

.com Solutions Inc.



FmPro Migrator - BASIC to LiveCode Conversion Procedure

FmPro Migrator - BASIC to LiveCode Conversion Procedure

1 BASIC to LiveCode Conversion

- 1.1 Introduction - BASIC to LiveCode Conversion 4
- 1.2 Step 1 - Create FmPro Migrator Project File 8
- 1.3 Step 2 - Select Conversion Options & Convert BASIC Files 12

2 VB6 to LiveCode Conversion

- 2.1 VB6 to LiveCode - Form and Script Conversion 18

BASIC to LiveCode Conversion

Introduction - BASIC to LiveCode Conversion

This document provides an explanation of the steps required to convert BASIC scripts to LiveCode scripts using FmPro Migrator Platinum Edition.

This document also includes VB6 to LiveCode conversion info, showing how to convert the VB6 .frm files and .bas script files into LiveCode stacks.

Revision 03

9/7/2013

[Updated LiveCode graphics screenshots.]

About the BASIC to LiveCode Conversion Process

The BASIC to LiveCode conversion process is designed to convert all of the BASIC files within a source directory, including all subdirectories. FmPro Migrator is designed to read BASIC files having a variety of file extensions (including: bas, vba, vbs or txt) during the conversion process.

Each file within the source directory is read into memory and analyzed on a line by line basis. Keywords and operators are read and converted to the equivalent keywords and operators in LiveCode.

BASIC Code Processing Features

- 1) Traversal of the files and subdirectories of the selected source directory. Re-creation of the same file and directory structure within the selected destination directory. Quickly convert all of the .bas files within the source directory and subdirectories.
- 2) Processing of .bas, .vba, .vbs and .txt source files, including support for Visual Basic, PowerBasic, ZBASIC/FutureBasic, RealBasic, VBScript and VisualBasic for Applications code.
- 3) High performance processing. Process hundreds of files and hundreds of thousands of lines of code in seconds.
- 4) Code indenting is maintained for most situations. The exception is CASE statements where additional Default statements get added.
- 5) Line continuation characters are removed, as part of the code parsing process.
- 6) Compiler directives starting with "\$" are commented out.
- 7) DIM/STATIC commands are converted into local/global variable definitions in LiveCode, in which the "As <VariableType>" definitions are removed and the static memory quantity is also removed, since it isn't needed in LiveCode. For instance the statement
DIM ParameterData2\$(3750)
gets changed to
local ParameterData2
- 8) BASIC variable suffix characters (% , \$) are removed from most instructions.
- 9) Each variable definition is checked to insure that it doesn't exactly match an existing LiveCode

- keyword. If a conflict exists between a variable name and an existing keyword, then an underscore character is added to the variable name.
- 10) Variable assignments are converted into LiveCode "put" commands. Colon or single quote comment operators following the assignment on the same line will be commented using the "--" LiveCode operator and placed to the right side of the assignment statement. Variable suffixes will be removed and the variable names checked against the LiveCode keywords list on the left side of the assignment operator.
 - 11) Private Sub definitions are converted into LiveCode private command handlers with the "end sub" replaced with "end <command name>".
 - 12) Public Sub definitions are converted into LiveCode private command handlers with the "end sub" replaced with "end <command name>".
 - 13) OPEN/INPUT/CLOSE commands using file numbers (#1, #2 etc.) are partially converted into LiveCode open/read/close file commands. Manual changes will be required in order to specify
 - 14) The BASIC For loop keyword is converted into a "repeat with" keyword in LiveCode. The For loop variable is fully checked for LiveCode keyword conflicts and BASIC variable suffix characters are removed. BASIC variable suffix characters are removed from the remaining variables.
 - 15) BASIC code labels ending with a colon are commented out.
 - 16) ON Error statements are commented out, along with GOTO commands. Error checking can be rewritten using Try/Catch statements in LiveCode.
 - 17) Passing parameters by reference using the ByVal keyword are converted into the @ reference passing character used in LiveCode.
 - 18) Functions having optional arguments specified with the Optional keyword will have this optional keyword removed. If default values are needed for the parameters, these should be defined manually within the function.
 - 19) SELECT CASE statements are converted into LiveCode SWITCH statements. Each CASE statement is closed with a break statement. CASE ELSE statements are converted into "default" statements.
 - 20) The DO WHILE/DO UNTIL keywords are converted into a "repeat while/until" statements, with the closing LOOP keyword converted into an "end repeat" statement. The LOOP keyword is processed if it is the last word of an instruction containing other keywords and if it is on a line by itself.
 - 21) The LET keyword is removed and the instruction is processed as a standard assignment statement.
 - 22) The LONG IF keyword is processed the same as a regular IF keyword. (ZBASIC/FutureBasic).
 - 23) SLEEP <interval> is converted into "wait <interval> milliseconds with messages".
 - 24) Function return parameters specifying the function name will be replaced with the LiveCode return keyword.

BASIC Omitted Functions and Keywords

Some Basic functions are purposely omitted from the conversion processing because they require manual work:

- 1) Instr(variable1, ".") -> offset(".", variable1] - The string to find and string to search parameter positions need to be swapped manually when converting to LiveCode.
- 2) Basic "+" operators can be used to represent either a mathematical operation or string concatenation.
- 3) bin\$() -> binaryEncode() - The LiveCode formatting needs to be manually applied to the data to be converted into binary. (ZBASIC/FutureBasic)
- 4) box, circle, button -> Create an object using the appropriate style of specified object. (ZBASIC/FutureBasic)
- 5) hex\$() -> binaryDecode() - The LiveCode formatting needs to be manually applied to the data to be converted into hex. (ZBASIC/FutureBasic)
- 6) kill path\$ -> delete file <filepath> (ZBASIC/FutureBasic)
- 7) WAITKEY\$ - There isn't an equivalent command to wait for keystrokes entered via the command line, which is how commands like INPUT\$ and WAITKEY\$ are used. LiveCode is event driven, so messages such as entering keystrokes within fields, and tabbing between fields can be captured and used to trigger code to run at the appropriate time.

Unsupported Features Requiring Manual Conversion

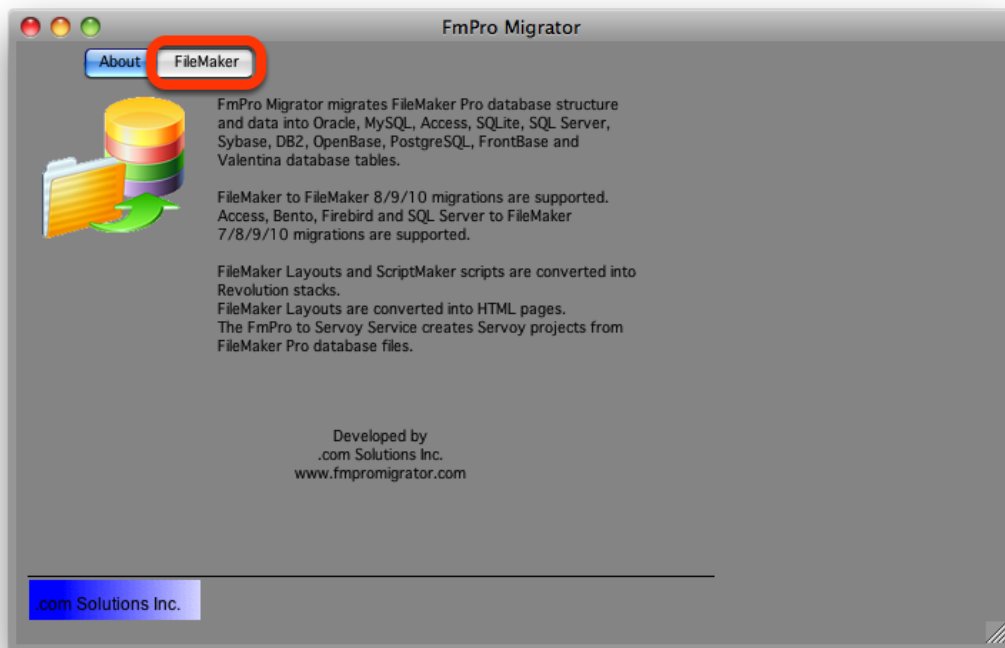
- 1) Poorly formed BASIC code will not get fully processed. For instance, keywords, operators and variables should generally be separated from each other by at least one space character.
- 2) One instruction per line can be parsed correctly. The additional instructions of BASIC code on the same line won't be completely converted. However commented code to the right of variable assignment statements will be commented appropriately.
- 3) Operating system specific or BASIC language specific functions won't be converted.
- 4) BASIC external libraries or calls to installed .dll files.
- 5) Syntax errors or other types of problems which prevent the BASIC code from executing won't be corrected.
- 6) BASIC object oriented code features won't be converted, but will remain in the converted code for reference purposes. For instance Button1.Caption won't be converted into "the name of button Button1". This is due to the difficulty of determining the type of object referenced in the original code (button, field, window etc.). **Note:** This dot notation conversion is done automatically for VB6 projects. See the VB6 chapter of this manual for more details.
- 7) Multiple variable assignments within the same instruction. These statements need to be separated into two separate instructions.
- 8) Automated conversion of twips to pixel coordinates between Visual Basic and LiveCode is not implemented. There are 1440 twips per inch and approximately 15 twips per pixel (depending upon screen resolution). LiveCode uses pixel based coordinates when defining object locations. **Note:** Twips to Pixel conversion is done for VB6 conversion projects. See the VB6 chapter of this manual for more details.

- 9) BASIC On Error Goto ErrorHandler code remains unconverted.
- 10) The use of # characters in place of double quotes for enclosing date values being assigned to a variable.
- 11) Removal of Visual Basic object type names during the conversion of object variable DIM statements. This limitation also applies to user defined data types.
- 12) IsEmpty(), Null and Error value implementations. There is too much chance of error if doing this type of conversion using a simple text replacement algorithm. But if an assignment statement is found to contain the text "Nothing" it is replaced with "empty" for compatibility with LiveCode.
- 13) ReDim statements are commented out, because they could either be used to clear an array or resize the array. The actual usage needs to be determined manually.
- 14) Visual Basic code using Collections.
- 15) Passing of named parameters to functions or handlers is not supported in LiveCode, and needs to be changed manually in the source code.
- 16) ELSEIF statements should be manually converted into SWITCH/CASE statements if there are more than 2 conditions being checked. This change will also make the code easier to read, understand and troubleshoot.
- 17) The DO WHILE/DO UNTIL statements are converted if both keywords are on the same line with each other. If the DO keyword is separated from the WHILE/UNTIL keyword then it won't be converted.
- 18) Very few recorded VBA Macro statements will be directly converted due to the reliance upon application-specific objects, properties and methods. Most of these features will only be available within the original Microsoft application.
- 19) Array references will be converted from parenthesis () to square brackets [] on the left hand side of the assignment operator. This change won't occur for array references on the right hand side.
- 20) WITH/END WITH keywords are not applicable in LiveCode, so they are commented out.
- 21) The RealBasic CountFields/CountFieldsB() function should be manually replaced with setting the itemDelimiter to the field delimiter and then getting the count of the number of items in the container.
- 22) The ZBASIC/FutureBasic cursor cursorID is converted into "set the cursor to" but it will be necessary to change the constants to the appropriate values used in LiveCode (watch, arrow etc).
- 23) The DELAY command is converted into "wait for". If you want to specify "with messages" then this text should be added manually along with a unit of time. (ZBASIC/FutureBasic).
- 24) PRINT and STDOUT keywords are converted into LiveCode "put" statements, which will by default send the output to the message box during development. This is probably not what you want with a modern application. Further work should be done to analyze how this info should be presented to the user within the context of an event driven development process.
- 25) PowerBasic Embedded x86 assembler in-line code which starts with a "!" character will be commented out, since this is not applicable to LiveCode development.

Step 1 - Create FmPro Migrator Project File

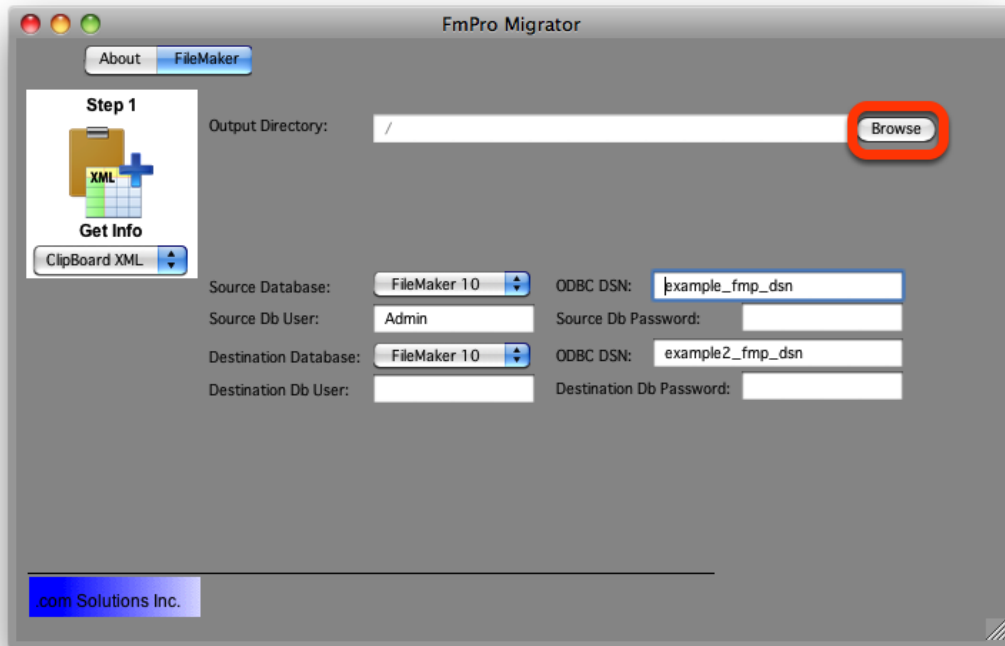
In order to perform a migration project, FmPro Migrator needs to create a MigrationProcess.db3 project file to store information about the migration project. Code conversion projects work a little differently than database conversion projects, so the Create Project File... menu is used to get the process started.

Open FmPro Migrator



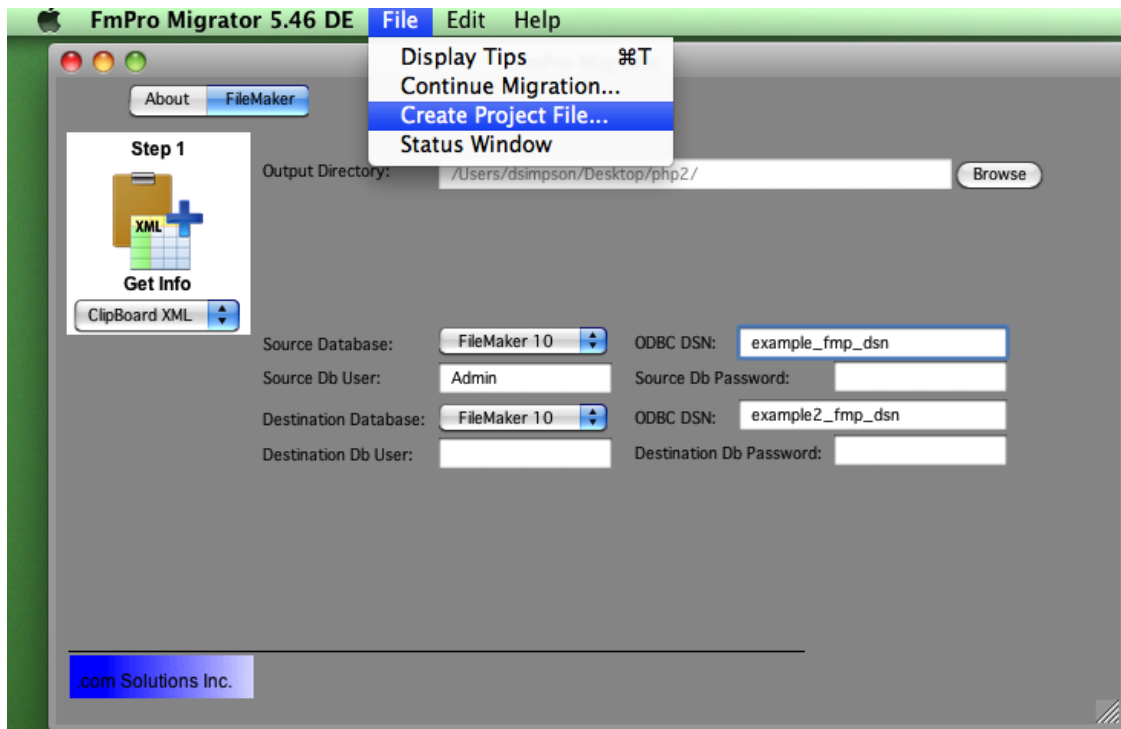
Click the FileMaker tab to select an output directory.

Click FileMaker Tab



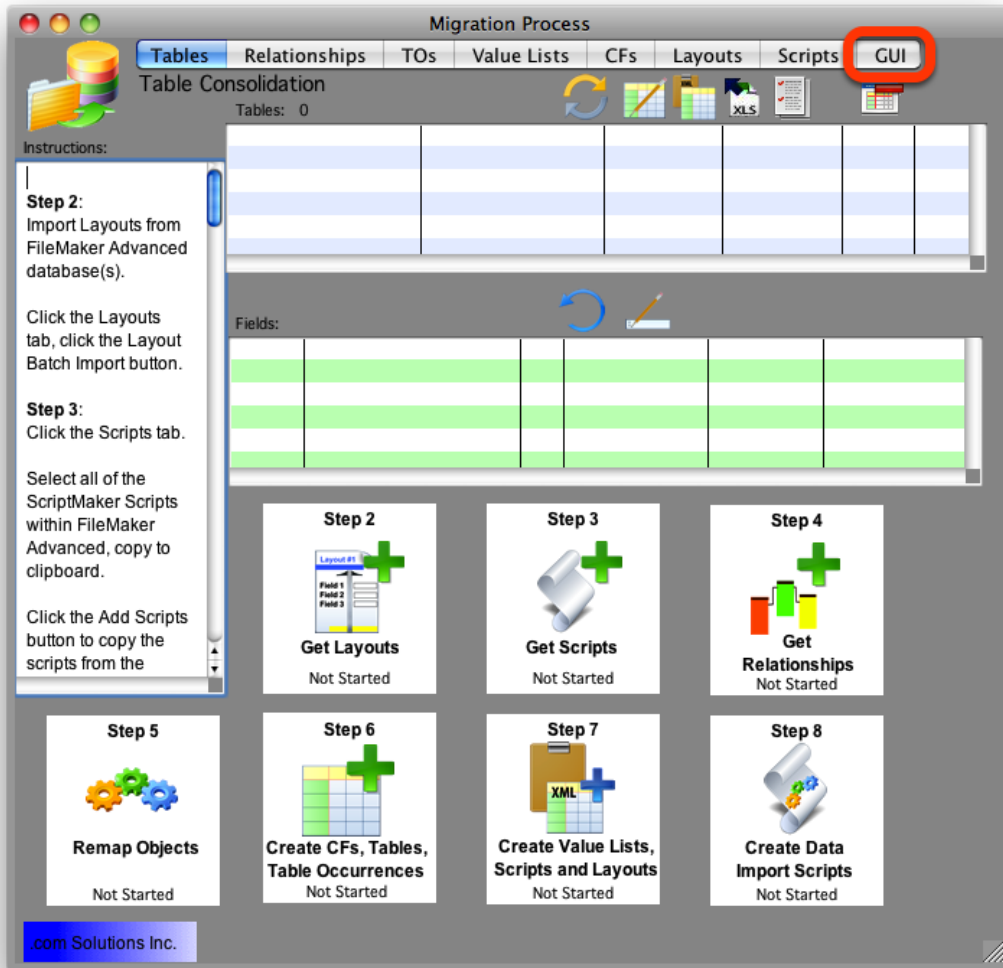
Click the Browse button to select the directory which will be used to store the FmPro Migrator project file. This directory can be the same output directory used for generating the converted scripts or stack file or it can be a different directory.

Select Create Project File... Menu



Select the Create Project File... item from the File menu. As soon as the FmPro Migrator project file has been created, the Migration Process window will open.

Click GUI Tab of Migration Process Window



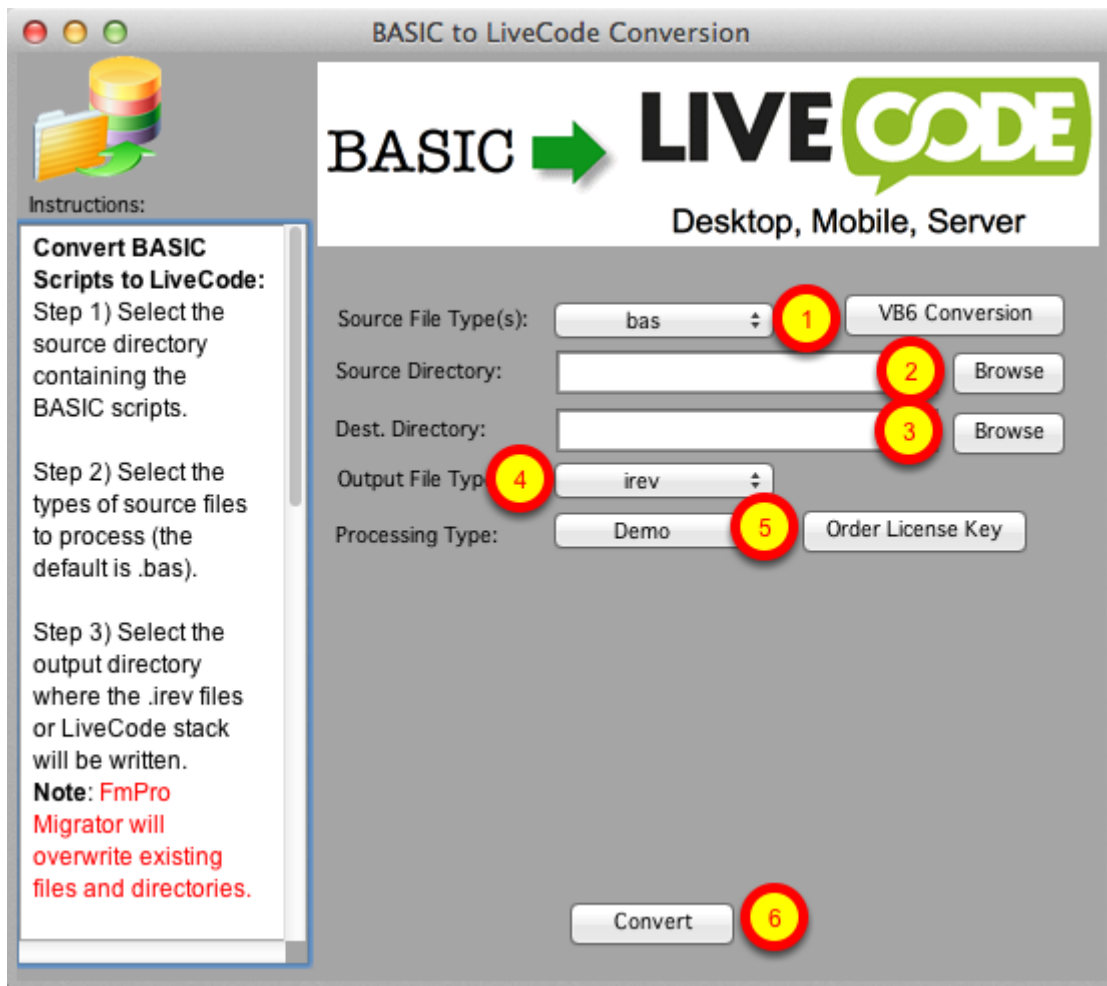
Since a database migration is not being done, ignore the contents of the various database migration features, and click on the GUI tab button.

Step 2 - Select Conversion Options & Convert BASIC Files

Click BASIC to LiveCode Button



(1) Select the BASIC to LiveCode option from the menu, then (2) Click the BASIC to LiveCode button to open the BASIC to LiveCode window.



There are several options which need to be set prior to performing a code conversion project:

1) Source File Type(s): bas, vba, vbs or txt.

2) Source Directory - This is the top-level directory containing your BASIC scripts. All enclosed directories will be traversed and files within those directories will also be processed.

3) Destination Directory - This is the output directory where the converted files will be written.

4) Output File Type - The BASIC files can be converted into .irev files or a single LiveCode stack having a card representing each converted source file.

5) By default, the BASIC to LiveCode process operates in Demo mode. In Demo mode, 5 files of unlimited length will be processed. Ordering a license key removes this limitation.

6) Click the Convert button to convert the BASIC files.

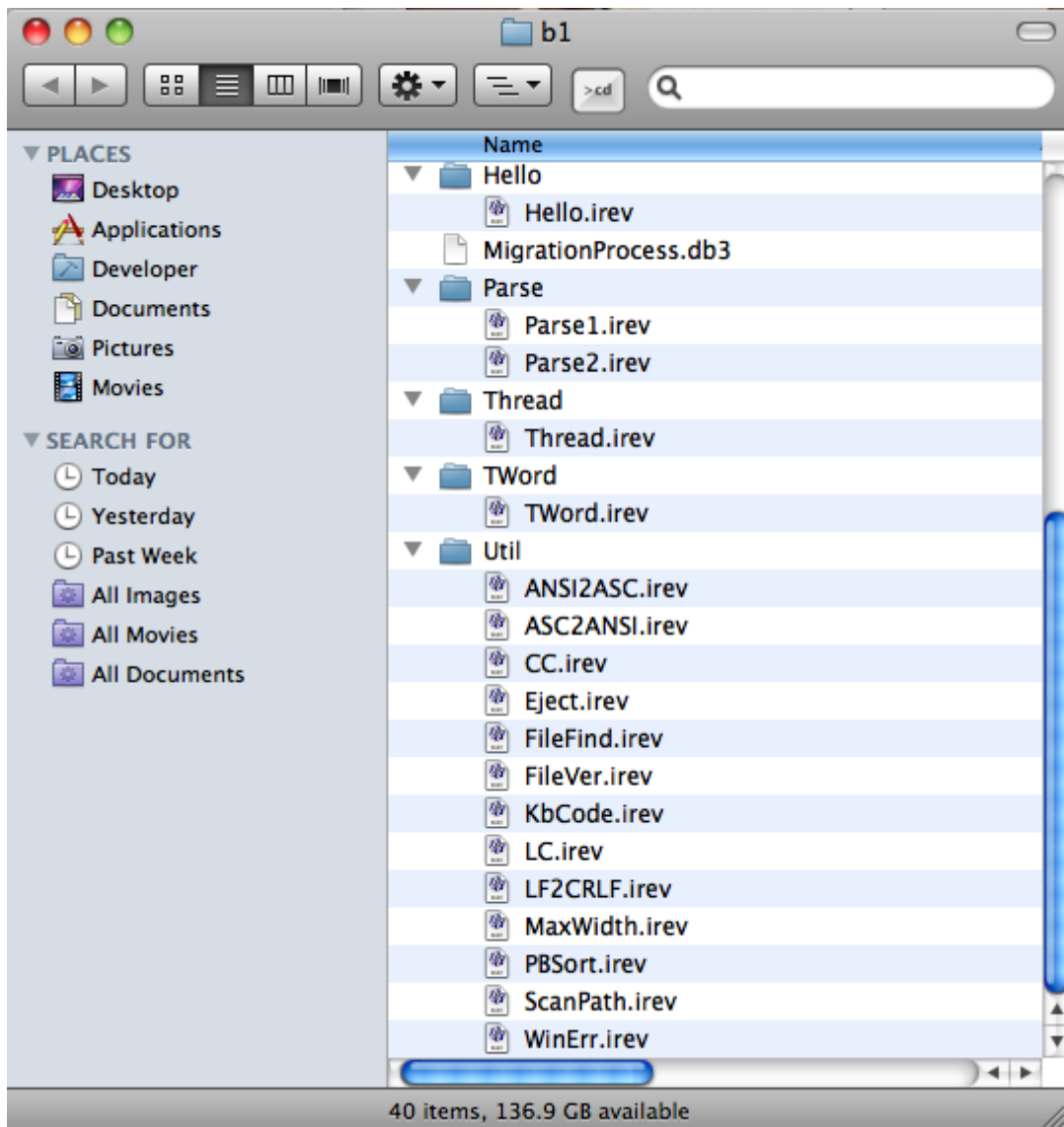
Note: The VB6 Conversion process is explained in a separate section of this manual.

Conversion Results



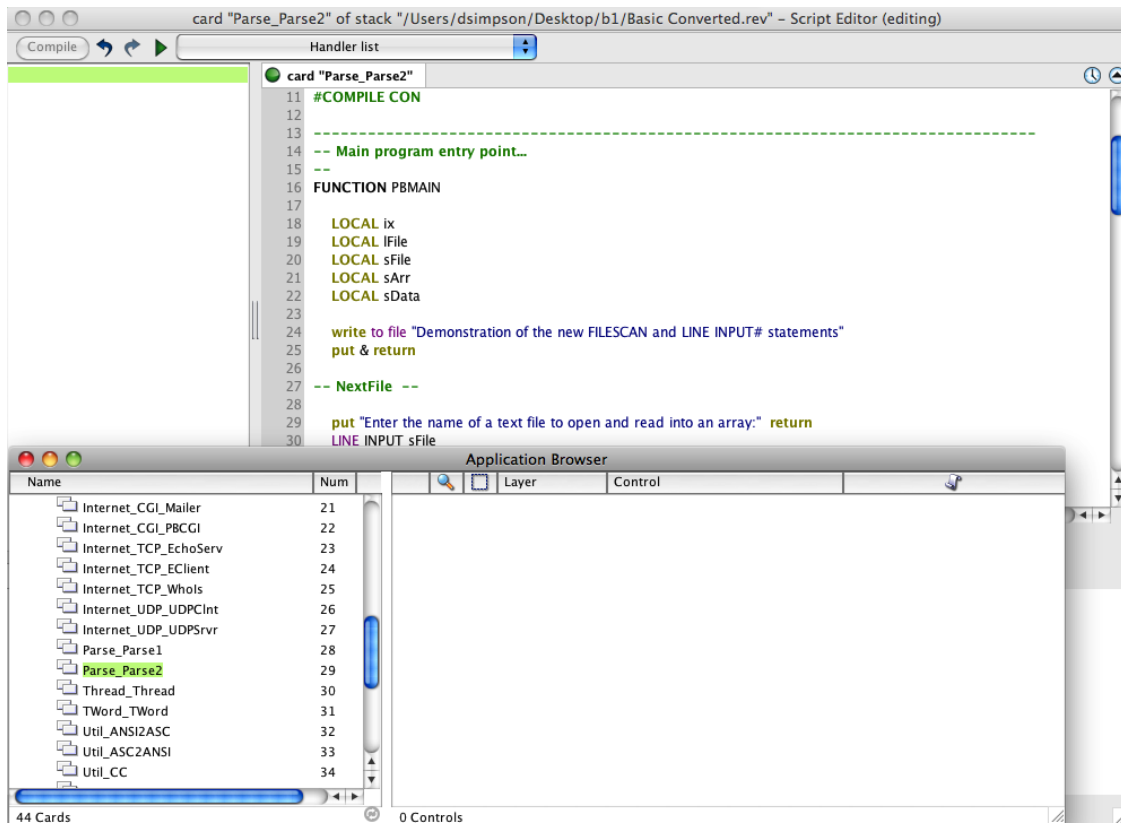
After clicking the Convert button, FmPro Migrator converts each of the files and displays the conversion results.

.irev Converted Files



The generated .irev files shown in the output directory. When generating .irev files, the <?rev tags are added to each file since each line of the file is considered executable code.

BASIC Converted.rev Stack with Converted Card Scripts



If the stack output file type is selected, a stack named BASIC Converted.rev will be created within the output directory. Each BASIC script is converted into a card having a name consisting of the subdirectory name and script name. Select any card in the Rev Application Browser and right-click on the card to select the Edit Script contextual menu item. The script will be opened in the LiveCode editor.

VB6 to LiveCode Conversion

VB6 to LiveCode - Form and Script Conversion

This chapter of the manual covers the process for converting VB6 projects into LiveCode stacks. Each .FRM form file in the Visual Basic 6 project is converted into a card within a LiveCode stack file, and each .bas code file is converted into a .irev text file containing LiveCode code.

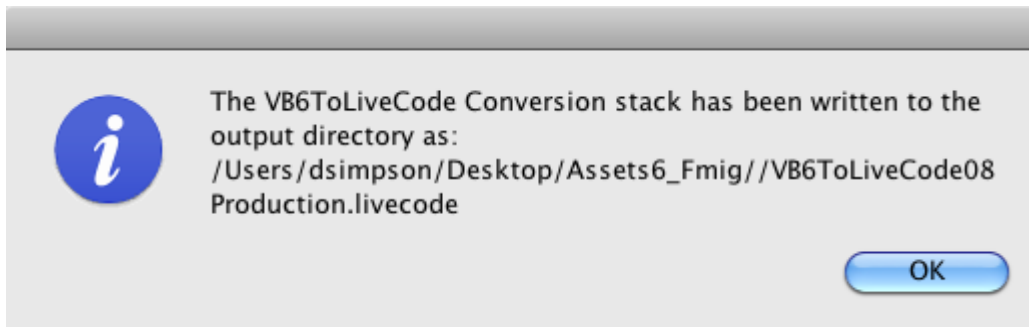
Open BASIC to LiveCode Conversion Window



Once the BASIC to LiveCode Conversion window has been opened, click on the VB6 Conversion button. Clicking this button will save either the Demo version (Demo mode), or Production version (Licensed mode) of the VB6ToLiveCode conversion stack file.

The Demo version of the VB6ToLiveCode Conversion stack converts up to 5 forms and 5 .bas script files. Once the license key has been entered for the BASIC to LiveCode Conversion feature, the production version of the stack will be saved to the output directory when clicking the VB6 Conversion button (as shown above). The VB6toLiveCode Production stack provides unlimited processing capabilities.

Saving the VB6ToLiveCode Conversion Stack



The VB6ToLiveCode Conversion stack file will be saved to the previously selected FmPro Migrator output directory.

Using The VB6ToLiveCode Conversion Stack



(1) Select a source directory containing the VB6 project files, (2) select an destination directory for the converted files, (3) click the Convert button.

VB6 Conversion Results

The screenshot displays the 'VB6 to LiveCode Conversion' application window. On the left, a sidebar contains instructions for converting VB6 forms and scripts to LiveCode. The main area features a header with the 'VB6' logo and 'LIVE CODE' logo, followed by the text 'Desktop, Mobile, Web, Server'. Below this, there are two text input fields for 'Source Directory' and 'Dest. Directory', both containing the path '/Users/dsimpson/fmpro_migratc', with 'Browse' buttons next to them. A 'Convert' button is located at the bottom center. A red-bordered box highlights the conversion results: '199 Forms Processed in 40.1 Sec.', '14 Script Files Processed in 0.6 Sec.', and '13,450 Lines of Code Processed.'.

Instructions:

Convert VB6 Forms and Scripts to LiveCode:

Step 1) Select the source directory containing the VB6 project.

Step 2) Select the output directory where the LiveCode stack will be created.

Note: Existing files and directories will be overwritten.

Step 3) Press the Convert button.

The VB6 to LiveCode stack will

Source Directory: /Users/dsimpson/fmpro_migratc Browse

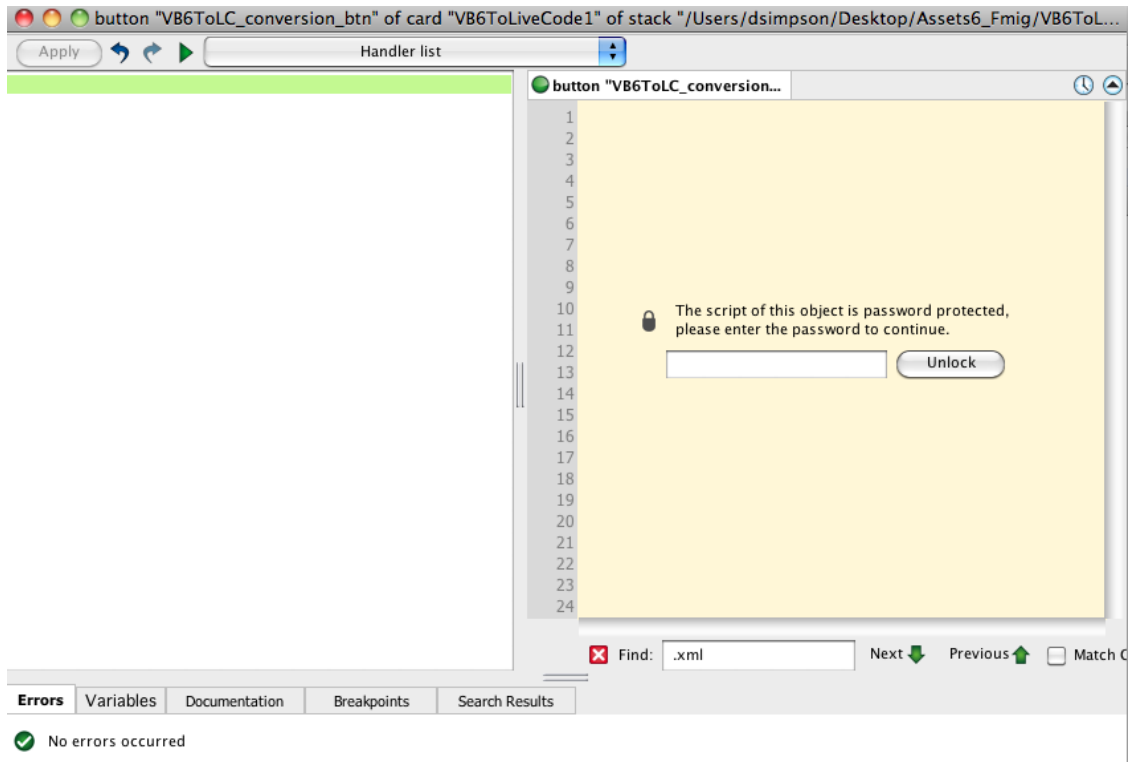
Dest. Directory: /Users/dsimpson/fmpro_migratc Browse

199 Forms Processed in 40.1 Sec.
14 Script Files Processed in 0.6 Sec.
13,450 Lines of Code Processed.

Convert

Once the processing has been completed, the results will be displayed, and the new VB6 Converted.livcode stack will be saved into the Destination Directory and left open in the LiveCode IDE in front of the VB6ToLiveCode Conversion stack.

Locked Stack Error



The VB6ToLiveCode Conversion stack is a locked stack, so that you won't be able to open any of its scripts within the LiveCode IDE.

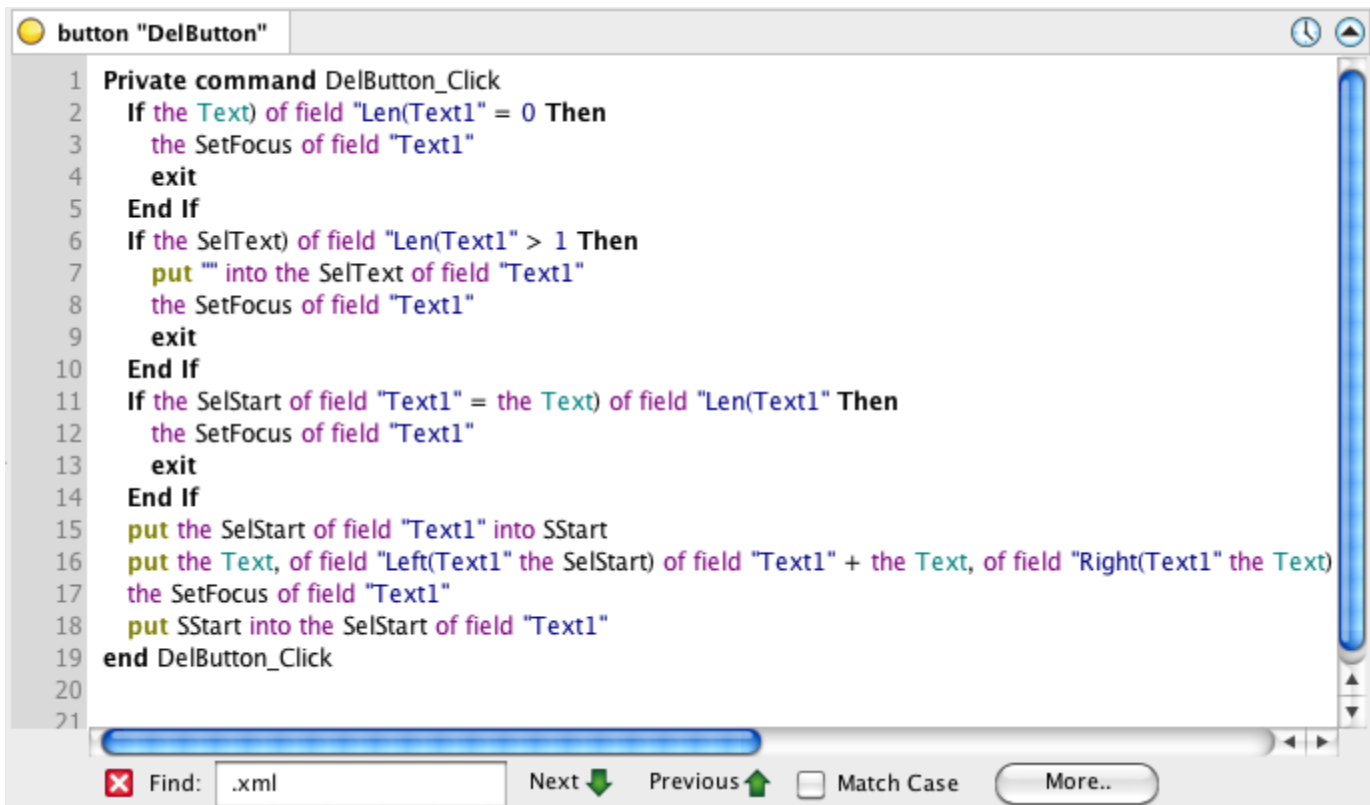
Unlike the rest of the FmPro Migrator application, the VB6ToLiveCode Conversion stack is implemented as a stack file which runs directly in the LiveCode IDE. The VB6 Conversion feature is implemented as a stack in order to allow it to insert converted scripts of unlimited length into objects on the converted cards.

DelButton - Original VB6 Code

```
Private Sub DelButton_Click()  
    If Len(Text1.Text) = 0 Then  
        Text1.SetFocus  
        Exit Sub  
    End If  
    If Len(Text1.SelText) > 1 Then  
        Text1.SelText = ""  
        Text1.SetFocus  
        Exit Sub  
    End If  
    If Text1.SelStart = Len(Text1.Text) Then  
        Text1.SetFocus  
        Exit Sub  
    End If  
    SStart = Text1.SelStart  
    Text1.Text = Left$(Text1.Text, Text1.SelStart) + Right$(Text1.Text, Len(Text1.Text) - Text1.SelStart - 1)  
    Text1.SetFocus  
    Text1.SelStart = SStart  
End Sub
```

This is the original VB6 code for the DelButton script.

DelButton LiveCode - Converted Script



```
1 Private command DelButton_Click  
2 If the Text) of field "Len(Text1)" = 0 Then  
3 the SetFocus of field "Text1"  
4 exit  
5 End If  
6 If the SelText) of field "Len(Text1)" > 1 Then  
7 put "" into the SelText of field "Text1"  
8 the SetFocus of field "Text1"  
9 exit  
10 End If  
11 If the SelStart of field "Text1" = the Text) of field "Len(Text1)" Then  
12 the SetFocus of field "Text1"  
13 exit  
14 End If  
15 put the SelStart of field "Text1" into SStart  
16 put the Text, of field "Left(Text1) the SelStart) of field "Text1" + the Text, of field "Right(Text1) the Text)"  
17 the SetFocus of field "Text1"  
18 put SStart into the SelStart of field "Text1"  
19 end DelButton_Click  
20  
21
```

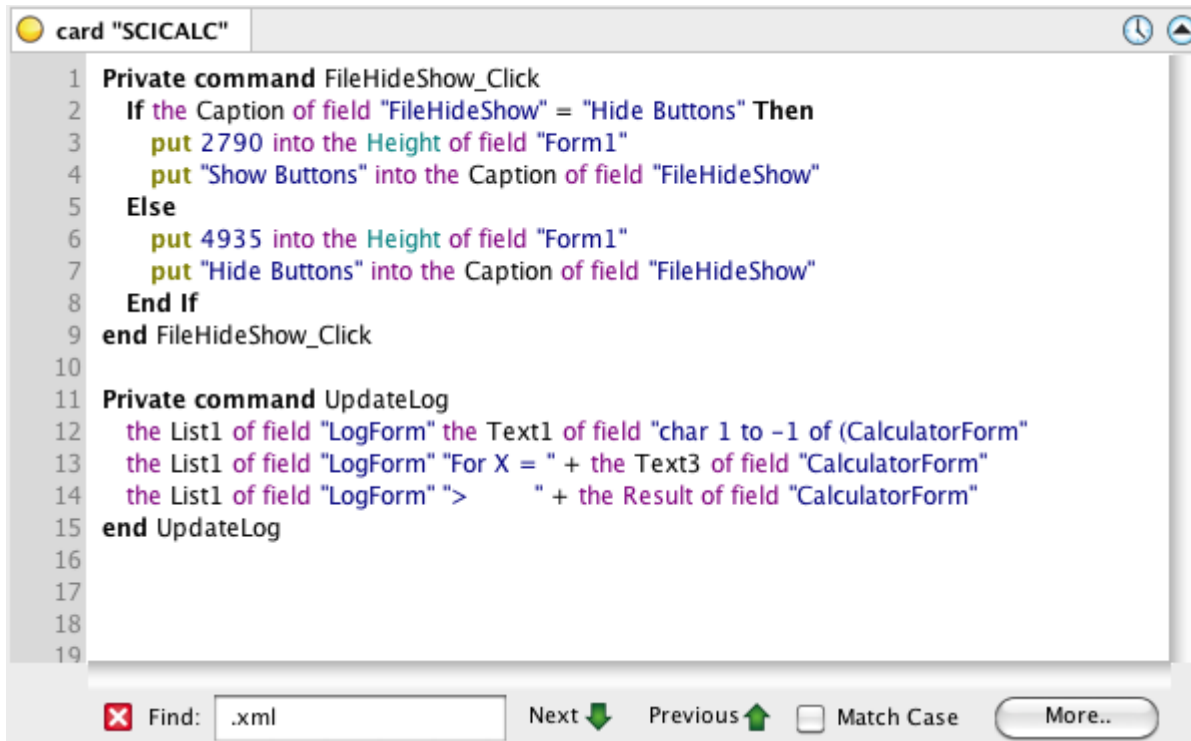
This screenshot shows the LiveCode version of the DelButton object script. This screenshot was taken after hitting the TAB key to reformat the code.

This code definitely needs work after the automated conversion process, but the basic structure of

the code syntax, functions and operators have been re-written for LiveCode. Most of the object properties will not make sense within the context of a LiveCode application.

The code has been moved from the .FRM file and placed within the DelButton object, as would be expected of a LiveCode application. But the handler has not been renamed using the on mouseUp() message to allow for multiple scripts to exist within the same object. As with LiveCode, there could be multiple messages and multiple scripts handling messages for any particular object.

SCICALC LiveCode Card Script

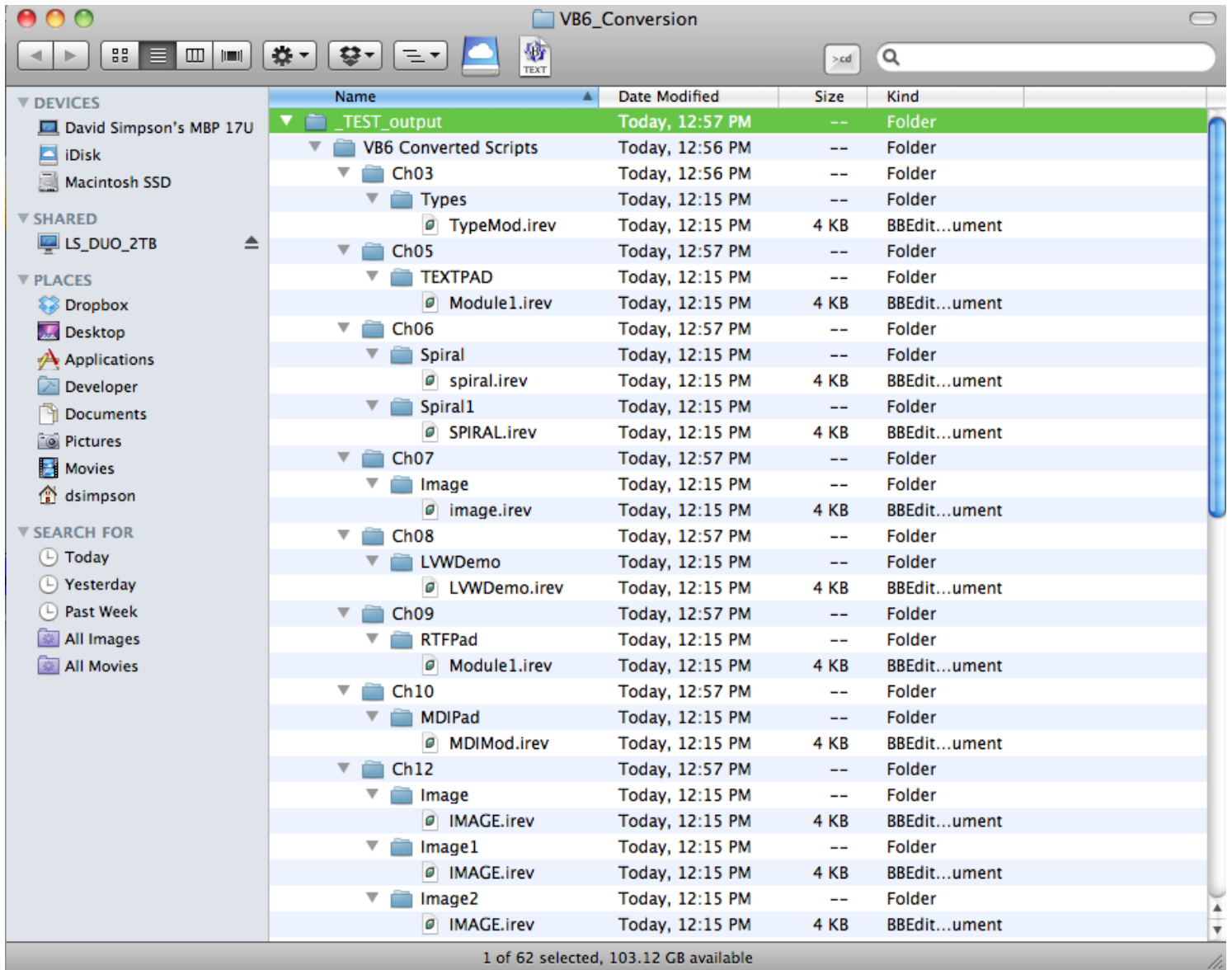


```
1 Private command FileHideShow_Click
2   If the Caption of field "FileHideShow" = "Hide Buttons" Then
3     put 2790 into the Height of field "Form1"
4     put "Show Buttons" into the Caption of field "FileHideShow"
5   Else
6     put 4935 into the Height of field "Form1"
7     put "Hide Buttons" into the Caption of field "FileHideShow"
8   End If
9 end FileHideShow_Click
10
11 Private command UpdateLog
12   the List1 of field "LogForm" the Text1 of field "char 1 to -1 of (CalculatorForm"
13   the List1 of field "LogForm" "For X = " + the Text3 of field "CalculatorForm"
14   the List1 of field "LogForm" ">      " + the Result of field "CalculatorForm"
15 end UpdateLog
16
17
18
19
```

Find: Next Previous Match Case

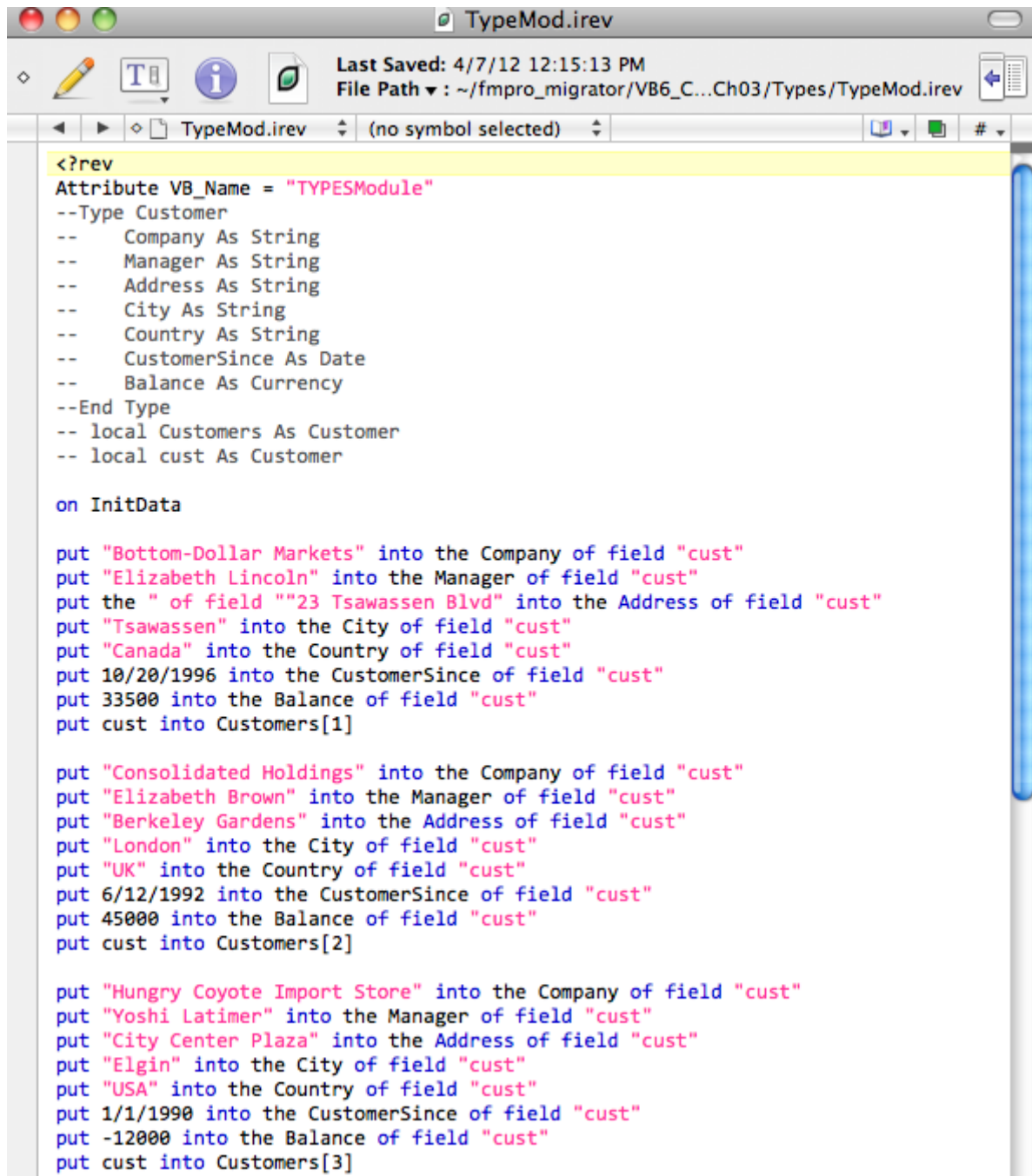
Any remaining VB6 code which is not associated with a form object, is placed into the script for the card itself, just as would be done with a typical LiveCode application.

VB6 Converted .irev Scripts



VB6 .bas files are converted into text files having the .irev extension and stored in a directory structure matching the original VB6 project.

TypeMod.irev LiveCode Script



```
<?rev
Attribute VB_Name = "TYPESModule"
--Type Customer
--  Company As String
--  Manager As String
--  Address As String
--  City As String
--  Country As String
--  CustomerSince As Date
--  Balance As Currency
--End Type
-- local Customers As Customer
-- local cust As Customer

on InitData

put "Bottom-Dollar Markets" into the Company of field "cust"
put "Elizabeth Lincoln" into the Manager of field "cust"
put the " of field ""23 Tsawassen Blvd" into the Address of field "cust"
put "Tsawassen" into the City of field "cust"
put "Canada" into the Country of field "cust"
put 10/20/1996 into the CustomerSince of field "cust"
put 33500 into the Balance of field "cust"
put cust into Customers[1]

put "Consolidated Holdings" into the Company of field "cust"
put "Elizabeth Brown" into the Manager of field "cust"
put "Berkeley Gardens" into the Address of field "cust"
put "London" into the City of field "cust"
put "UK" into the Country of field "cust"
put 6/12/1992 into the CustomerSince of field "cust"
put 45000 into the Balance of field "cust"
put cust into Customers[2]

put "Hungry Coyote Import Store" into the Company of field "cust"
put "Yoshi Latimer" into the Manager of field "cust"
put "City Center Plaza" into the Address of field "cust"
put "Elgin" into the City of field "cust"
put "USA" into the Country of field "cust"
put 1/1/1990 into the CustomerSince of field "cust"
put -12000 into the Balance of field "cust"
put cust into Customers[3]
```

VB6 Supported Form Objects

The following object types are converted from VB6 forms:

- PictureBox
- ImageBox
- Label
- TextBox

Frame
CommandButton
CheckBox
OptionButton - (grouped if within a Frame)
ComboBox
ListBox
HorizontalScrollbar
VerticalScrollbar
DriveListBox
DirectoryListBox
FileListBox
Slider
TabDlg

VB6 Unsupported Form Objects

Timer
Shape
Line
Data
OLE
ActiveX Controls

Additional Conversion Notes

- 1) Note: Objects having their coordinates set as negative numbers will be reset to a coordinate value of 0.
- 2) All VB6 forms must use the default measurement of Twips, as these are converted into Pixel coordinates prior to conversion. Any forms which use any other unit of measurement should be updated in Visual Basic 6 to use Twips prior to conversion.