they match the fields converted from the original database. To see the format, perform a file export first, in order to verify the order of the columns.

Primary key columns are skipped on import, since these are already configured for auto-enter incrementing to the next higher serial number for each new record.

Data will be imported into database columns for the table associated with the layout/form - not related tables. BLOB columns aren't supported for data import/export purposes.

Duplicated Column Names Error



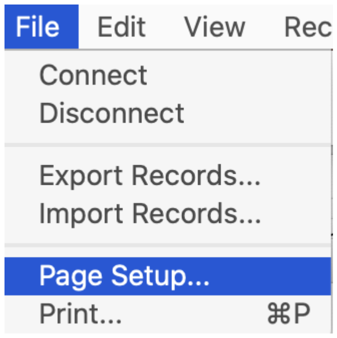
```
The file cannot be imported due to the
following (12) errors:
Column name: "item" is in column 1 but
it should be in column 44.
Column name: "category" is in column 2
but it should be in column 45.
Column name: "item" is in column 4 but
it should be in column 44.
Column name: "category" is in column 5
but it should be in column 45.
Column name: "model" is in column 6
but it should be in column 46.
Column name: "date_purchased" is in
column 11 but it should be in column
13.
Column name: "vl_test_fld1_numeric" is
in column 32 but it should be in column
47.
Column name: "vl_test_fld2_numeric" is
in column 33 but it should be in column
48.
```

Asource database file may contain multiple copies of the same field on a layout or form. This situation will prevent importing of records because the columns have been duplicated.
For data import purposes, it is recommended that layouts/forms should be designed with only one copy of the fields needing to be imported.

Printing

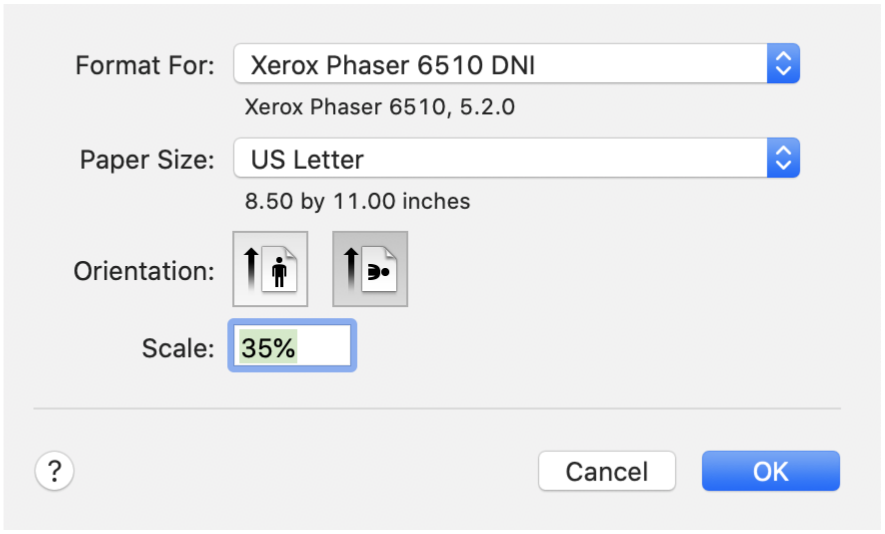
The LiveCode stack created by FmPro Migrator includes Page Setup and Printing to printer and PDF output options.
These menu options are used to print from any of the cards of the stack which have been created by FmPro Migrator, except the Login card (since the login card contains no data or database fields).

Page Setup Menu



The Page Setup.. menu displays the standard Page Setup dialog provided by the operating system.

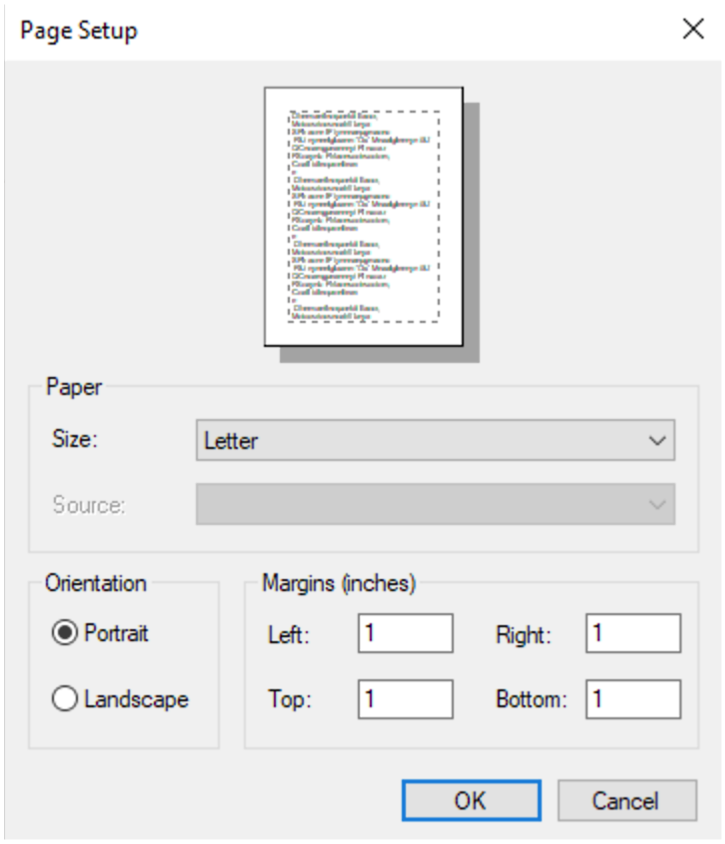
Page Setup Dialog - macOS


Format For: Xerox Phaser 6510 DNI
Xerox Phaser 6510, 5.2.0
Paper Size: US Letter
8.50 by 11.00 inches
Orientation:
Scale: 35%

From within the Page Setup dialog, the name of the printer, paper size, paper orientation and print scale can be set.
The selected page setup info is stored within the application preferences JSON file and is used during printing.

Note: Cards in the converted stack which are too large to fit on a singe page should be scaled using the Scale percentage field of the Page Setup dialog. The page scale factor is utilized for printer output and also for PDF output. The page setup settings are read and used to set the scaling during PDF output to match the printer output.

Page Setup - Windows



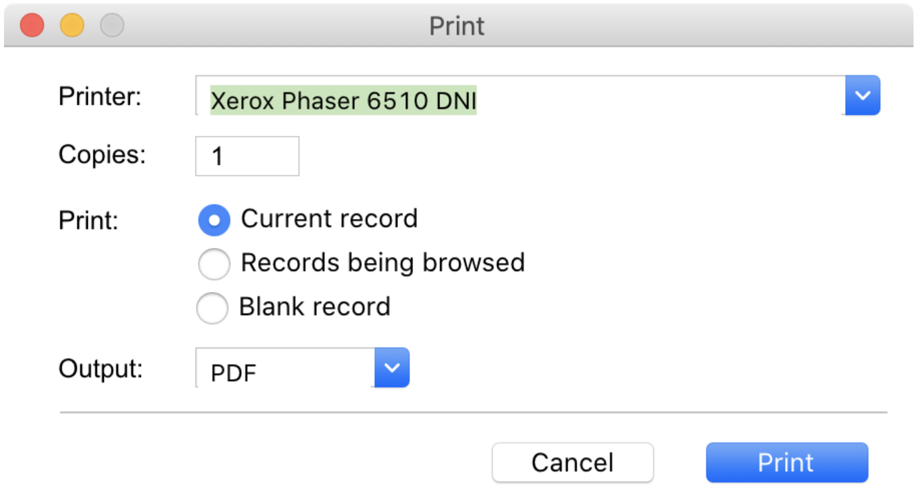
Size: Letter
Source:
Margins (inches)
Portrait [✓]
Landscape
Left: 1
Right: 1
Top: 1
Bottom: 1

From within the Page Setup dialog, the name of the printer, paper size, paper orientation and print scale can be set.
The selected page setup info is stored within the application preferences JSON file and is used during printing.

Note: Cards in the converted stack which are too large to fit on a singe page cannot be scaled on Windows. On Windows, it is best to either select a larger paper size or reduce the amount of info on the card so that it will fit.

Idea: Inside the stack level printMenuPrintPages handler the printScale factor could be updated programmatically to specify a print scale factor. This could be set via the Print dialog by adding a new field for Print Scale and passing this value thru to the printMenuPrintPages handler via the gAppPrefsArray.

Print Dialog



Printer: Xerox Phaser 6510 DNI

Copies: 1

Print: [✓ Current record] Records being browsed Blank record

Output: PDF

The modal PrintDialog substack is used to print cards of the converted stack file. This dialog is the same on macOS and Windows.

When clicking the Print button, the results are stored within the gAppPrefsArray and read from this global array by the printMenuPrintPages handler.

Supported features include:
Printing the current record.
Printing the records being browsed (including support for found-sets).
Printing a blank record.

Each of these output options can be printed to a PDF file or directly to the selected printer.