.com Solutions Inc.

FmPro Migrator - FileMaker Pro to PHP Conversion Procedure

FmPro Migrator - FileMaker Pro to PHP Conversion Procedure

6 PHP Conversion - Manual Tasks

Step 1 - Import Database Info - FileMaker Pro

Licensing FmPro Migrator

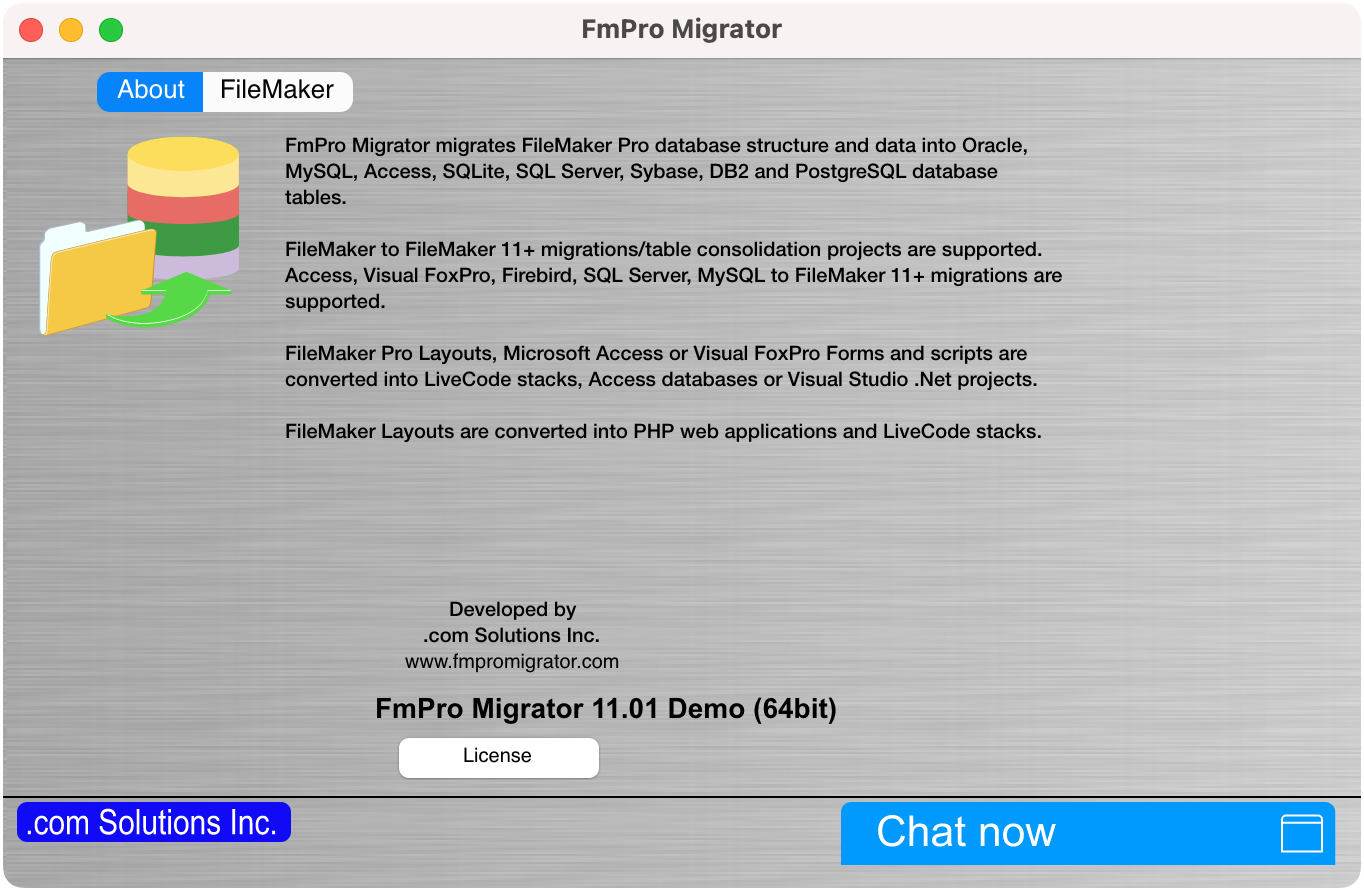
As of FmPro Migrator v11, all FmPro Migrator downloads are fully functional with features unlocked via a single license key.
This section of the manual shows how to enter the license key to unlock the features within FmPro Migrator.

FmPro Migrator 11.01
4/14/2024

Demo Edition Dialog

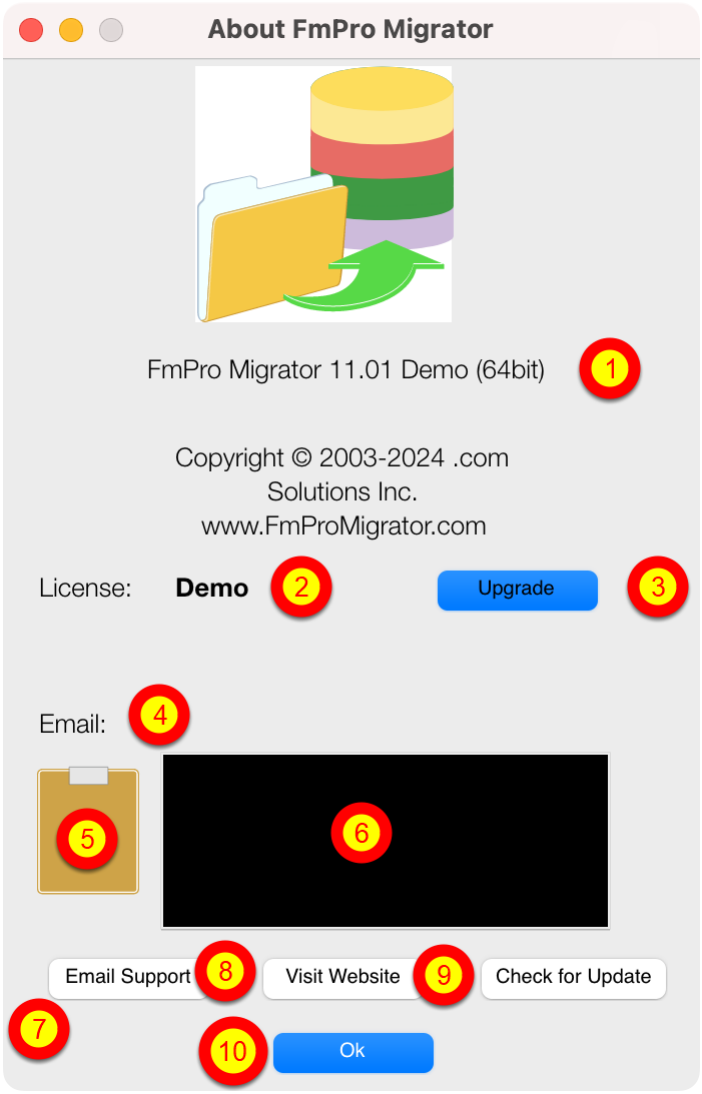
When launched the first time, FmPro Migrator will be running in Demo mode as shown in this screenshot. Clicking the Ok button opens the order page of the website.
In addition to transferring data for 5 database fields, the conversion features shown on the GUI tab of the Migration Process window will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

Demo Mode - About Tab



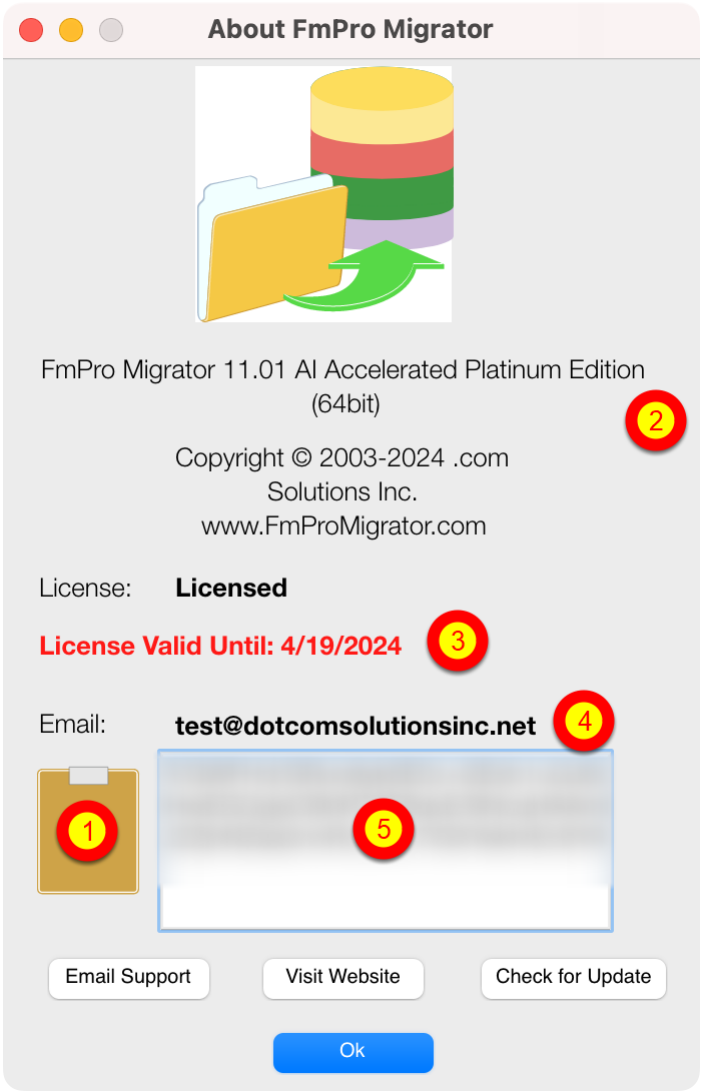
Clicking the "License" button on the About tab, or selecting the Help -> License/About menu items will open the About window where you can enter the license key.

About/License Window



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



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 (as well as other databases) into PHP web applications.

Document Version 13
Updated for FmPro Migrator Platinum Edition 11.01.
Added new Licensing FmPro Migrator info, updated screenshots.
4/14/2024

FileMaker 12+ Notes

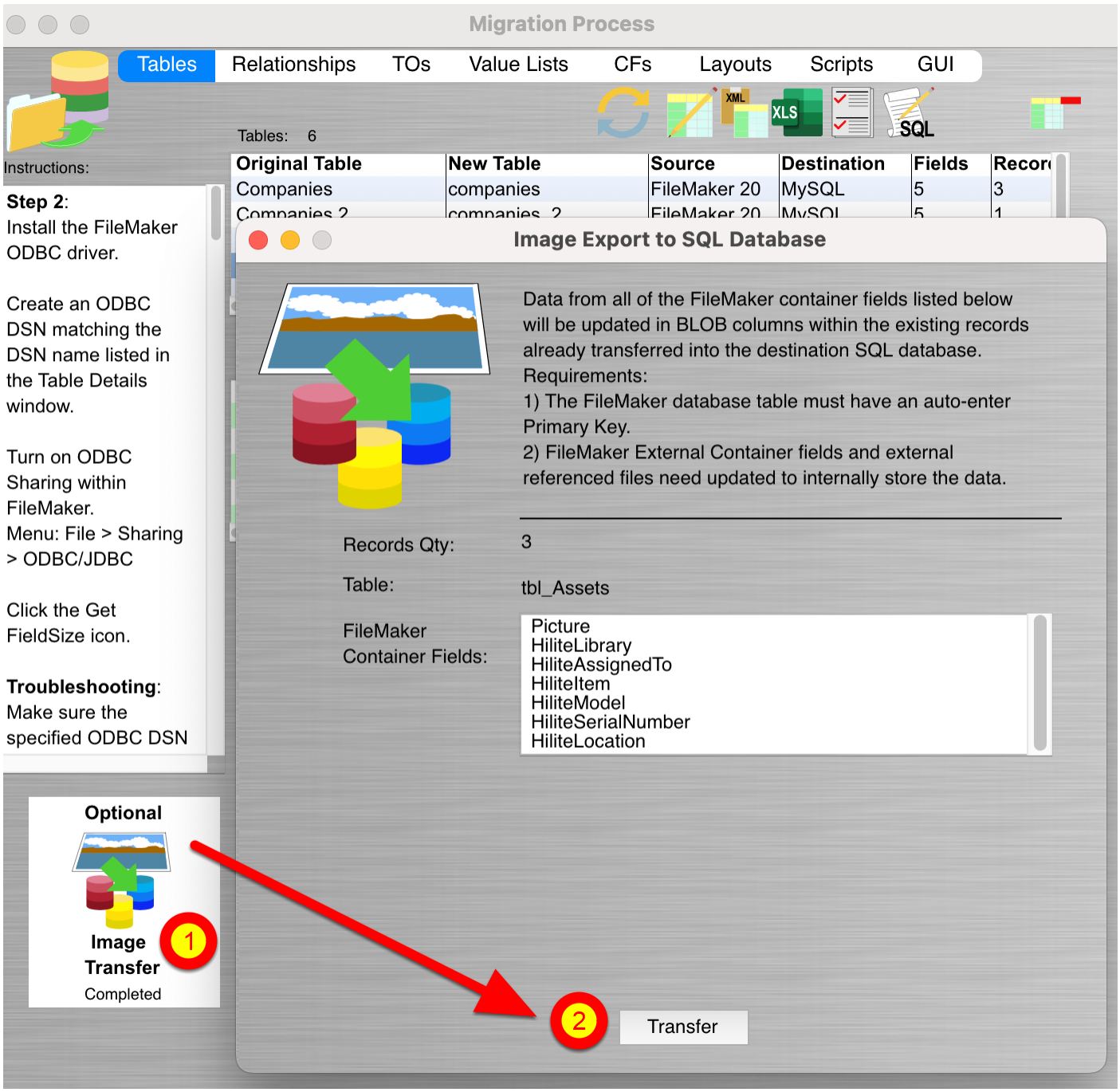
FileMaker 12+ versions represent a major improvement in the design surface, enhanced functionality, performance and a change in the binary file format. This change includes major changes to the FileMaker layout XML format used by FmPro Migrator to generate PHP web forms. Though all themes are supported if applied to individual objects, not all new features (rounded corners of fields, object blending features) are supported at this time.

FileMaker 11 Notes - Unicode Characters

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 or FileMaker Pro 12+ version 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.

This issue is resolved with the Image Export to SQL Database feature within FmPro Migrator, as shown on the Tables tab [as shown in this screenshot]. Since this problem is solved

Pre-Migration Tasks - PHP Conversion

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

1) The CakePHP framework requires that each database table 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 PHP web application will display errors about missing columns unless these columns are restored as virtual fields within the model files.

FmPro Migrator also checks for TOs matching reserved words in the CakePHP framework. Therefore, it is recommended that developers should import the tables, relationships, value lists and only 1 layout into FmPro Migrator. Then perform a test conversion so that FmPro Migrator can perform the pre-flight test on the project. Manually make changes to the FileMaker database, reimport into FmPro Migrator and test with 1 layout again. Perform this task interactively until the preflight test passes. Then import the rest of the layouts and scripts into FmPro Migrator in order to generate the entire PHP web application.
Example: Table name Global should probably be changed to something like: GlobalTable in order to avoid conflicts.

4) Avoid table names which will require FmPro Migrator to rename the table. If the table name represents a reserved word in the SQL database being used, it will be renamed with a trailing underscore character. But the underscore character is a special character to CakePHP signaling that the following character is to be uppercased as a model name. This can be confusing to CakePHP, to it is best to avoid these types of names, especially if they are reserved words. Tables/TOs containing spaces will automatically be replaced with underscore characters, so this is not generally a problem. But a trailing underscore in the name is to be avoided.

5) If the Basic Authentication Type is selected on the PHP Conversion preferences window, FmPro Migrator will add two tables (users, tokens) to the MigrationProcess.db3 project file. These tables will show up in the list of tables in the Migration Process window after clicking the Convert button on the PHP Conversion window. If a Users table already exists in the list of tables, this

process will be skipped, and errors will occur when attempting to create or use login accounts. For best results, if a Users table already exists in the original database solution, rename the table in FileMaker before performing the conversion.

6) FileMaker databases which have Auto-Enter Creation Date or Modification Date properties assigned to fields can be easily converted to have the same functionality within the generated PHP web application. Database fields named: **created** - will become Auto-Magic CakePHP fields which will be populated with the creation timestamp when new records are added. Database fields named: **modified** or **updated**, will be updated with a modification timestamp whenever the record is modified. Therefore it is recommended that fields having these auto-enter properties should simply be renamed in FileMaker prior to performing the automated conversion. Making the modification within FileMaker will insure that the changes are correctly reflected within all layout objects and scripts.

7) FileMaker databases containing repeating fields should be redesigned to eliminate the use of repeating fields prior to conversion into the PHP web 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 PHP conversion process is done, the portals will be converted into data grid objects in the PHP web application.

8) The Save Records as PDF script step attached to a button will be directly converted into PHP code. The generated PHP code will use the html2ps package. Having a folder named html2ps located in the same folder selected as your ExtJS source directory, will enable FmPro Migrator to automatically copy this folder into the CakePHP /app/vendors folder during the initial project creation. The check for this source directory only occurs once, when the CakePHP application is first created in your htdocs directory.
Amodified version of this html2ps directory suitable for use with CakePHP has been uploaded to the PHP Conversion web page.

9) WebViewer & SuperContainer URLs. FmPro Migrator converts WebViewer objects into iFrames on the converted PHP web application forms. It is recommended that the prefix of the URL should be defined within a Global Field for these URLs in order to facilitate the possibility of having to change the SuperContainer server in the future. The Global field will be defined with an empty value by default within the beforeFilter() function of the app_controller.php file. Changing this value within the app_controller will enable calculated URLs to work correctly when the application is running. Note: If URLs are being stored within the FileMaker database using Stored Calculation fields, the URL will need to be updated within your database server if the SuperContainer server is moved. Using Unstored Calc fields to define these URLs will not require this change to the database table data, as the URL will be calculated automatically as the application is running.

Manual Layout Changes

1) Layout text labels. With FmPro Migrator Platinum Edition 6.79, it is no longer necessary to manually adjust text labels wider in most cases. ACSS "whitespace: pre" property is automatically added to each text label to prevent word wrapping. With webkit-based web browsers, large text blocks will word wrap properly within the bounding rectangle of the text object - just like within FileMaker. However IE 9 will display these large blocks of text as one long horizontal line. For these situations, the whitespace CSS property can be removed from the text label, thus allowing the text to word wrap within the bounds of the text object rectangle on both webkit and IE 9 web browsers.

2) Image Sizes. Image object bounding rectangles may need to be adjusted to fit the actual display size of the image on the layout. FmPro Migrator uses the object's bounding rectangle as the rectangle which will be used to create the object on the web form.

3) Fields displayed as Checkbox and Radio button groups work best when they are laid out either horizontally or vertically on the layout. Fields taller than 20 pixels are laid out horizontally on the web form, otherwise they are laid out vertically. Radio Button/Checkbox groups displayed as columns of objects is supported. These columns of buttons will look best when using a monospaced font. Customization of the number of columns to display is done within the form controller.

4) Grouped objects are are converted best if they are ungrouped prior to conversion. Some grouped objects may be skipped or colors may be wrong if they remain grouped during the conversion.

5) Grouped Text/Image button objects. **Note:** FileMaker 12 grouped objects require no changes. FileMaker developers often create grouped image object with a text label placed over top of the image. Even if the two objects are ungrouped, the text label being located over top of the image can prevent the mouse click from reaching the object below. Acouple of workarounds include: Set the text object to have the same script step link as the underlying image. This way the user can click the text label to activate the linked script step, or if they click around the edge of the text label and touch the underlying image object the script step will still be triggered. Another work-around is to build the image object with the embedded text within a graphics program, and paste the object onto the layout.

6) Field Vertical Size. Field contents will display better if they are at least 23 pixels tall, in order to display character descenders.

7) FileMaker Radio Button and Checkbox Group Columns. FileMaker displays Radio Buttons and Checkboxes in multiple columns on the layout if the field object is tall and wide enough. FmPro Migrator attempts to simulate this same visual behavior for Radio Button and Checkbox Groups taller than 20 pixels and wider than 94 pixels. Depending upon the average widths of the value list

items, it may be necessary to change the number of columns which have been estimated by FmPro Migrator. This change can be made in the calls to the prepareInputGroup() function located within the form controller file for any form. Unlike FileMaker, the item padding is calculated in characters instead of actual text width. Therefore the columns only will line up perfectly when using a monospaced font.
An alternate prepareInputGroup2() function is available within the application_controller.php file for use with non-monospaced fonts. This enhanced version of the code requries creating a fonts directory within the webroot directory and adding TrueType font .ttf files. The font name is specified within the prepareInputGroup2() function.

A Note About Other Databases

This migration process is specifically optimized for the conversion of FileMaker Pro databases into PHP web applications. 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. It is also not practical to extract info from ActiveX controls for the building of charts. So if charts are an important feature of the source database application, charts should be added within the FileMaker database. Then, when all changes have been made to the FileMaker Pro database, import the FileMaker Pro database into FmPro Migrator for conversion into the PHP web application. If these types of changes are not made, it is likely that some converted forms will generate errors due to a missing table name within the Model and Controller PHP files.

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. 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, and also create relationships in 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.

Step 1 - Import Database Info - Microsoft Access

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.

Step 1 - Import Database Info - Visual FoxPro

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.

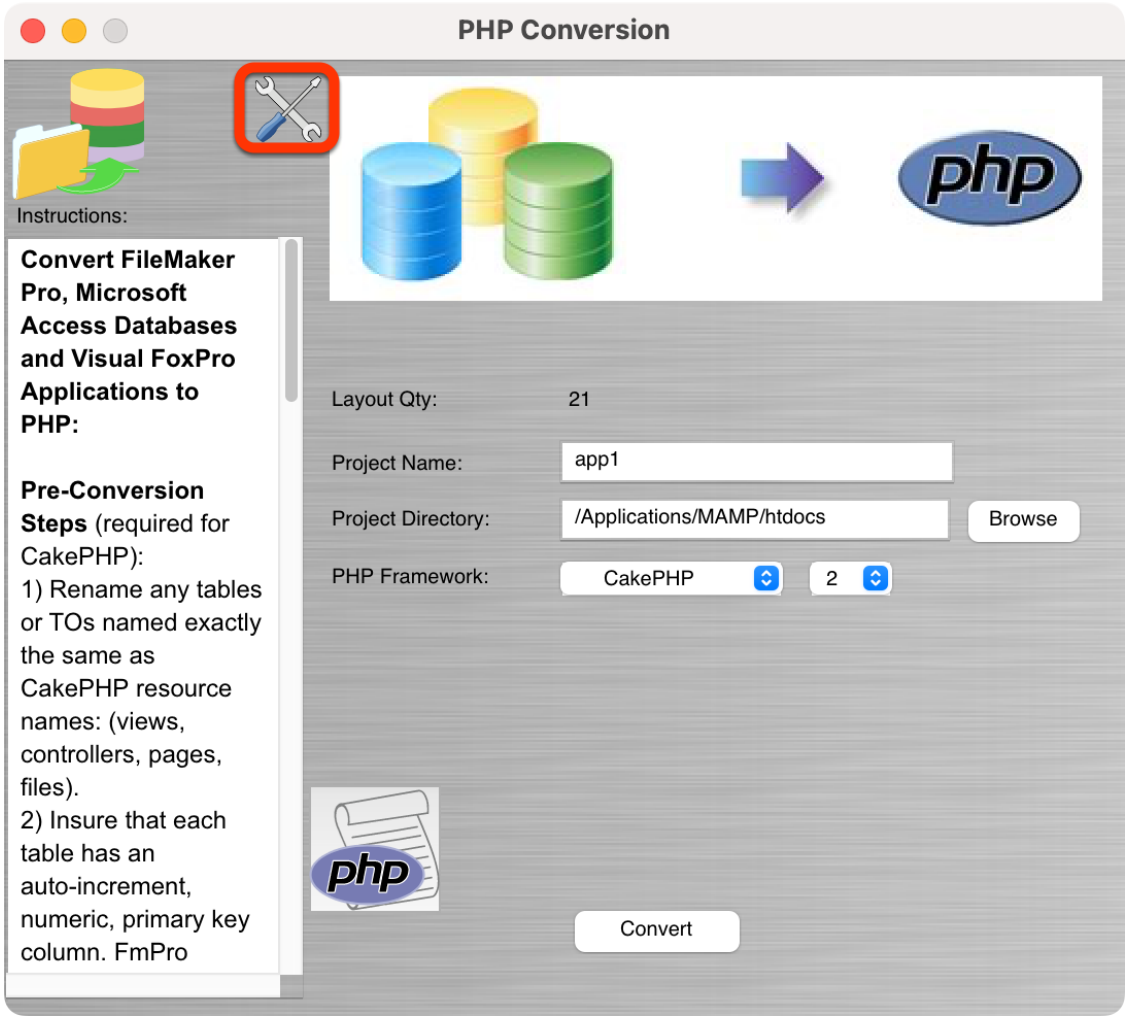
2) Download and following the instructions in the How to Import Visual FoxPro Projects into FmPro Migrator PDF manual from the FmPro Migrator support web page. Select Help from the Help menu in FmPro Migrator and your web browser will open this web page.

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.

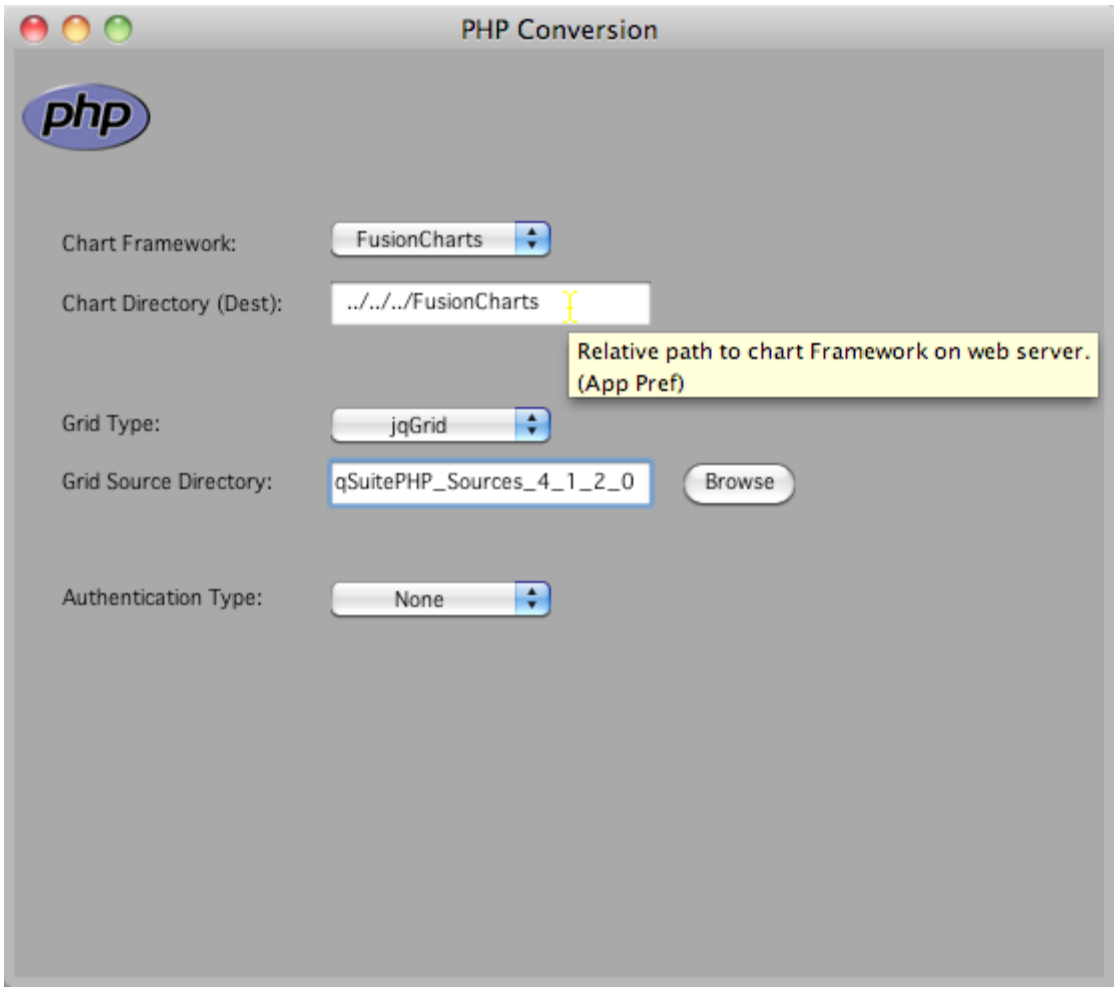
PHP Conversion Processing

Configuring Preferences



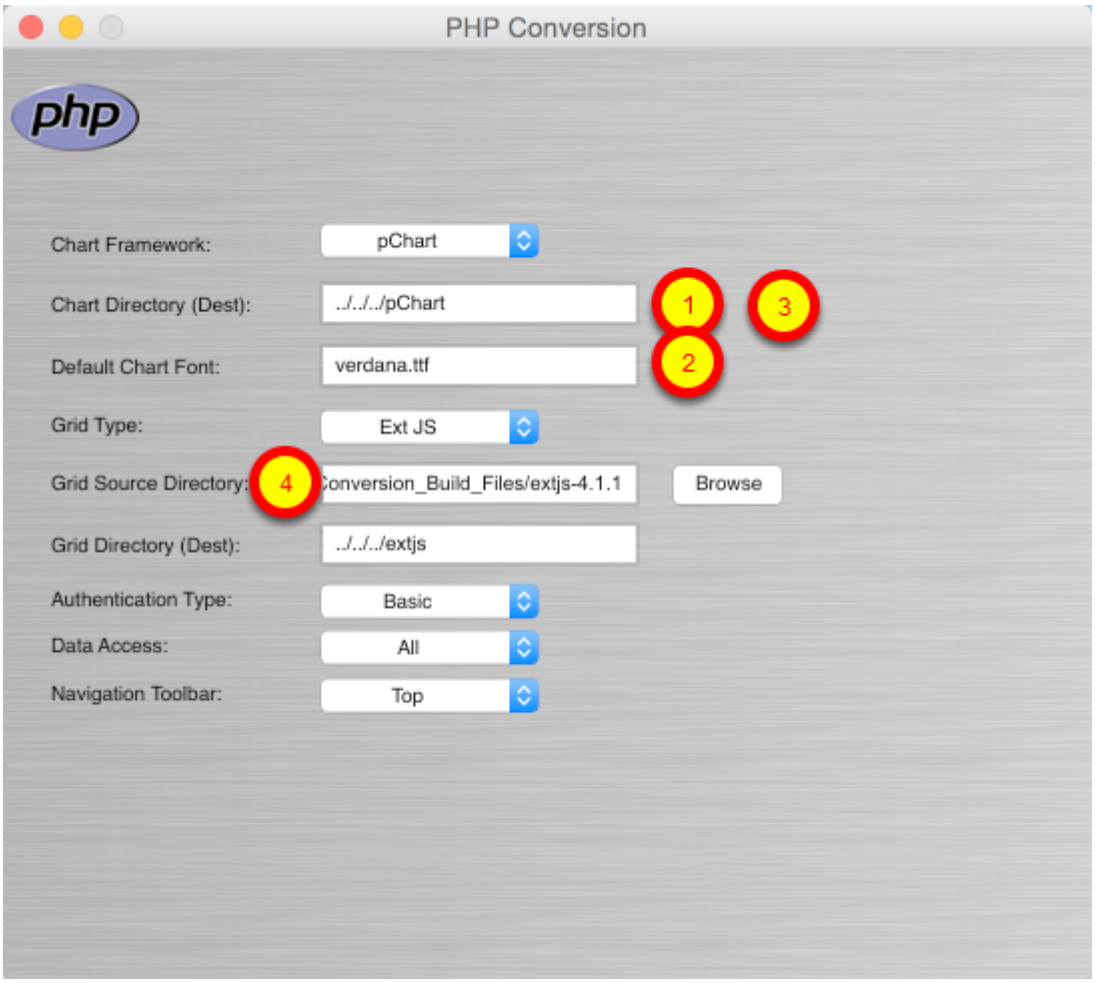
Before generating PHP scripts, it is important to configure preferences which will be used to generate the new PHP web application.
Click the Preferences icon in the upper left corner of the PHP Conversion window.

Prefs Storage Locations



Some preference items are stored globally in the FmPro Migrator application Prefs file, and are used for all PHP Conversion projects. Other preference items are stored in the MigrationPreferences.dat file within the output directory. The storage location for each prefs item is noted in the tooltip for each item, as (Project Pref) or (App Pref).

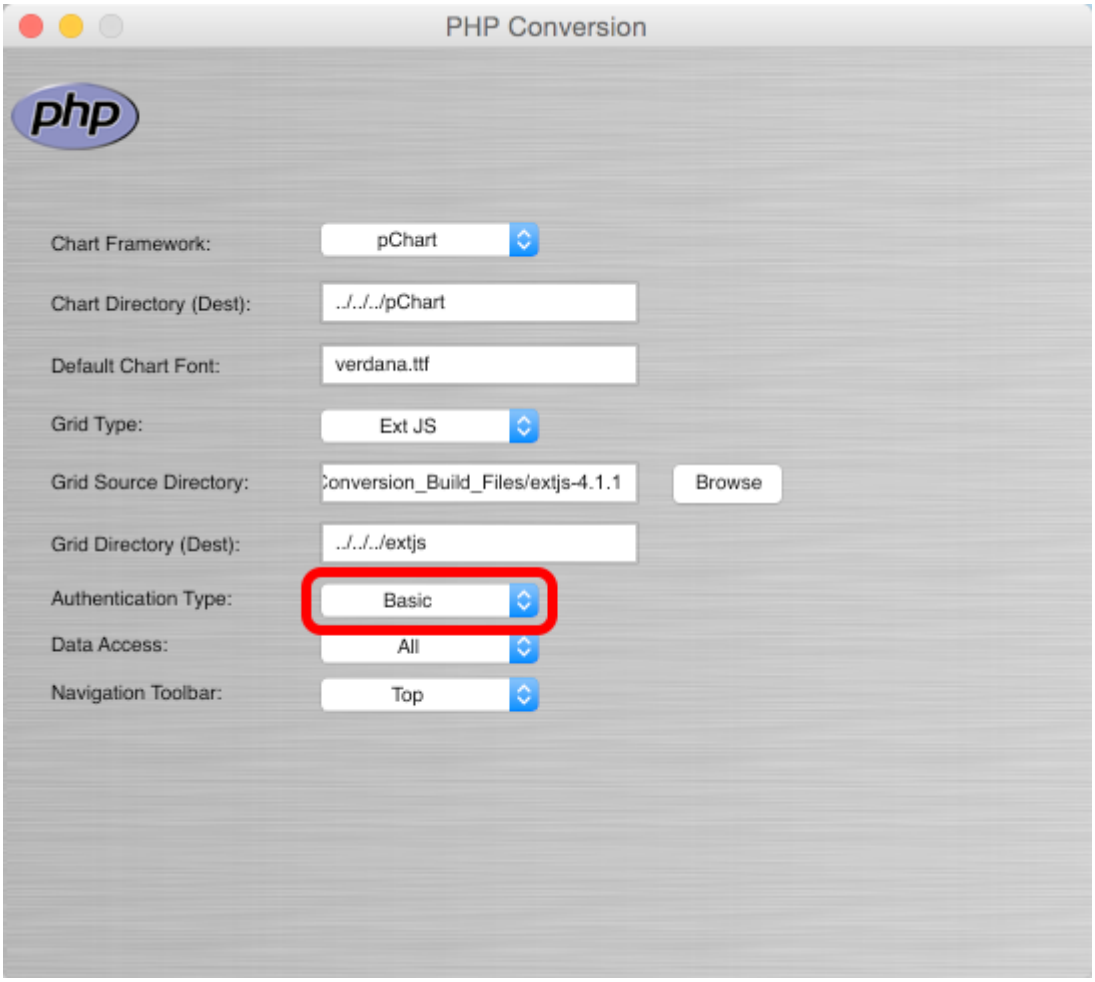
Application Preference Items



The application preference items include:
1) pChart destination directory relative path. This is the relative pathname used within the generated scripts to utilize the pChart library scripts. If the pChart directory is copied to the top-level of the web server htdocs/public_html directory, then the relative path from within the generated PHP project will be: ../../../pChart. This value will need to be changed if either the generated project is stored at some other directory level or if the pChart directory is moved or renamed.
2) The default font used in pChart scripts. The Verdana.ttf font is one of the TrueType fonts included within the pChart library. There are many sources of TrueType fonts, including MacOS X and Windows operating system directories, as well as internet websites. If this field is left blank, the font names embedded within the original FileMaker chart will be used within the pChart code. If a referenced font is not available within the pChart fonts directory, the text won't be displayed within the chart.
3) FusionCharts destination directory relative path. This is the relative pathname used within the generated PHP scripts to utilize the Fusion Charts library scripts.
4) The source directory on the local hard disk for the jqGrid library. If this library is not found,

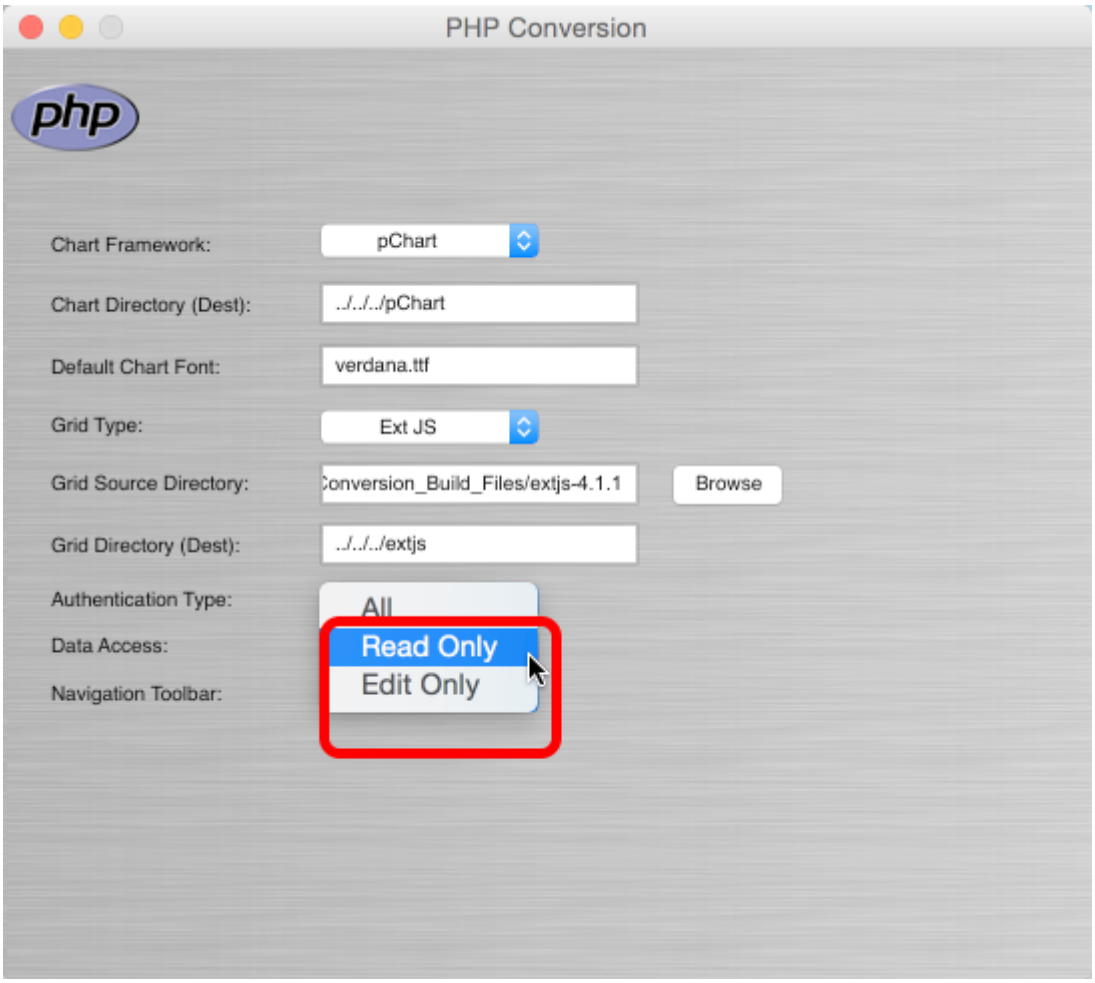
FmPro Migrator will skip copying these files into the application directory and Grid objects on forms won't function. **Note:** it is always recommended to download even a 30 day trial version of the jqGrid library from www.trirand.net, in order to create a complete project.

Building Basic Authentication Tables



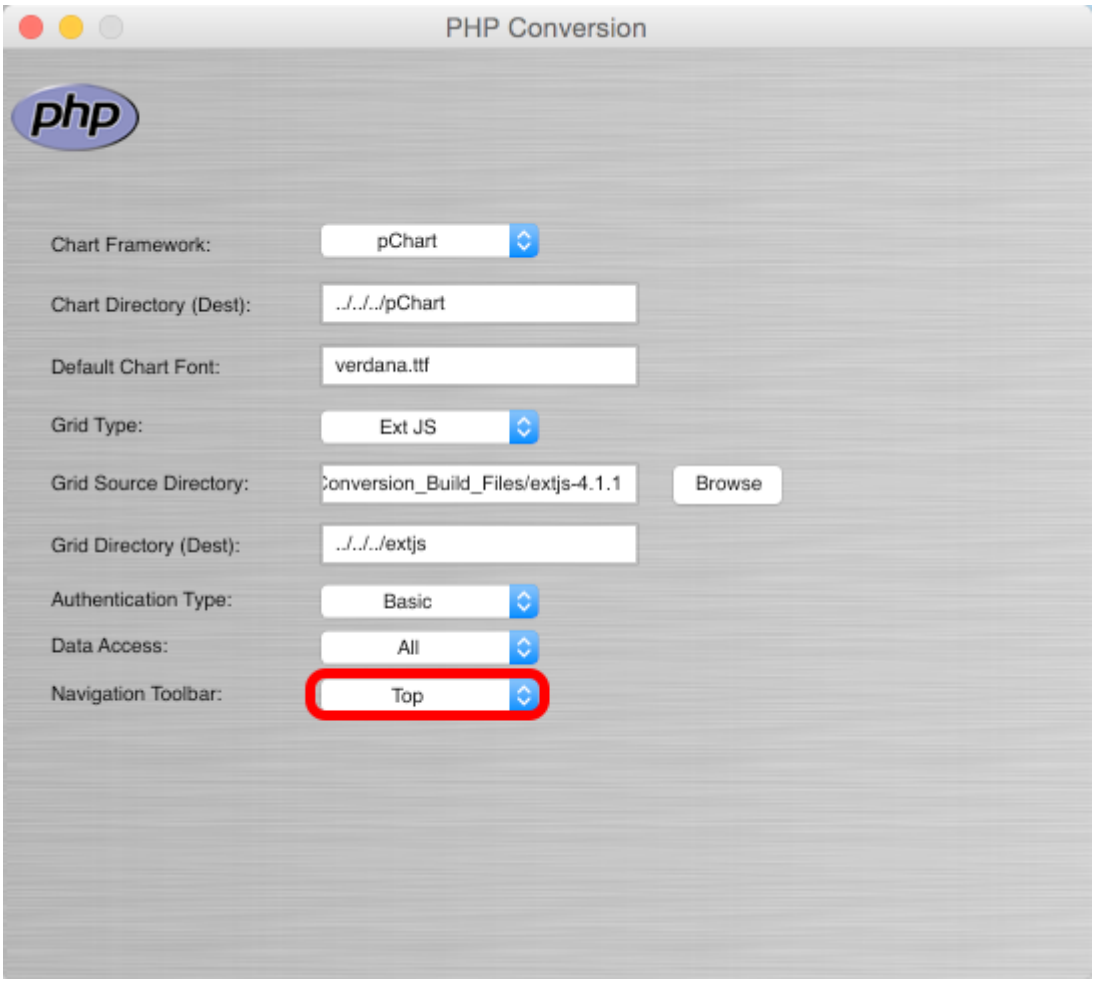
If the Basic Authentication Type has been selected, two tables (Users, Tokens) will automatically be added to the MigrationProcess.db3 project file the first time the Convert button is clicked. These tables will be listed in the list of tables on the Tables tab of the Migration Process window. Before running the generated application, click on the Create Table button for each table to create each table in the destination database.

Data Access Options



By default, the Data Access option is set to "All", which provides full access to the data in the database.
Selecting "Read Only" Data Access will disable the Add/Edit/Delete buttons on the navigation toolbar, and will disable scripted buttons having these features on the converted forms.
Selecting "Edit Only" enables editing of records, but prevents Add/Delete actions.
For disallowed Data Access features, the controller code is also removed from the form controller php file, preventing a user from accessing the action by manually changing the URL.

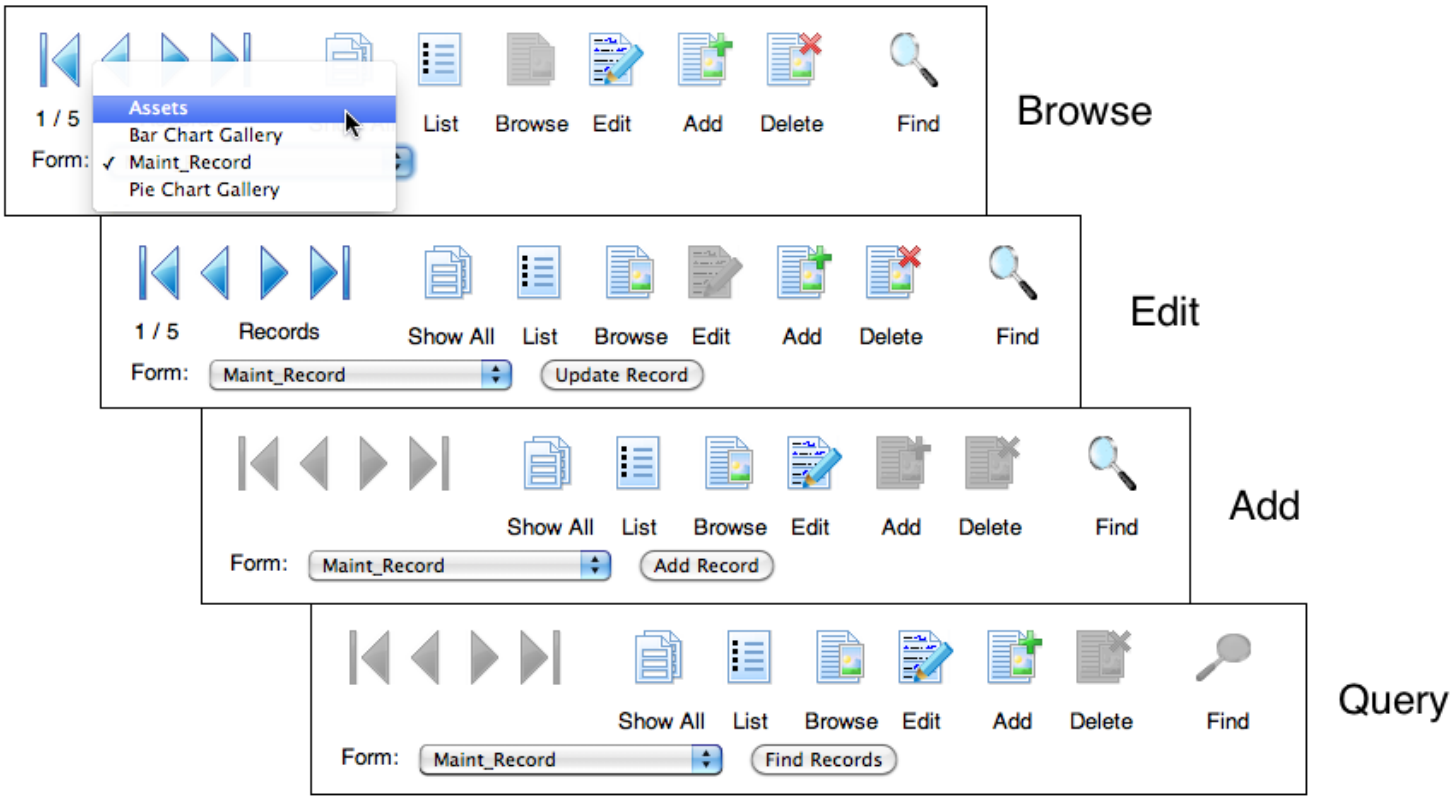
Enabling the Navigation Toolbar



Enabling the Navigation Toolbar option, generates code and views to create a form and record navigation toolbar which will be displayed at the top of each form. All button scripts which include navigation (First, Last, Previous, Last record navigation, Goto Layout, Switch to Browse Mode, switch to find Mode) will also be modified to direct the user thru the toolbar controller to insure that the toolbar is properly displayed.

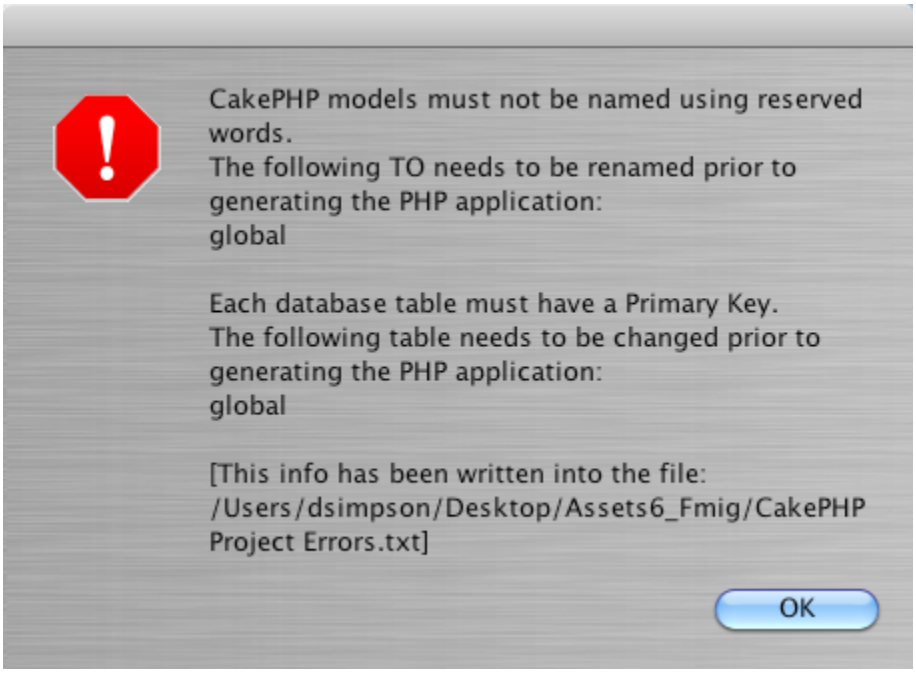
Form submission for Add, Edit, Query operating modes is done thru clicking the Submit button on the toolbar, instead of the submit button on the underlying form. There is no Submit button placed onto the form itself if the navigation toolbar is being used.

4 Navigation Toolbars



These screenshots display toolbars having Data Access = "All" enabled. The navigation toolbar icons are changed to remove support of features which are not enabled by the Data Access option.

Preflight Check Dialog



FmPro Migrator performs a couple of checks on the imported TOs and base tables prior to generating PHP scripts. These checks include:
1) Verification that each base table contains a primary key column. FmPro Migrator looks for the first table column having Unique and Not Empty validation. If multiple columns have these same validation properties, the Primary Key column can be set manually for each table by double-clicking on the column name in the list of fields on the Tables tab of the Migration Process window.

2) Acheck is made to insure that none of the TOs in the database have the same name as a CakePHP reserved word. The following 56 reserved words are checked for a conflict:

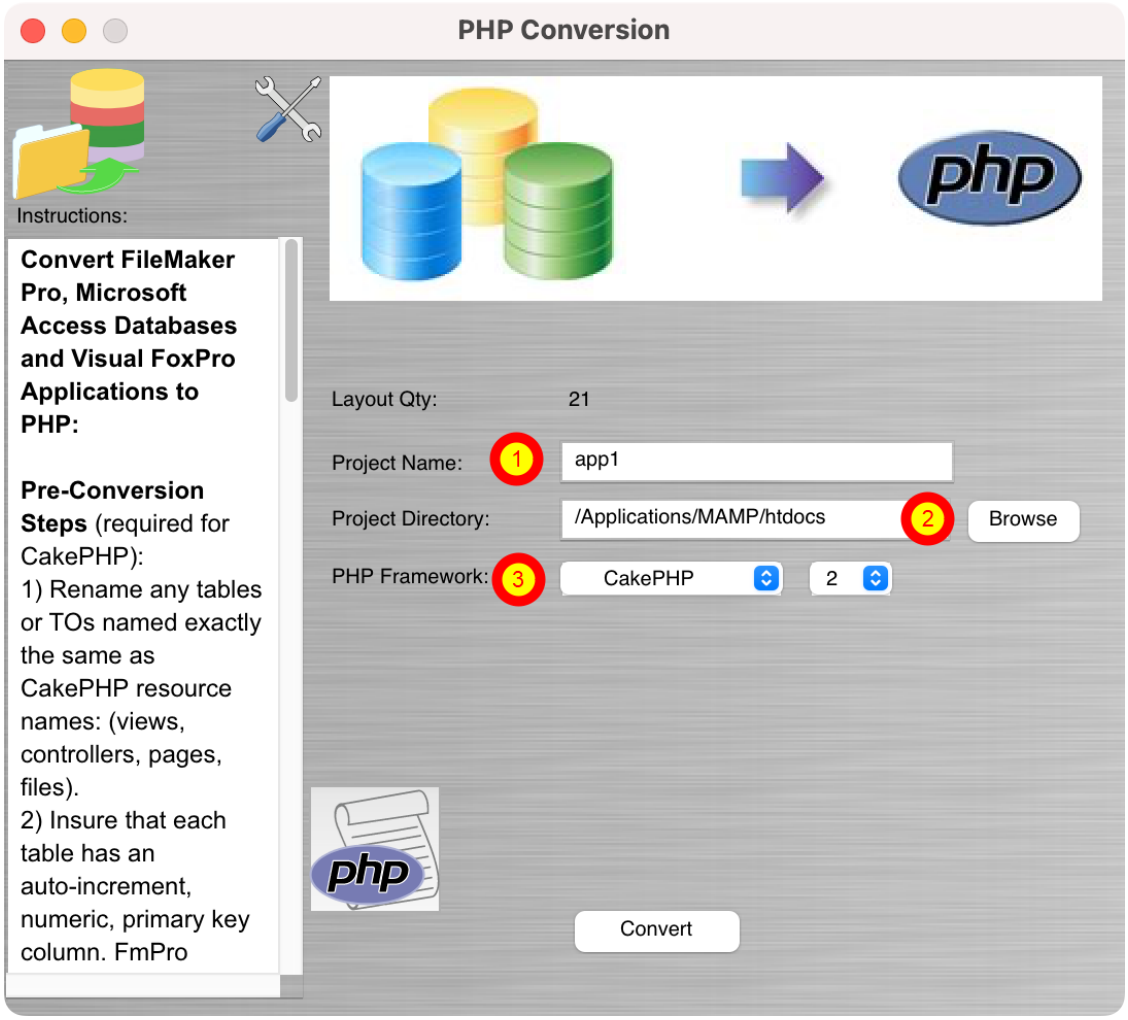
view,controller,page,global,__,a,aa,am,config,convertSlash,countdim,debug,e,env,fileExistsInPath,h

Therefore, it is recommended that developers should import the tables, relationships, value lists and only 1 layout into FmPro Migrator. Then perform a test conversion so that FmPro Migrator can perform the pre-flight test on the project. Manually make changes to the FileMaker database, reimport into FmPro Migrator and test with 1 layout again. Perform this task interactively until the preflight test passes. Then import the rest of the layouts and scripts into FmPro Migrator in order to generate the entire PHP web application.

Using Licensed Mode - PHP Conversion

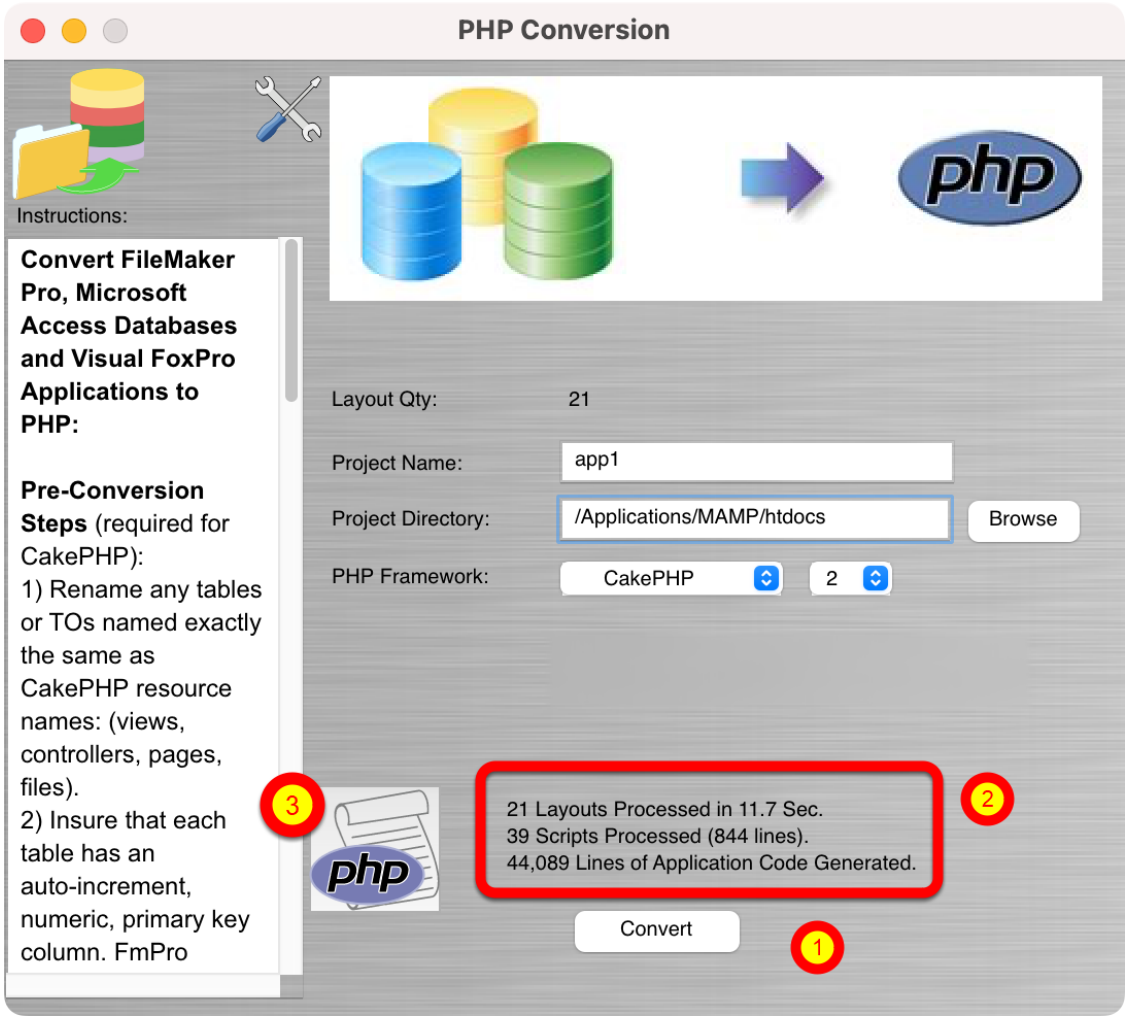
Using the PHP Conversion feature in Demo mode, limits processing tasks to 5 Layouts and scripts. Entering the license key supplied with FmPro Migrator enables processing the number of layouts supported by the license key. Using the License Key allows for the processing of an unlimited number of database files for the purchased Layouts quantity during the duration of the license key.

PHP Conversion Properties



(1) Enter a name for the new PHP web project which will be created by FmPro Migrator. The project name will be used as the web directory name which will be created for the project. If the project folder already exists, it will be re-used, otherwise, it will be created and the php framework files will be copied into the directory during the conversion.
(2) Select an output directory where the generated project files will be written. Anew folder having the same name as the project will be created within the selected Project Directory folder.
(3) Select the PHP framework name.

Migration Statistics



(1) Clicking the Convert button performs the conversion of the Layouts in the project with the (2) resulting processing statistics displayed below the license key field.
(3) The Code Conversion Workbench button performs an overall better conversion of the scripts into PHP by using machine learning models - which is covered in the Code Conversion Workbench manual.

About the Generated PHP Web Application

PHP Conversion - Database Configuration Notes

SQL Server Configuration Notes

1) Install SQL Server Native Client (v11 for SQL Server 2012 - which gets installed automatically in System32
2) Configure php.ini using:
extension=php_sqlsrv_53_ts.dll
3) Make sure XAMPP 1.77 or higher is used - in order to use the vc9 compiled versions of Native Client 11, with SQL Server 2012.
4) Restart web server
5) Install newest dob_sqlsrv.php file from gitHub.
6) The database schema should already be created with the name of the login username, and create a top-level login for the overall server using the login name. These steps should have been completed during the data conversion process.
7) Change the created tables to be owned by dbo. instead of the login user.

Oracle Configuration Notes

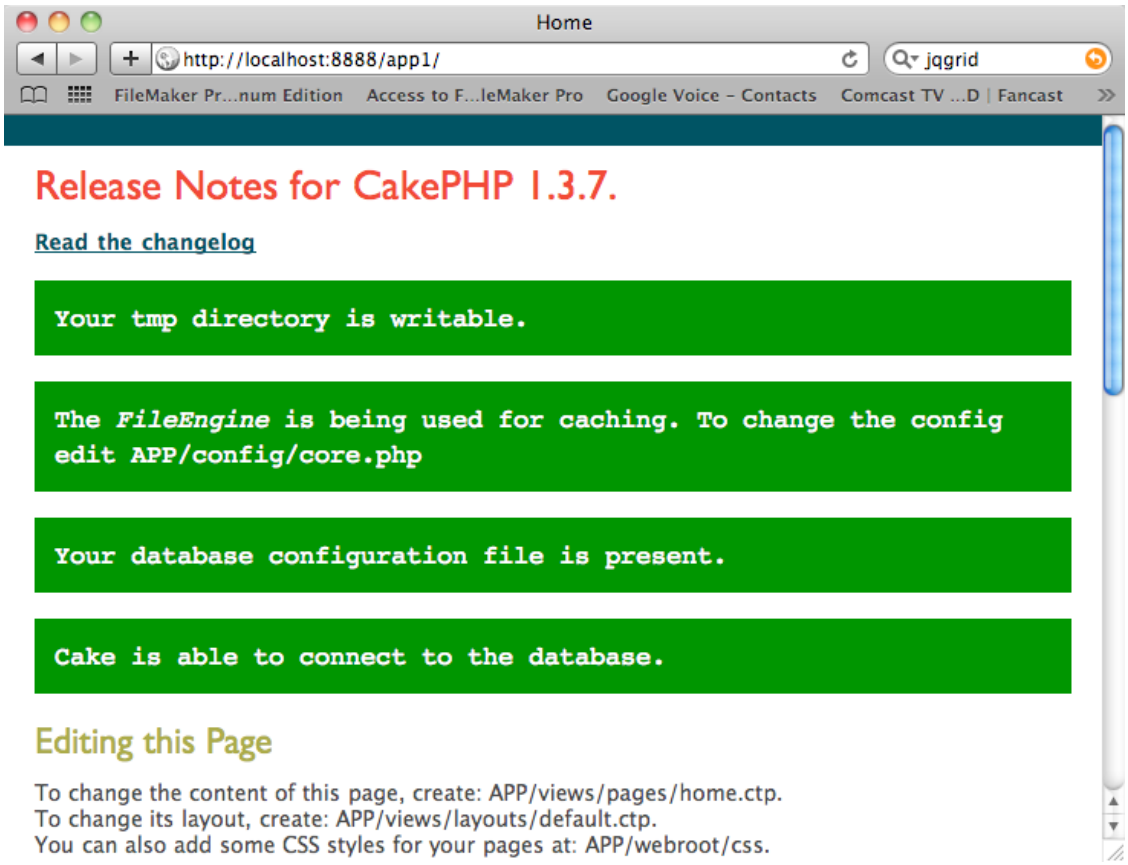
1) If XAMPP is being used on Windows, enable the oci8.dll. The oci8 feature is already included in XAMPP and only needs to be enabled after installing the Oracle Instant Client.
2) If MAMP or XAMPP is being used on MacOSX, then install the oci8.so:
3) Install the Oracle Instant Client and the associated SDK.
4) Download and install the PHP source files for the version of PHP used in MAMP/XAMP.
5) Using MAMP pathnames, this commands are:
./configure in / Applications/MAMP/bin/php/php5.3.6/include/php
6) /Applications/MAMP/bin/php/php5.3.6/bin/pecl install oci8
7) Add this line to the php.ini file:
extension=oci8.so
8) Set the DYLD_LIBRARY_PATH so that the Oracle Instant Client libraries can be found:
9) Restart the Apache web server.
10) Check the output of phpinfo to verify that oci8 has been loaded.

For this manual, an installation of XAMPP or MAMP has been installed, and is running on the local computer at port# 8888.

Opening the app1 Web Application

Open your web browser to your local web site where the PHP web application has been generated. Click on the app1 directory, which is the web application directory generated by FmPro Migrator.

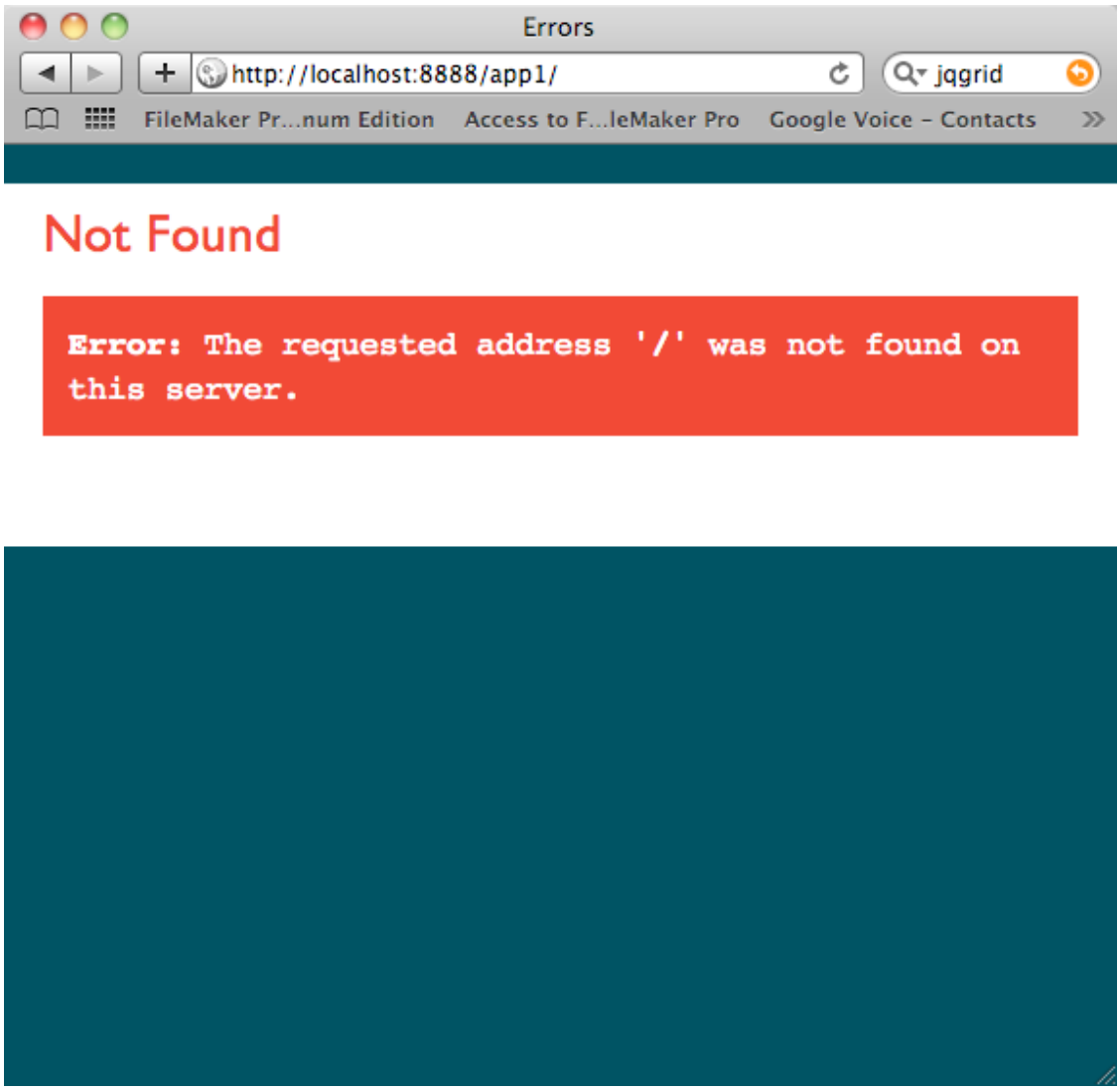
CakePHP Startup Page



If the debug configuration = 2 (which is the default value used by FmPro Migrator), the CakePHP startup page will be displayed. FmPro Migrator has automatically configured the database connection, using the Destination Database info entered into the FileMaker tab of FmPro Migrator. The default salt and cipher-seed values in the core.php file have also been changed to random values, which are different for each project.

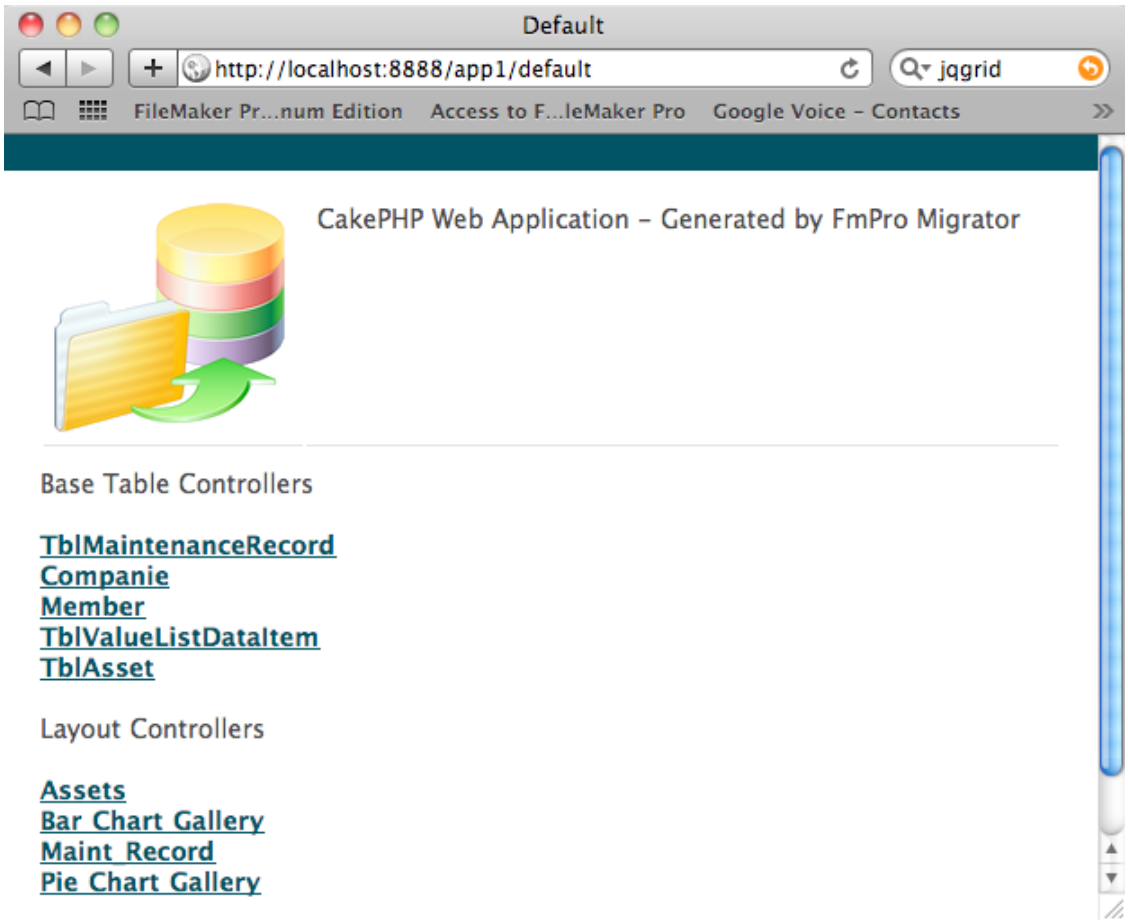
Proceed to enter in "/default" into the web browser to start using the web application.

Debug = 0 Result



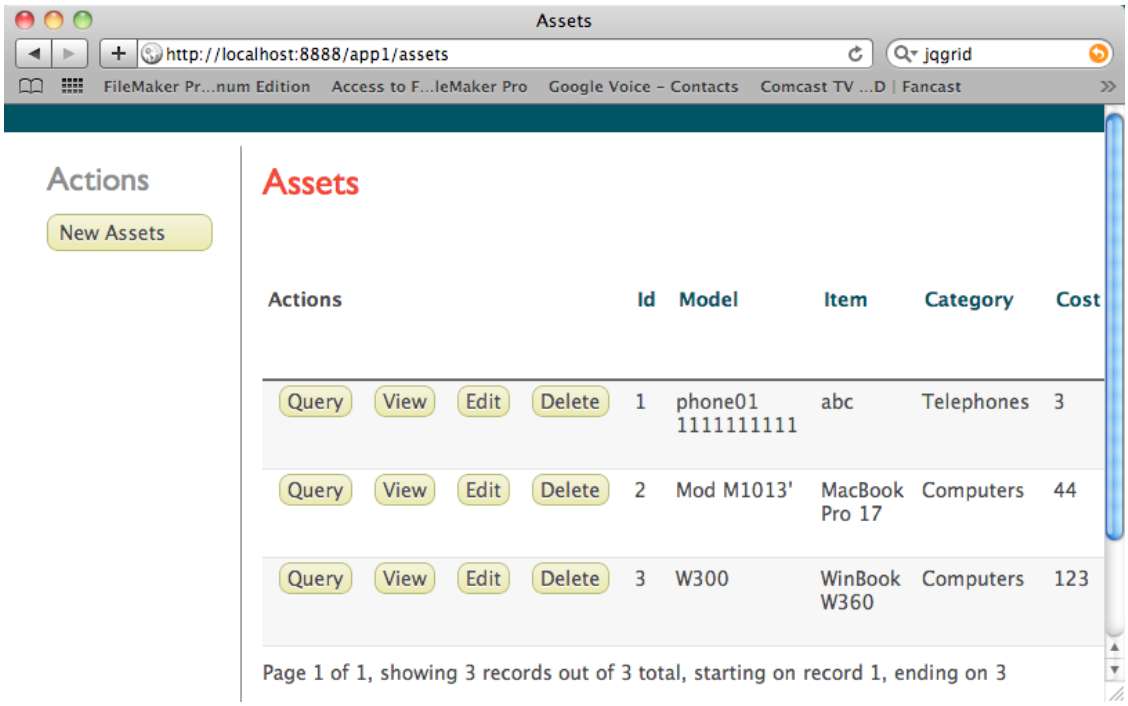
If the debug value in the core.php file is configured with a value = 0, you will get an error message that the "address '/' was not found". Proceed to enter in "/admin" or "/default" into the web browser to start using the web application. The debug value must be set = 0 to view web pages containing converted charts.

Default Web Application Page



FmPro Migrator creates a default page file: /app/views/pages/default.ctp. This default page includes links to controllers created for each Base Table and another set of links created for each converted Layout in the project.

Assets Layout Controller - index.ctp view



Clicking on the Assets layout controller, opens the index view.
Clicking any one of the Actions buttons, will open the associated view for the selected action.
Query - Enables searching the records of the database.
View - Displays a read-only view of the record in the database.
Edit - Displays an updatable view of the record in the database.
Delete - Prompts with a warning dialog, then deletes the record in the database.

Files Which Are Preserved When Re-Generating the Application

It is likely that the conversion process will be iterative, because cosmetic changes may need to be made to the FileMaker layouts in order to improve the generated output view files. For this reason, some globally used web application files are only generated once, and are not overwritten on subsequent project re-generations.

Database.php

The app/config/database.php file contains the database server configuration information. This information might be manually changed in order to switch between local test servers, staging servers or production servers during the development and testing process.

Core.php

The app/config/core.php file contains the application Configure::write('debug', 2); status, which by default is set = 2. By setting this value to 2, the project displays the green welcome screen showing that the database connection has been made, and that the salt value has been changed from its default value.
The debug configuration needs to be changed to 0, in order for charts to be displayed in the web browser.

Generated Report Files

CakePHP Project Errors.txt

Each database table must have a primary key column named ID. If this criteria is not met, then an error dialog is displayed and the CakePHP Project Errors.txt file is written to the output directory with the names of the tables which need to be corrected.

Conversion Summary Results.txt

This file contains a summary of the project stats, including:

?? Layouts Processed in ?? Sec.
?? Scripts Processed (?? lines).
?? Lines of Application Code Generated.

Where ?? symbols are replaced with the actual project stats. This info can be useful for project estimating purposes.

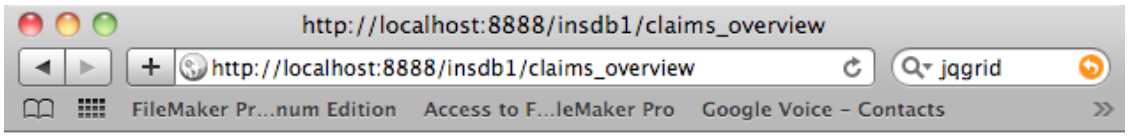
create_indexes.sql

FmPro Migrator generates the create_indexes.sql file to enable DBAs to generate indexes for foreign key columns once the data has been transferred from FileMaker Pro to the SQL database server. Typically, adding these indexes can improve web application performance by 6% - 50%.

Duplicate Objects Report.xls

During the processing of layouts, FmPro Migrator checks for the existence of duplicated layouts throughout the project. If two layouts have exactly the same contents, the 2nd occurrence of the layout is marked as a duplicate layout. This report can help you spot accidental duplications which may have occurred during the capturing of the layouts from FileMaker Pro.
If two layouts have different contents but they have the same name, then the 2nd layout gets renamed, and listed in the report with the new name.

Missing Table Report.txt



Parse error: syntax error, unexpected T_OBJECT_OPERATOR, expecting T_STRING or T_VARIABLE or '{' or '$' in /Applications/MAMP/htdocs/insdb1/app/controllers/claims_overview_controller.php on line *10*

Call Stack

#	Time	Memory	Function	Location
1	0.0000	324376	{main}( )	../index.php:0
2	0.0474	664264	Dispatcher->dispatch( )	../index.php:83
3	0.0489	675420	Dispatcher->__getController( )	../dispatcher.php:116
4	0.0489	675448	Dispatcher->__loadController( )	../dispatcher.php:382
5	0.0490	676252	App->import( )	../dispatcher.php:410
6	0.0740	3607772	App->__find( )	../configure.php:954
7	0.0740	3608224	App->__load( )	../configure.php:1035

Each layout must be assigned to a table in FileMaker Pro prior to performing a successful conversion. Without an assigned table, the controller can't be configured to use the correct model, which causes the Parse error on line 10 displayed in this screenshot.

The report also attempts to locate fields which have missing table name or field name attributes set correctly.

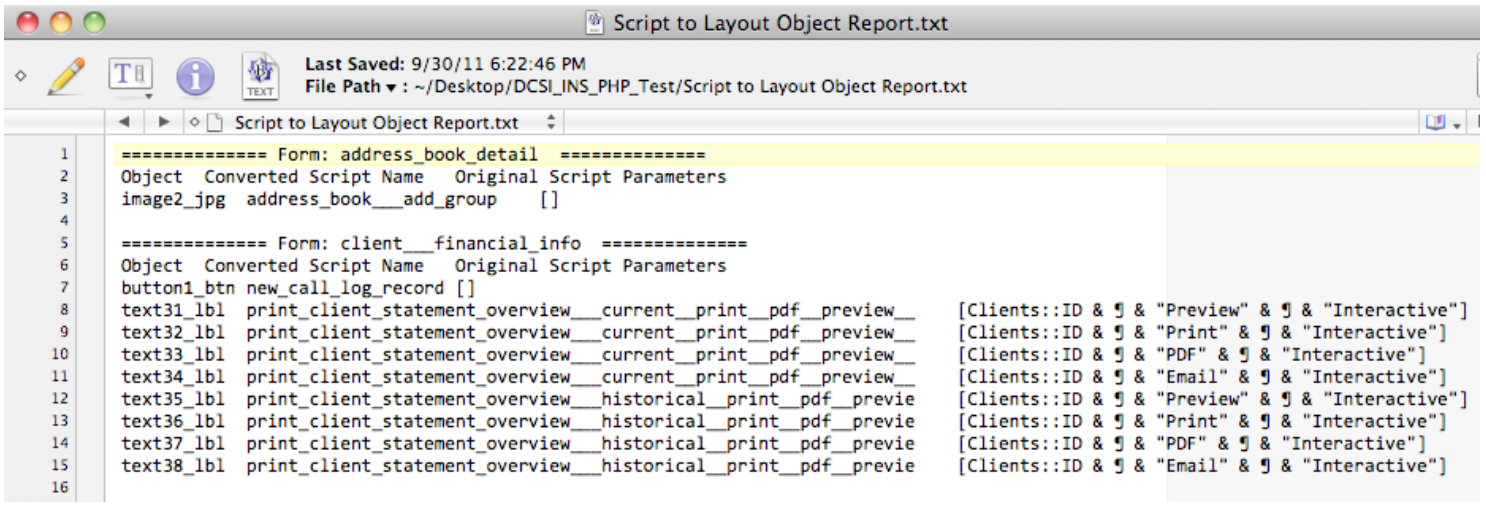
The following text shows an example of the contents of this report:

Object Name Form Name
    address_labels
2 address_labels

Total Layouts = 86
Layouts with Missing Tables = 4
Layout Name

binder__new_business
claims_overview
envelope
policy_subjectivity

Script to Layout Object Report.txt

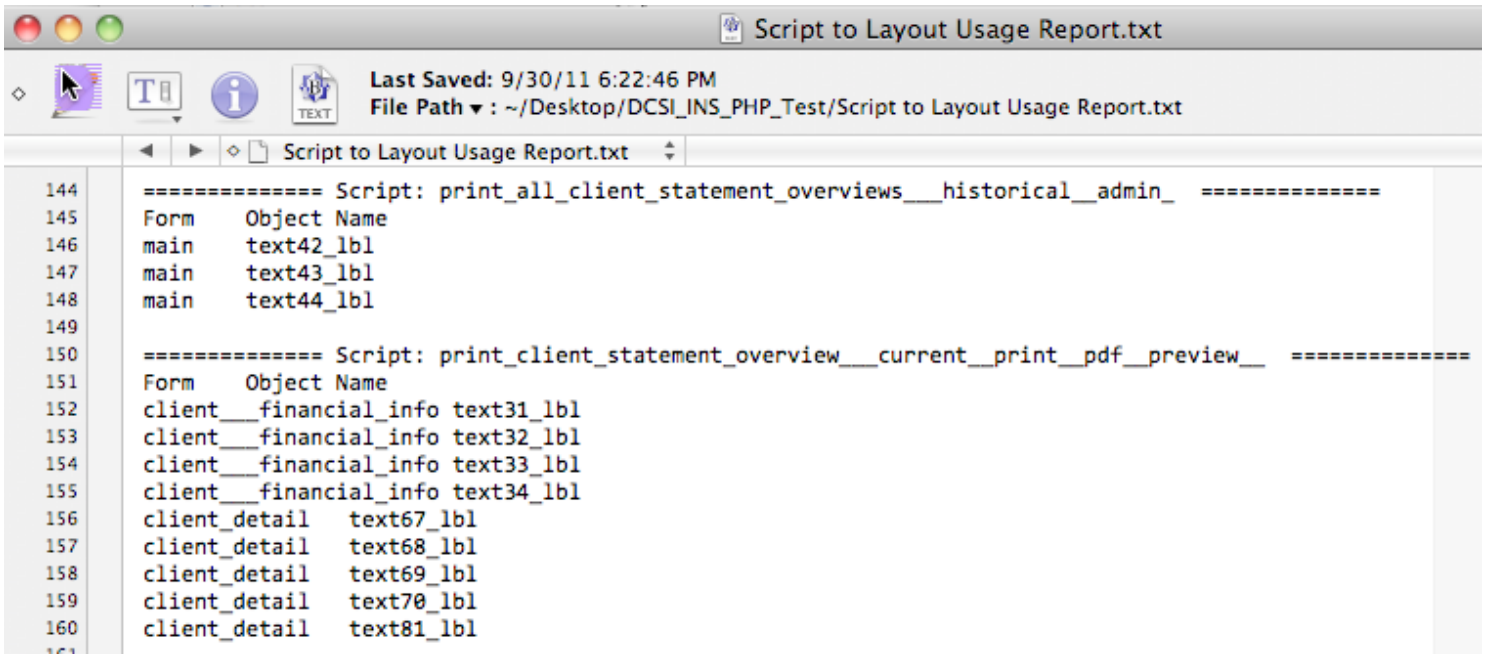



```
1  ============== Form: address_book_detail  ==============
2  Object  Converted Script Name   Original Script Parameters
3  image2_jpg  address_book___add_group    []
4
5  ============== Form: client___financial_info  ==============
6  Object  Converted Script Name   Original Script Parameters
7  button1_btn new_call_log_record []
8  text31_lbl  print_client_statement_overview___current__print__pdf__preview__    [Clients::ID & ¶ & "Preview" & ¶ & "Interactive"]
9  text32_lbl  print_client_statement_overview___current__print__pdf__preview__    [Clients::ID & ¶ & "Print" & ¶ & "Interactive"]
10 text33_lbl  print_client_statement_overview___current__print__pdf__preview__    [Clients::ID & ¶ & "PDF" & ¶ & "Interactive"]
11 text34_lbl  print_client_statement_overview___current__print__pdf__preview__    [Clients::ID & ¶ & "Email" & ¶ & "Interactive"]
12 text35_lbl  print_client_statement_overview___historical__print__pdf__previe    [Clients::ID & ¶ & "Preview" & ¶ & "Interactive"]
13 text36_lbl  print_client_statement_overview___historical__print__pdf__previe    [Clients::ID & ¶ & "Print" & ¶ & "Interactive"]
14 text37_lbl  print_client_statement_overview___historical__print__pdf__previe    [Clients::ID & ¶ & "PDF" & ¶ & "Interactive"]
15 text38_lbl  print_client_statement_overview___historical__print__pdf__previe    [Clients::ID & ¶ & "Email" & ¶ & "Interactive"]
16
```

The Script to Layout Object Report.txt file lists the usage of each Perform Script script step for all objects within the layouts of the original FileMaker database. The converted object name, converted script name and script parameters are listed. Each script is executed as a function call within the current controller file or AppController. By default, the converted scripts are written into the app/converted_scripts directory when the web application is created. Therefore these scripts will generally need to be modified and copied into the AppController, layout controller or a helper file in order for it to be found.

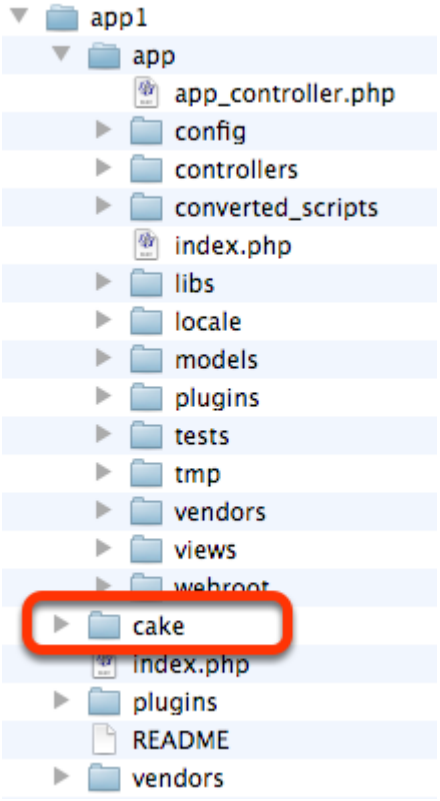
Layout objects which use the Perform Script step have the script name and parameters written into the comment for the object, in order to prevent the possibility of errors while running the converted application.

Script to Layout Usage Report.txt



The Script to Layout Usage Report.txt file lists every layout object which uses each script within the database. This info provides a quick reference for where the script is used throughout the application in order to assist the PHP developer concerning the placement of the script in the web application. For instance, a script which is only used within a single layout controller, should probably be written into the layout's controller file. Ascript which is used in many places within the application could be placed within the AppController or a helper file.

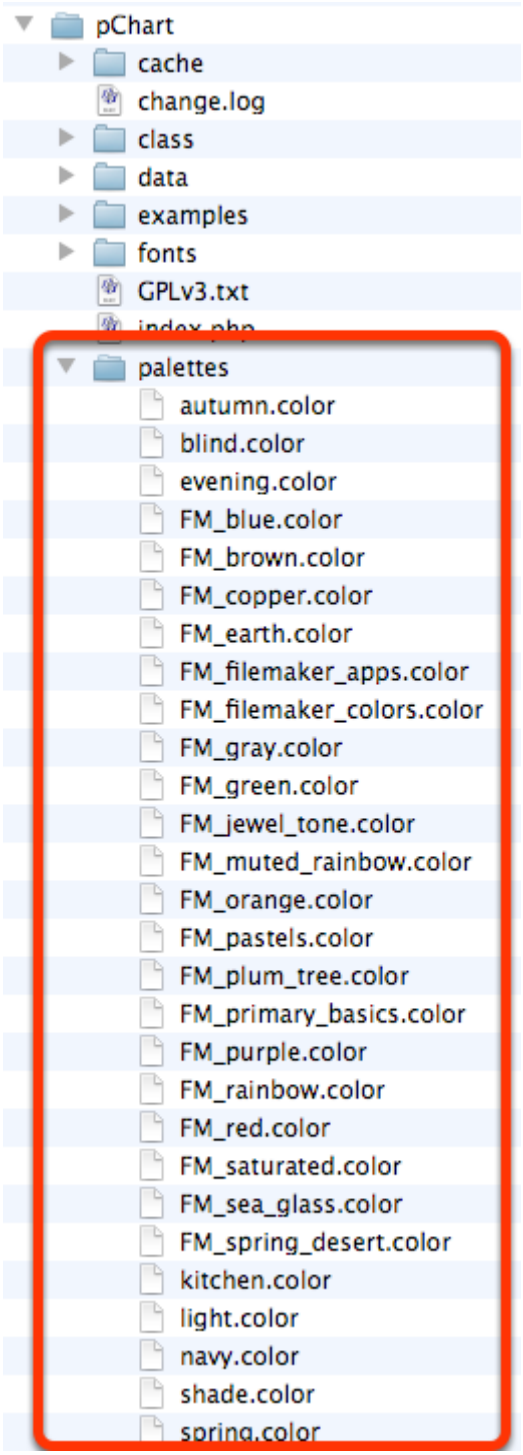
CakePHP Framework Files



To upgrade the version of CakePHP used within the generated web application, download the latest version of the CakePHP framework and replace the cake directory within the application.

Note: FmPro Migrator currently supports CakePHP 1.3x versions. CakePHP 2.0 represents a major rewrite, requiring different file naming and other changes throughout the web application. A command line tool is provided with CakePHP 2.0 which can make many of these changes automatically. If you use this automated upgrade tool, please provide some feedback to .com Solutions Inc. regarding the results.

pChart Framework



The pChart Framework can be upgraded by replacing the pChart directory at the top-level of your htdocs/public_html directory, where FmPro Migrator placed it. Before deleting the old directory, copy the palettes directory to your new pChart directory. FmPro Migrator uses a customized version of this directory which contains 20 files named FM_??.color to represent the color schemes used to create FileMaker 11 compatible charts.

Fusion Charts

No modifications occur to the Fusion Charts directory by FmPro Migrator. At this time, the Fusion Charts directory needs to be copied manually into your web directory, and by default is expected to be located at the top level of the web server htdocs/public_html directory.

jquery.js

The app/webroot/js/jquery/jquery.js file written by FmPro Migrator is version 1.5.1. This version cannot be upgraded to version 1.6 at this time, or the jquery.datePicker.js file will stop working.

jqGrid

FmPro Migrator copies your selected jqGrid/jqSuite directory, and moves files around as necessary whenever it creates a new project. This process only occurs if a new project directory needs created. It will be necessary to look thru the webroot directory to duplicate this effort manually.

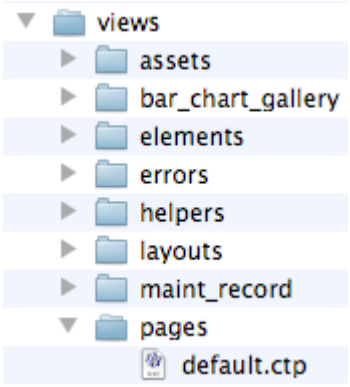
Everything Else...

All of the other PHP, JavaScript and .css files should be able to be upgraded without difficulty. For instance the Adobe Spry framework files and Gritter.js files are unchanged by FmPro Migrator. But the flash_info.ctp and flash_error.ctp files which use jquery.gritter.js have been customized.

Changing the Application Root Web Page

Adding a new view file <app name>/app/views/pages/home.ctp will replace the standard CakePHP diagnostic page which is displayed when debugging is enabled.

Changing the Application Default Page



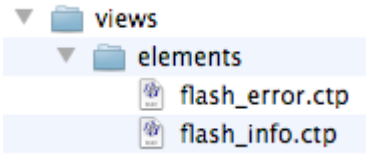
Changing "default" routing:
The <app name>/app/config/routes.php file contains a default route for the "default" action, listed as follows:

```
// Temporary route for initial startup page created by FmPro Migrator - remove for production use
Router::connect('/default', array('controller' => 'pages', 'action' => 'display', 'default'));
```

This is intended for development, and should probably be commented out for production use.

It is possible to change the FmPro Migrator generated index page by updating or replacing the file: <app name>/app/views/pages/default.ctp

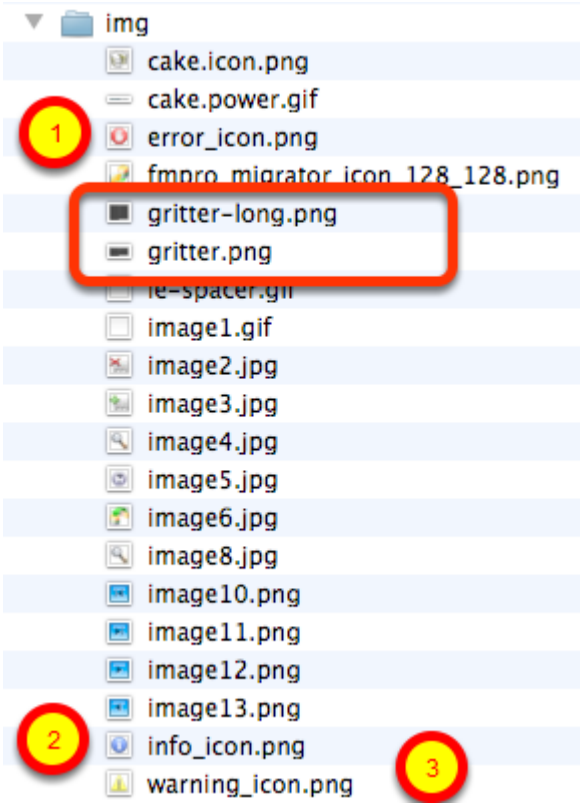
Gritter Dialogs - Timeout/Sticky Settings



Flash messages are directed to either the flash_error.ctp or flash_info.ctp files located within the views/elements folder.
By default, the flash_info.ctp file includes a timeout value which causes the window to fade out after a few seconds.

The flash_error.ctp file has the sticky = true; property set, which means that the dialog stays on the screen until it is clicked by the user.

Gritter Dialogs - Background Image Color, Size, Transparency



By default, the Gritter dialogs use the app/webroot/img/gritter.png file, but they can be changed to use the gritter-long.png file if needed. In most cases, the Gritter dialogs expand automatically to handle more text, so the gritter-long.png file may not be required.
Furthermore, the transparency or color of the background image may also be changed. Or different dialogs could be configured to use background image files having a different color or transparency. This functionality could be made dependent upon the business logic of the web application. Dialogs in one part of an application could have a specific look and other parts of the app might have a slightly different look serving as a visual reminder to the user concerning either the type of message or the part of the application which generated the message.

The (1) error_icon.png, (2) info_icon.png or (3) warning_icon.png files could also be changed. The warning_icon.png file is not currently used by the application, but it could be used in the design of a new view (i.e. flash_warning.ctp).

Changing Value List Values

```
function define_condition_list_value_list() {
    // Value List: condition_list
    $condition_list_value_list = array("New"=>"New","Excellent"=>"Excellent","Good"=>"Good","Poor"=>"Poor","Scrap"=>"Scrap");
    return $condition_list_value_list;
    }

function define_general1_0_value_list() {
    // Value List: general1_0
    $general1_0_value_list = array("1"=>"");
    return $general1_0_value_list;
    }
```

Value lists in the generated PHP web application are defined in a central location, the app/app_controller.php file. This means that changing the definition of a value list in this one single location, will affect every menu, checkbox set or radio button set which uses the value list throughout the entire application - just like editing a value list within the FileMaker Manage Value Lists dialog.

By default, a custom value list consisting of a single value = 1, will be created with an empty display value. This makes it possible to use the value list for single checkbox objects, where the value does not need to be displayed because there is already a separate text label next to the checkbox.

Web Browser Compatibility

The following notes document cosmetic issues discovered with various web browsers and the generated PHP web application.

Summary

1) BLOB column images are only displayed within IE8+, Safari, Chrome and FireFox browsers.
2) The positions of the FusionCharts displayed on the Pie Chart Gallery, and Bar Chart Gallery in the app1 solution are only displayed correctly within Safari on MacOS X. This seems to be an issue with the layout of the objects on these two forms, since object position issues have not be noticed anywhere else.
3) Cosmetic issues may occur with FusionCharts fallback javascript charts, except on Safari. Using pChart generated charts is recommended if charts need to look identical across platforms and different mobile devices.
4) The jqGrid generated PDF files only seem to display data correctly within the MacOS X Preview application. The jqGrid does work in IE9 on Windows 7.
5) PNG images having transparency only display correctly with IE7 and higher browsers.
6) pChart generated charts always work perfectly across all browsers and the charts are always displayed in the correct position on the web page.
7) Playing of the Beep sound only works on Safari on IOS when displaying a player object.
8) ExtJS 4 Grid objects work correctly across all tested browsers shown here. To scroll the list of records within the ExtJS grid on touch devices (iPhone/iPad etc), use a two finger swipe.
9) The CCS implemented tooltips, don't display a drop shadow on IE browsers, but otherwise the tooltips work perfectly.

In general, all of the commonly used web application features like menus, navigation toolbar, tab controls, colors, vector graphic objects, object positions work perfectly going all they way back to IE6 on Windows XP.

IE6 on Windows XP [Deprecated Browser]

1) Images within BLOB columns are not displayed.
2) The CakePHP index page buttons are displayed as links, instead of being styled as buttons.
3) pChart Charts - all charts are displayed perfectly.
4) FusionCharts - Without FlashPlayer installed - Display with cosmetic issues with chart titles/legends. All charts were displayed, but were displayed at the wrong position on the page. Charts were displayed staggered down the page and offset to the right - but only on the test chart web pages.
5) FusionCharts - With Flash Player X installed - Display much better. All charts were displayed, but were displayed at the wrong position on the page. Charts were displayed staggered down the page and offset to the right - but only on the test chart web pages. Tab controls with embedded tab controls work perfectly.

6) Core2CRM converted solution - No BLOB columns displayed, vector graphics objects displayed perfectly, jqGrid portals displayed correctly. Export jqGrid to PDF - displayed the title of the exported data using Adobe Reader 10, but no actual data was displayed.
7) Export jqGrid to PDF - displayed the title of the exported data using Adobe Reader 10, but no actual data was displayed.
8) INSDB converted solution - Takes 30 seconds to 2 minutes to display all 6 portals when advancing between records, depending upon the performance of the test computer. PNG images having transparency are not displayed correctly, and show a grey background where they should be transparent.
9) Beep sound plays correctly, Toolbar displays and operates correctly [though the Add/Update button is cropped by a few pixels].
10) JavaScript Print dialog doesn't work, and displays an error.
11) Open URL works correctly for image link objects and JavaScript implementation (as implemented via button, vector graphics objects).
[Deprecated Browser - No new features are tested with IE6]

IE7 on Windows XP

1) Images within BLOB columns are not displayed.
2) The CakePHP index page buttons are displayed correctly as buttons.
3) pChart Charts - all charts are displayed perfectly.
4) FusionCharts - Without FlashPlayer installed - Display with cosmetic issues with chart titles/legends. But in general, most chart elements displayed correctly. All charts were displayed, but were displayed at the wrong position on the page. Charts were displayed staggered down the page and offset to the right - but only on the test chart web pages.
5) Core2CRM converted solution - No BLOB columns displayed, vector graphics objects displayed perfectly, jqGrid portals displayed correctly. Takes only a few seconds to advance between records. Tab controls with embedded tab controls work perfectly.
6) INSDB converted solution - Takes only a few seconds to advance between records, displaying all 6 portals and data. PNG images with transparency work perfectly.
7) Beep sound plays correctly, Toolbar displays and operates correctly [though the Add/Update button is cropped by a few pixels].
8) JavaScript Print dialog doesn't work, and displays an error.
9) Open URL works via JavaScript to open the web page under the Navigation Toolbar frame (as implemented via button, vector graphics objects). Open URL as a link using a button object - does not work.

IE8 on Windows XP

1) Images within BLOB columns are displayed perfectly.
2) The CakePHP index page buttons are displayed correctly as buttons.
3) pChart Charts - all charts are displayed perfectly.

4) FusionCharts - With FlashPlayer Installed - Some cosmetic issues with display of data fields and the positions of the charts on the page, otherwise the charts are displayed correctly.
5) Core2CRM converted solution - All features work perfectly, including tab controls within tab controls, jquery datepicker, jqGrid.
6) INSDB converted solution - Takes only a few seconds to advance between records, displaying all 6 portals and data. PNG images with transparency work perfectly.
7) Export jqGrid to PDF - displayed the title of the exported data using Adobe Reader 10, but no actual data was displayed.
8) JavaScript print dialog works, Beep sound plays correctly, Navigation Toolbar displays and works correctly.
9) Open URL works correctly for image link objects and JavaScript implementation (as implemented via button, vector graphics objects).

IE9 on Windows 7

1) Images within BLOB columns are displayed perfectly.
2) The CakePHP index page buttons are displayed correctly as buttons.
3) pChart Charts - all charts are displayed perfectly.
4) FusionCharts - Without FlashPlayer - Some cosmetic issues with display of data fields and the positions of the charts on the page, otherwise the charts are displayed correctly.
5) jqGrid - Does not display at all in any solution. This is not a reproducible problem by the jqGrid developer, so this may be a machine specific issue.
6) Core2CRM converted solution - All features work perfectly, including tab controls within tab controls, jquery datepicker, except for the jqGrid.
7) JavaScript print dialog works, Beep sound plays correctly, Navigation Toolbar displays and works correctly.
8) Open URL works correctly for image objects and JavaScript implementation (as implemented via button, vector graphics objects).

Safari 5 - Windows

1) Images within BLOB columns are displayed perfectly.
2) The CakePHP index page buttons are displayed correctly as buttons.
3) pChart Charts - all charts are displayed perfectly.
4) FusionCharts - With FlashPlayer Installed - Some cosmetic issues with display of data fields and the positions of the charts on the page, otherwise the charts are displayed correctly.
5) Core2CRM converted solution - All features work perfectly, including tab controls within tab controls, jquery datepicker, jqGrid.
6) INSDB converted solution - Takes only a few seconds to advance between records, displaying all 6 portals and data. PNG images with transparency work perfectly.
7) Beep sound does not play.
8) Toolbar displays and operates correctly, JavaScript Print dialog works.
9) Open URL works via JavaScript to open the web page under the Navigation Toolbar frame (as

implemented via button, vector graphics objects). Open URL as a link using an image object - works correctly and opens in a new window.

Safari 5 on MacOS X

All features displayed correctly, all charts displayed in the correct position.

Safari 5 on IOS 5

All features work correctly - except playing Beep sound.
1) Beep sound does not play, unless the HTML5 media player object is displayed for the user to click.
2) To scroll the list of records within the ExtJS grid on touch devices (iPhone/iPad etc), use a two finger swipe.

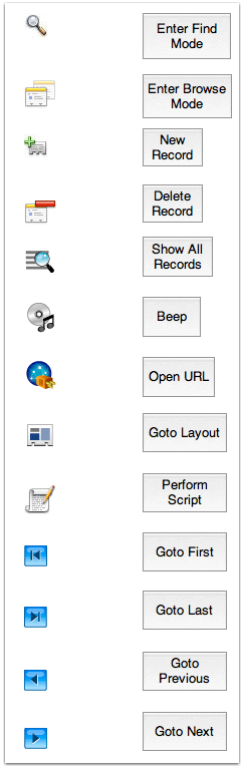
FireFox - MacOS X

All features work correctly, except FusionCharts are displayed in the wrong positions on the web page.

Google Chrome on MacOS X

All features work correctly.
1) JavaScript print dialog works, Beep sound plays correctly, Navigation Toolbar displays and works correctly.
2) Open URL works correctly for image link objects and JavaScript implementation (as implemented via button, vector graphics objects).

14 Commonly Used Script Steps



Virtually every object on a FileMaker layout can have an assigned script step, including: button, image, text label, rectangle, rounded rectangle, circle/oval and line objects. Within the converted web application these objects include PHP and JavaScript button scripting to implement the original functionality.

In addition to object scripting features, two report files are generated to provide script and object dependency reporting for each layout and script within the converted solution.

Supported layout object script steps include:
• Enter Find Mode - Redirects the user to the Query form for the same layout.

• Enter Browse Mode - Redirects the user to the View form for the same layout, using the record stored within the $_SESSION['current_record'] variable for the current model.

• New Record - Redirects the user to the Add form for the same layout.

• Delete Record - Prompts for confirmation, then deletes the current record if the user clicks Ok.

• Show All Records - Clears the $_SESSION['foundset'] array for the current model.

• Beep - Uses Javascript code to download and play the included beep1.wav sound file. The implementation of this feature also makes it possible to specify additional sound files for different tasks within the web application.
Note: The beep1.wav or beep1.mp3 file does not play automatically on IOS devices (iPad/iPhone), IE8, or IE9, but it can be played manually by displaying the audio controller using hidden="false" in the document.ready JavaScript code.

• Open URL - Opens the specified web page URL in a new browser window.

• Goto Layout - Opens the View form for the specified converted layout, and opens the most recently visited record. If a form using this model has not previously been visited, the first record in the model will be displayed.

• Perform Script - Executes the specified converted script as a function call within the current controller file or AppController.
By default, the converted scripts are written into the app/converted_scripts directory when the web application is created. Therefore these scripts will generally need to be modified and copied into the AppController, layout controller or a helper file in order for it to be executed properly.

Layout objects which use the Perform Script step have the script name and parameters written into the comment for the object, in order to prevent the possibility of errors while running the converted application. These parameters will need to be edited and passed to the function which is being called.
<!-- Button: button22_btn Script: go_to_form_layout("Parameter1" & ¶ & "Parameter2" &¶ & "Parameter3") -->

The Script to Layout Usage Report.txt file lists every layout object having a Perform Script step within the database. This info provides a quick reference for where the script is used throughout the application in order to assist the PHP developer regarding the placement of the script in the web application. For instance, a script which is only used within a single layout controller, should probably be written into the layout's controller file. Ascript which is used in many places within the application could be placed within the AppController or a helper file.

• Print - Opens the web browser print dialog to print the currently displayed form.

Record navigation script steps include: Goto Record/Request [First], Goto Record/Request [Last], Goto Record/Request [Previous], Goto Record/Request [Next].

JavaScript Compatibility Notes & Image Objects

Many of the supported script steps are implemented with JavaScript code implemented within the Document Ready function at the top of each view.ctp, add.ctp, edit.ctp view file.
The script steps listed below, are implemented as direct links implemented via CakePHP for

graphic image objects:
Enter Find Mode
Enter Browse Mode
New Record
Show All Records
Open URL
Goto Layout
Goto Record/Request [First], Goto Record/Request [Last], Goto Record/Request [Previous], Goto Record/Request [Next]

If you want to achieve better cross-platform, cross-browser compatibility, you may want to implement your scripted objects on FileMaker layouts by using image objects instead of using button, text or vector graphic objects which will be implemented purely in JavaScript code.
The above listed script steps will also work if JavaScript as been disabled within the web browser, if the script has been attached to an image object on the layout.

Using the ExtJS Grid

FmPro Migrator Platinum Edition 6.68 introduces the use of the ExtJS 4 grid object, as an alternative to using the jqGrid object for the PHP web implementation of FileMaker portal objects. The ExtJS grid offers full compatibility with any database server which is compatible with CakePHP (including FileMaker Server Advanced using an ODBC connection).

Some advantages to using the ExtJS grid include:
1) Only one connection is made to the database for the CakePHP application overall, no additional separate connection gets made to the SQL database server. This reduces configuration issues.
2) No URL dependencies. The ExtJS grid code reads the URL differently, so that it is not sensitive the application's URL prefix.
3) [Future Implementation] - The ExtJS grid offers the potential to handle relationships of any level of complexity, for which CakePHP models have been created. All related records are already available in memory when the parent record is displayed. At the present time, only fields within the Portal's TO are supported, but this limitation is expected to be resolved with a future release. The jqGrid has problems parsing complex joins between tables, and may fail when attempting multi-table joins. The ExtJS grid is intended to make it possible to avoid this limitation.

Downloading ExtJS

The ExtJS JavaScript framework is available from the Sencha website. ExtJS is dual licensed via open source GPL and commercial licensing, licensed on a per-developer basis. If you are developing a commercial website as a consulting, your customer also needs to purchase a license once you deliver the web application. Your customer does not have to buy support, only a license.
Note: ExtJS 4.11 is the minimum version supported by FmPro Migrator. Earlier versions of ExtJS may display the grid in the wrong location on the web page, or might not display any titles or data within the grid.

Once you have downloaded and unzipped ExtJS, store the folder in a convenient location on your hard drive, not the htdocs directory. FmPro Migrator will take care of copying ExtJS to your local htdocs or public_html directory. FmPro Migrator will only copy these files the first time it builds the web application directory. If the ExtJS grid was not selected as the type of grid to used the first time the web application directory was created by FmPro Migrator, you can:
1) Delete the web application directory, FmPro Migrator will re-build it and copy all of the necessary files.
2) Manually copy the ExtJS folder directory into your htdocs directory and rename it as: extjs.

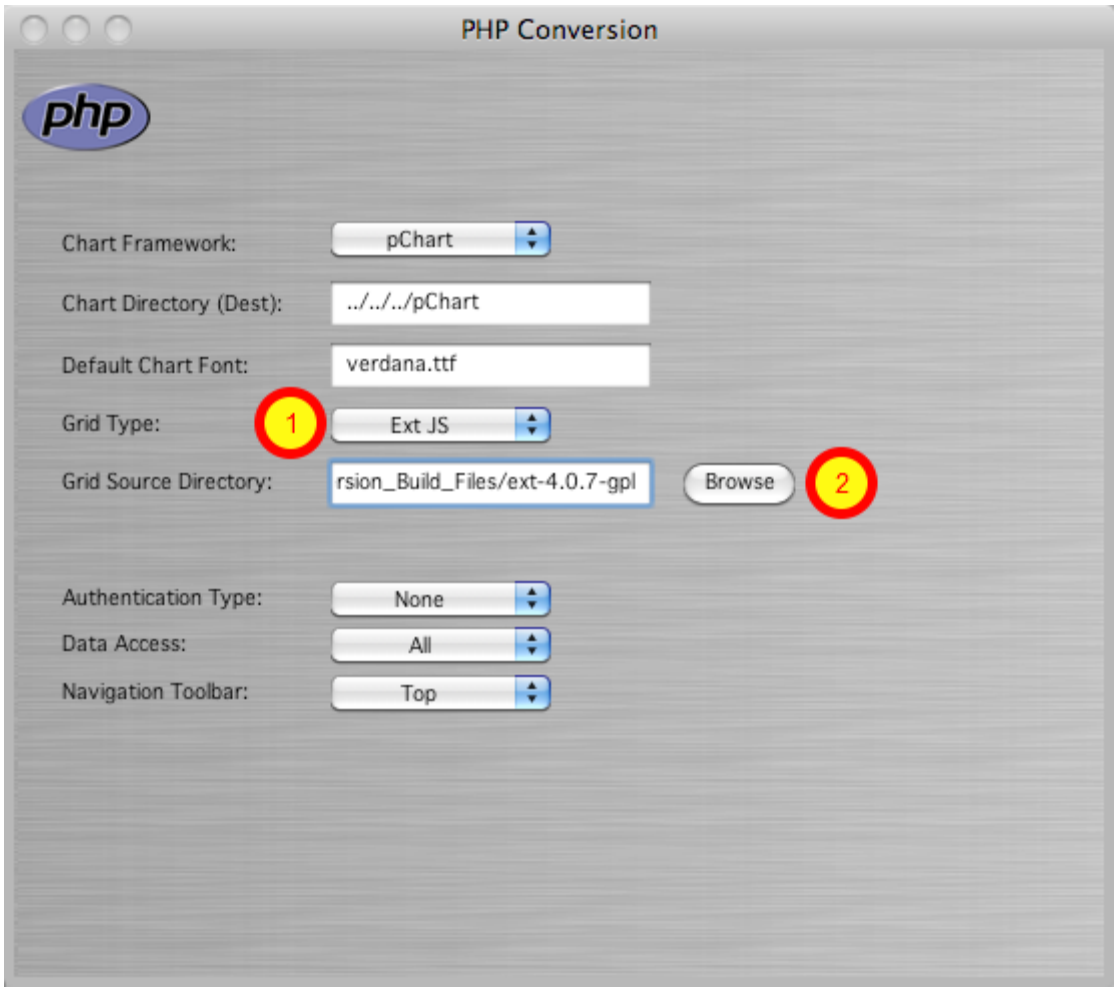
Pre-Conversion Portal Size Consideration

```
columns: [
                // ---------- Asset_ID ----------
                {header: "Asset_ID", dataIndex: 'asset_id', readOnly: true},
                // ---------- ID ----------
                {header: "ID", dataIndex: 'id', editor:{xtype:'textfield'}},
                // ---------- Date ----------
                {header: "Date", dataIndex: 'date_', xtype: 'datecolumn', width: 90, format: 'm/d/Y'
                        editor: {
                                xtype: 'datefield',
                                allowBlank: false,
                                format: 'm/d/Y',
                                minValue: '01/01/1970',
                                minText: 'Cannot have a start date before 1/1/1970!'
                                //maxValue: Ext.Date.format(new Date(), 'm/d/Y')
                                }
                },
                // ---------- Notes ----------
                {header: "Notes", dataIndex: 'notes', editor:{xtype:'textfield'}},
                // ---------- Maint_Picture ----------
                {header: "Maint_Picture", dataIndex: 'maint_picture', editor:{xtype:'textfield'}},
                // ---------- Maint_Condition1 - Menu: condition_list_value_list ----------
                { header: "Maint_Condition1", dataIndex: "maint_condition1", flex: 1,
```

Web based grid objects should not generally be created with a vertical size of less than 100 pixels/points. This is due to the requirement for the grid object to display a row of column titles as well as a row of control icons (Add/Delete icons). Therefore any FileMaker portals which are smaller than 100 pixels/points tall, should be increased in height for proper display in the generated web application. It is easier to make these types of changes within FileMaker than it is to modify the generated files, especially if there are other objects which need to be moved at the same time.

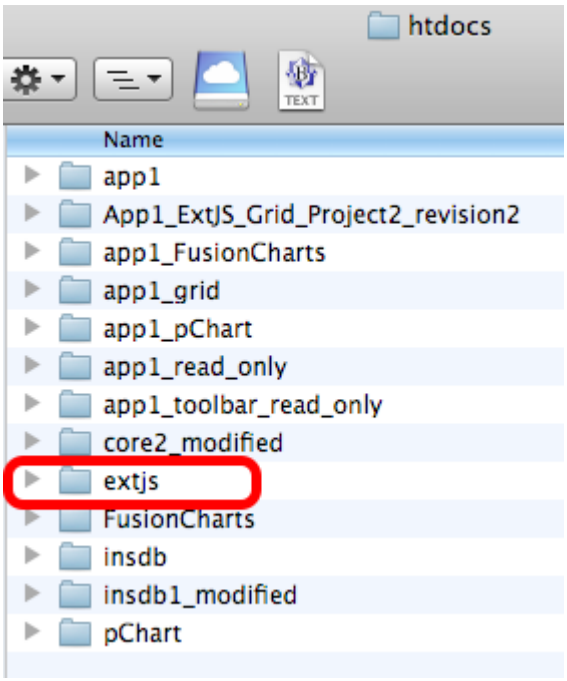
Since it is not practical for FmPro Migrator to automatically remove the unneeded text labels above portals, these should be removed manually before converting the layouts. By default, FmPro Migrator will use the original field name as the title for each grid row. The column header text can be easily edited within column definition section of the generated ExtJS grid files.

Selecting ExtJS in Preferences



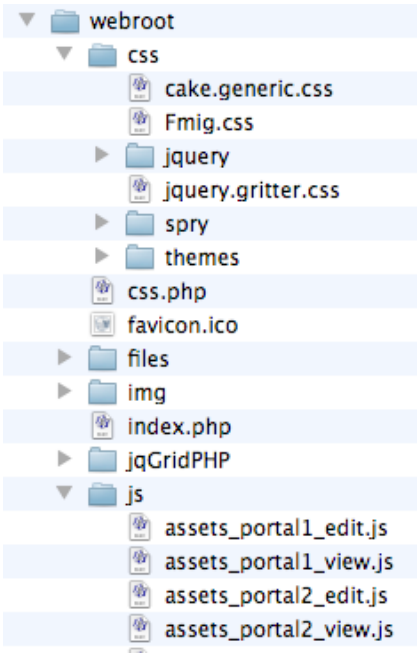
Click the preferences icon on the main PHP Conversion window. (1) Select Ext JS from the Grid Type menu. (2) Click the browse button then select the unzipped ExtJS directory on your hard drive. FmPro Migrator will automatically copy the ExtJS directory the first time you generate a project. The ExtJS folder will be named extjs at the top-level of your web server directory, so that it can be used by all of your applications. You will need to copy this extjs folder to the same relative location on your production web server.

ExtJS Web Folder Location



This screenshot shows an example of an htdocs directory having multiple CakePHP web applications, which are all sharing one copy of extjs, FusionCharts and pChart software.

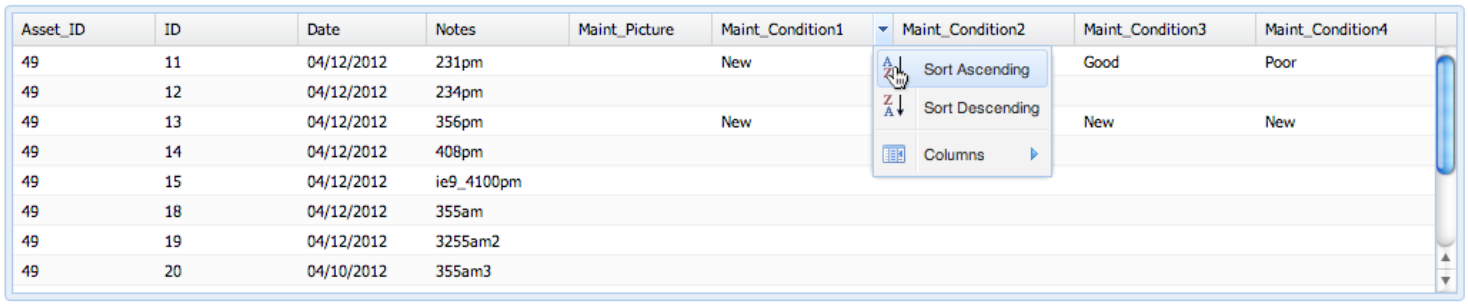
ExtJS Grid File Locations



The generated ExtJS grid JavaScript files are stored within the webroot/js folder. There are two .js files created for each portal on a layout, one file for the read-only view.ctp view and another file for the editable edit.ctp view file. These files are named using the format:

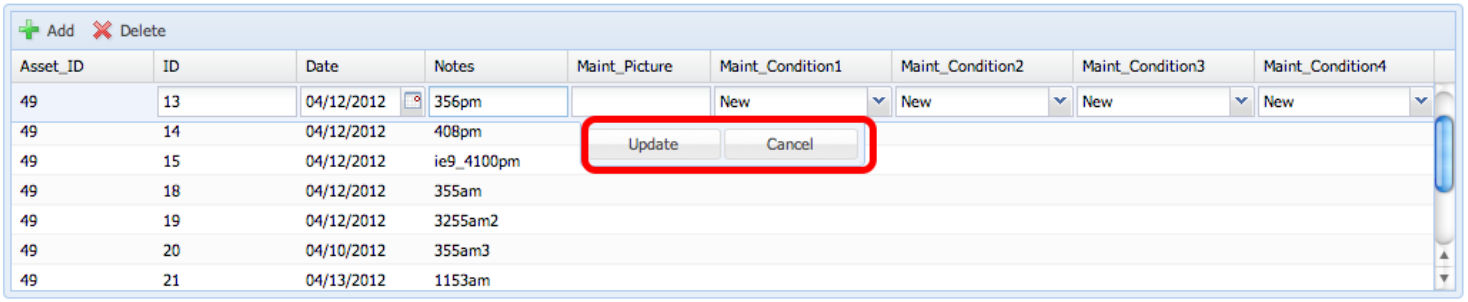
<layout name><portal name>_view.js
or
<layout name><portal name>_edit.js

ExtJS Grid - View Form Usage



Users can sort the rows of the grid by selecting the drop down menu within the title of any column of the grid.
Users may also reduce the number of columns in the grid by selecting the Columns item from the drop down menu, then unchecking any columns which they don't want to have displayed. This change is not persistent.

ExtJS Grid - Edit Usage



Grid objects generated for edit.ctp forms, include a row editing capability. To edit a row of the grid, double-click on the row. Edit any fields of the row, then click the Update or Cancel buttons.
Notice that the grid object contains a row of control icons above the column titles. The user many click on a row, then delete the row, or click on the "+" button to add a new row.

ExtJS Grid Limitations

1) At the present time, the PHP code which copies array data into JSON format for use by the ExtJS grid, only supports fields within the portal's relationship TO. A future release of FmPro Migrator will be required in order to remove this limitation.
2) Date fields must have a minimum date of 1/1/1970.
3) On touch devices, a two finger swipe is required in order to scroll thru related records in the

grid. Officially, Sencha states that ExtJS is not supported on touch devices.
4) In order to edit an existing row, it is necessary to double-click on the row in the web browser. It may not be possible to do this on a touch device. Selecting a row for deletion does work, and adding a row with the green "+" button also works on touch devices.
5) Container fields, checkbox, radio button, vector graphic objects, image objects and push button objects from the original FileMaker portal will not be converted into equivalent objects on either the ExtJS or jqGrid objects.
6) The Ext JS grid may be adversely affected by enabling the Google PageSpeed feature for your website. Having Google PageSpeed enabled may cause the Ext JS grid to display only for the first record, and then may disappear when navigating to other records in your application.

Using Save Records as PDF

The Save Records as PDF script step attached to a button will be directly converted into PHP code. The generated PHP code will use the html2ps package. Having a folder named html2ps located in the same folder selected as your ExtJS source directory, will enable FmPro Migrator to automatically copy this folder into the CakePHP /app/vendors folder during the initial project creation. The check for this source directory only occurs once, when the CakePHP application is first created in your htdocs directory.
Amodified version of this html2ps directory suitable for use with CakePHP has been uploaded to the PHP Conversion web page.

app_controller save_records_as_pdf function

```
function save_records_as_pdf($filename,$records_to_save,$download_type,$form_name,$model_name,$pkcolumn)
/*
* ----- create a PDF of the specified form ---
* @filename = output PDF filename, without .pdf extension
* @records_to_save = records to use for creating pdf (CurrentRecord,RecordsBeingBrowsed,BlankRecord)
* @download_type = prompt for file download or display inline within browser (download,file,browser)
* @form_name = name of view file to render into pdf (Contact/view, Contact/pdfview)
* @model_name = name of the model
* @pkcolumn = name of primary key column
*/
```

All web applications include the save_records_as_pdf function within the app_controller.php file. This function can be called from any part of the application. This function supports each of the FileMaker features: CurrentRecord, RecordsBeingBrowsed and BlankRecord. This function displays the specified form and converts it to Postscript for the creation of the PDF file. This process is memory and CPU intensive, and generally requires configuring at least 128Mb of RAM for PHP within your php.ini file. Generating multi-page PDF files may take 750Mb of RAM.

Advantages of Using html2ps for PDF Creation

```
function save_records_as_pdf_image28_png() {
// Create PDF Filename: 'Member - Current Record' Using: CurrentRecord
$this->save_records_as_pdf('Member - Current Record','CurrentRecord','download','members/view','Member','id');
}

function save_records_as_pdf_image28_png2() {
// Create PDF Filename: 'Members Foundset' Using: RecordsBeingBrowsed
$this->save_records_as_pdf('Members Foundset','RecordsBeingBrowsed','download','members/view','Member','id');
}

function save_records_as_pdf_image28_png3() {
// Create PDF Filename: 'Members Blank Form' Using: BlankRecord
$this->save_records_as_pdf('Members Blank Form','BlankRecord','download','members/query','Member','id');
}
```

The code shown here, is the form controller code which calls the app_controller save_records_as_pdf function.

1) The converted layouts of your FileMaker database are already converted into view files, which are immediately usable for generating PDF files. No additional development is required.
2) A developer can easily create a PDF suitable form by copying the existing view.ctp file, and manually removing unneeded objects like the Save as PDF button used to trigger the script. The PDF specific view file could be named pdfview.ctp, which can be specified in the calling function within the view controller.
3) By default, the PDF is downloaded to the web browser and opened or saved as a separate file. This parameter can be changed to "browser" to make the PDF display inline within the web browser window.
4) If you need to generate PDF files from within a script, instead of being trigger by a user clicking a button on a layout, just call the save_records_as_pdf() function from within your existing script.

Disadvantages of Using html2ps for PDF Creation

1) The HTML to Postscript conversion process requires noticeable time, memory and CPU resources. This process is not suitable for generating large quantities of PDF files. It generally takes about 5 seconds to generate each PDF page.
2) JavaScript objects like the raphael.js vector graphic objects are not rendered.
3) PNG images containing an Alpha channel will display with a black background.
4) PDF pages by default are rendered about 18% larger than the original web form. Though efforts to render at a different size have not yet been successful.
5) Setting the Landscape true/false configuration property has not yet been able to render the PDF in portrait mode.

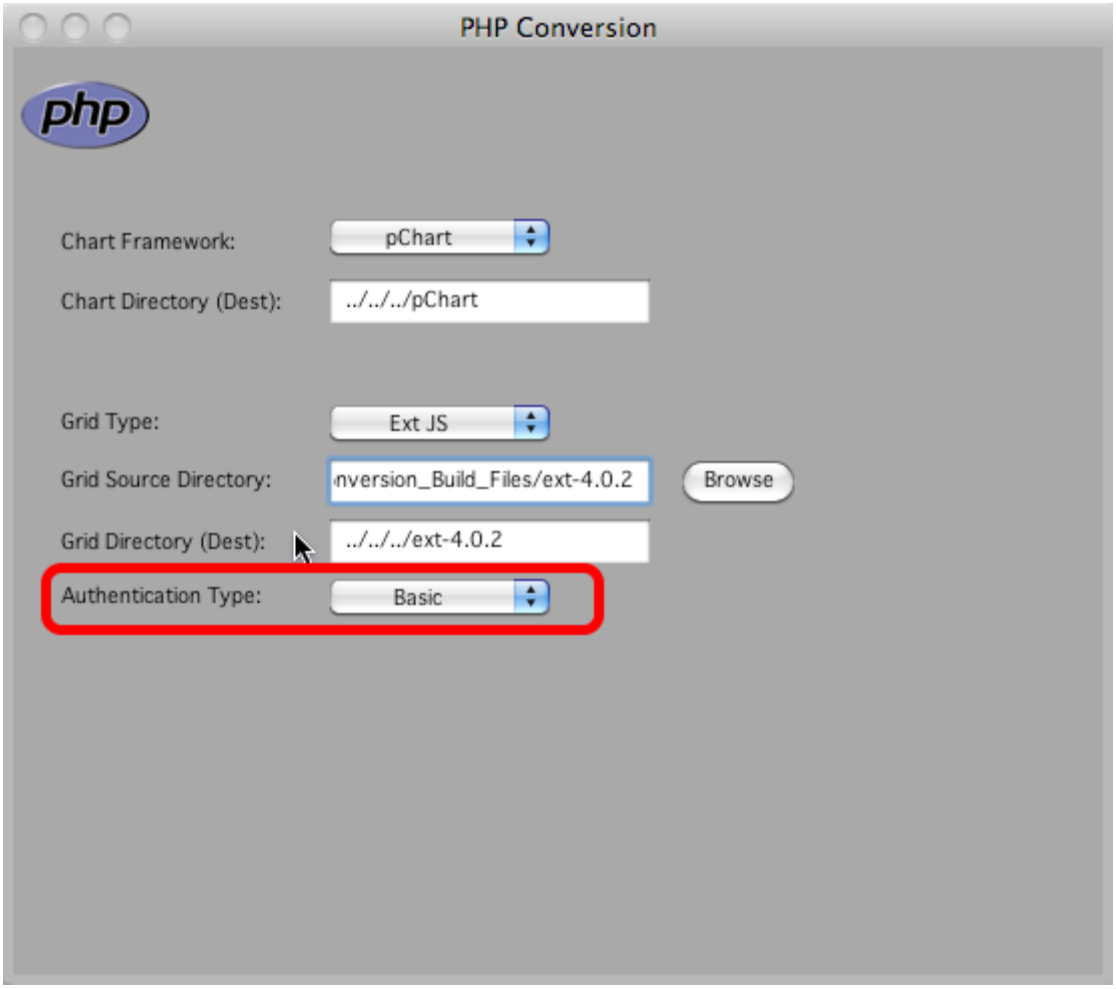
An Alternative to Using html2ps

A high performance PDF library named tcpdf is also available as an open source project. The tcpdf PHP class does require manually writing PHP code to create every object which needs to appear within the PDF file.

PHP Conversion - Manual Tasks

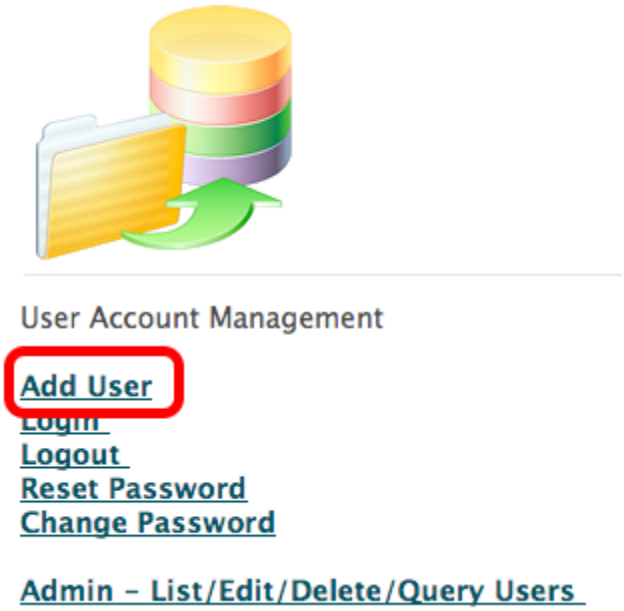
Selecting the Basic Authentication Type in the PHP Conversion preferences window, will generate login and role based security code within the PHP web application. This technique can be used to implement a straightforward, low overhead authentication system, without the complexity of the ACL authentication method.

Authentication Preferences Setting



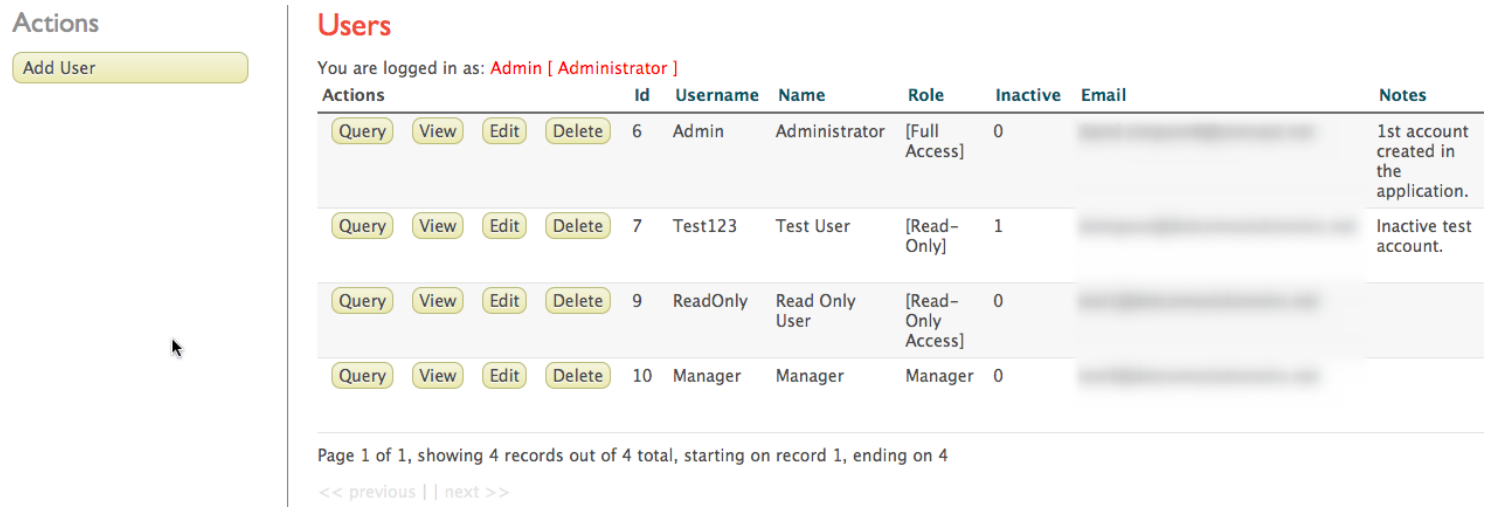
Selecting the Basic Authentication Type on the Preferences window, adds login and role based security settings to the generated PHP application.

Add User Link - Default Page



On the Default application page, click on the Add User link to add a new user account.
Note: The first user account created will automatically be granted the [Full Access] role within the web application.
To assign a role to any additional accounts, click on the Admin - List/Edit/Delete/Query Users link.

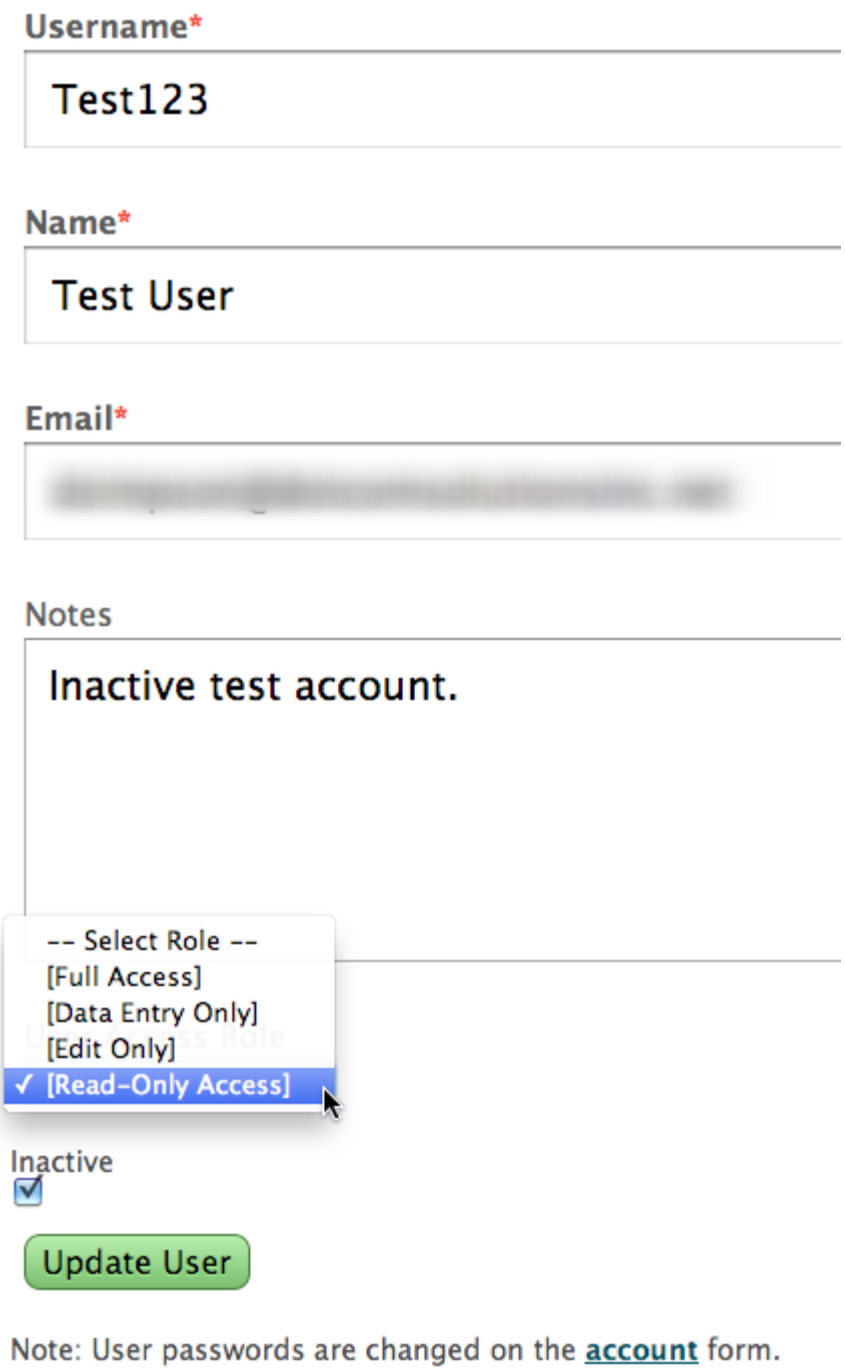
User Accounts List



Click the Admin - List/Edit/Delete/Query link to update information for any user account. You must be logged in with an account having the [Full Access] role in order to see this list of user accounts or make any changes to the accounts in the list.

Note: You cannot delete the account which is currently logged in.

Editing a User Account



Notes can be added to an account, the Role can be changed, and an account can be marked as Inactive, which prevents the user from logging in.

Changing passwords is reserved for the account action. Users do not have to have the [Full Access] role in order to be able to change their own password, but they do have to be logged in.

Authentication Code - app_controller.php

```
class AppController extends Controller {
    var $helpers = array('Html', 'Form', 'Paginator', 'Javascript', 'Session');
//  var $components = array('Session','Auth' => array(
//                                  'authorize' => 'controller',
//                                  'userScope' => array('User.inactive' => false),
//                                  'loginError' => 'Login Error: Invalid username or password',
//                                  'authError' => 'Your user account role does not have access to this feature'));
//
//      public function isAuthorized() {
//  switch (true) {
//      case ($this->Auth->user('role') == '[Read-Only Access]') && ( $this->action=='add' || $this->action=='edit' || $this->action=='delete
//          $this->Session->setFlash('Your user account role (' . $this->Auth->user('role') . ') does not have access to this feature (' . $t
//          return false;
//          break;
//      case ($this->Auth->user('role') == '[Edit Only]') && ( $this->action=='add' || $this->action=='delete'):
//          $this->Session->setFlash('Your user account role (' . $this->Auth->user('role') . ') does not have access to this feature (' . $t
//          return false;
//          break;
//      default:
//          return true;
//          break;
//      }
//  }
```

By default, the authentication code will be commented within the app_controller.php file. The code is generated this way in order to give the developer the opportunity to add login accounts to the users table. After you have added at least one user login account, uncomment this code in the AppController to enable role based security throughout your web application.

Note1: There will be another var $components statement a few lines below the AppController class declaration statement. It will look like this:
__public $components = array('Session');
And this line will also need commented when you uncomment the code shown here, because you cannot re-declare the same $components twice.

Note2: If you don't have any accounts with the [Read-Only] or [Edit-Only] roles, you may leave the two CASE statement blocks commented, which will default to a true returned value. Any tests you add to this SWITCH statement will be evaluated for every action used within the application.

Changing the Default Email Address in Recover Action

```
$token = $Token->generate(array('User' => $user['User']));
$this->Session->setFlash('An email has been sent to your account, please follow the instructions in this email.','flash_info');
$this->Email->to = $user['User']['email'];
$this->Email->subject = 'Password Recovery';
$this->Email->from = 'Support <support@example.com>';
$this->Email->template = 'recover';
$this->set('user', $user);
$this->set('token', $token);
$this->Email->send();
```

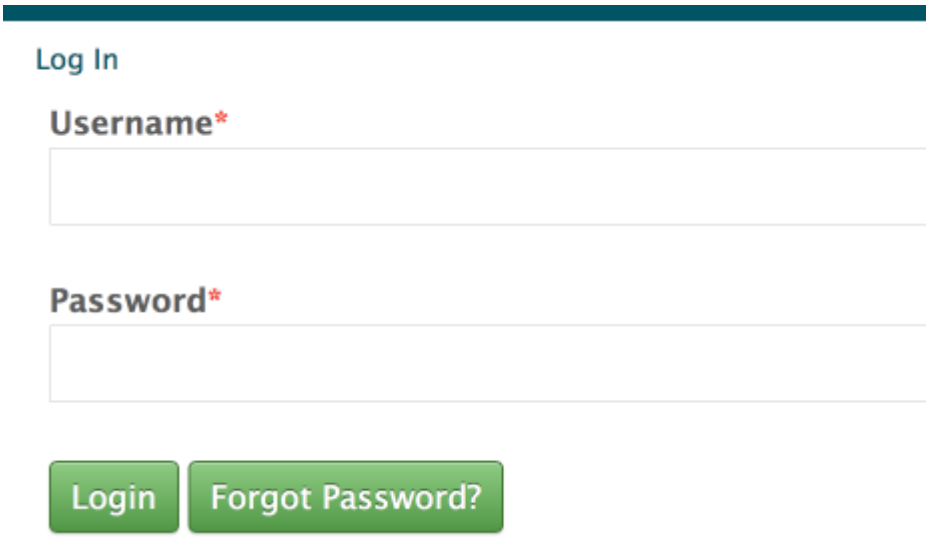
The email From: address should be changed within the recover() action of the users_controller.php file. The email template files within: app/views/elements/email/text directory may also need customization.

Adding Additional Layout Controller Authentication

```
function isAuthorized() {
    switch (true) {
        case ($this->Auth->user('role') == 'Manager'):
            $this->Session->setFlash('Your user account role (' .
            $this->Auth->user('role') . ') does not have access to this
            feature (' . $this->action .').','flash_error');
            return false;
            break;
    }
}
```

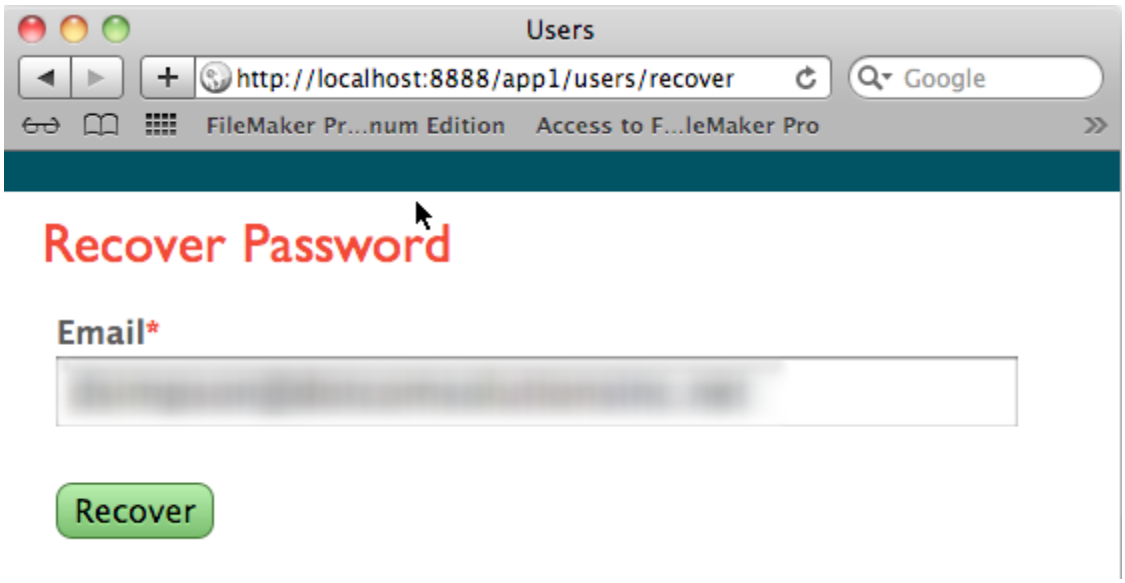
Within each converted layout controller file, you may add additional authentication tests. This example code prevents access to the converted layout forms for any accounts having the Manager role.
Additional CASE blocks may be added to prevent access to specific actions based upon the user's role. The SWITCH statement within the AppController provides an example of code which can be copied and modified.

Reset Password Feature - Step 1 - Click Forgot Password Button



The login screen includes a "Forgot Password?" button. Clicking this button takes the user to the Recover Password form.

Reset Password Feature - Step 2 - Enter Email

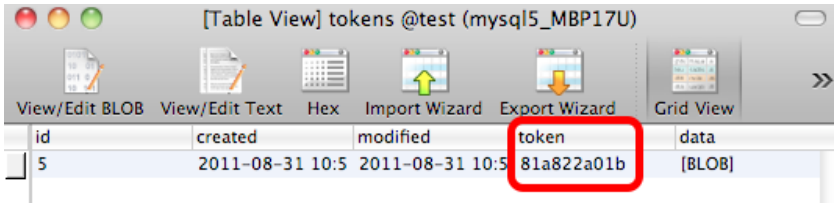


Clicking on the Rest Password link, sends an email to the currently logged in user's email account with instructions for how to reset the password.
A temporary password is entered into the tokens table, enabling the user to log in and reset their password. This process is necessary, because the password is never stored in cleartext anywhere in the database. Even an administrator does not know the cleartext version of the password. Only the encrypted password is stored in the database.

Note1: If a user is currently logged in, the user will be redirected to the account action to change their existing password. If the user is already logged in, they don't need to reset their account because they already know their password or they would not be logged in. This is why the "Forgot Password?" button is located on the login form.

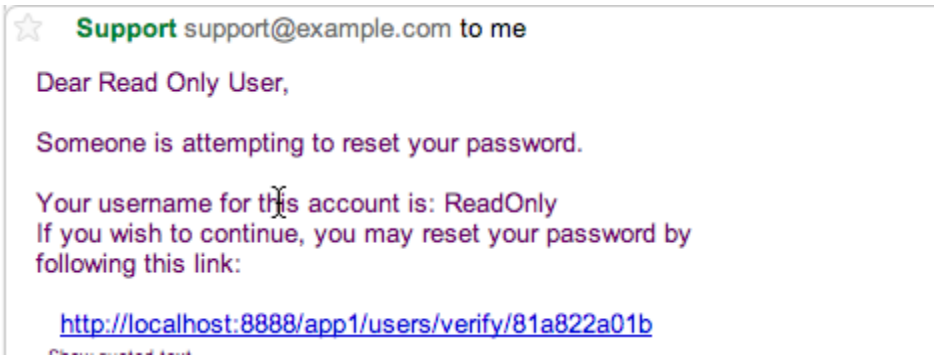
Step #2 - The user enters their email address into the recover view, then clicks on the Recover button.
A record will be entered into the tokens table of the database. This is why email addresses need to be unique among all of the user accounts, because the email address is the only way to identify the user.

Reset Password - Token Table Record



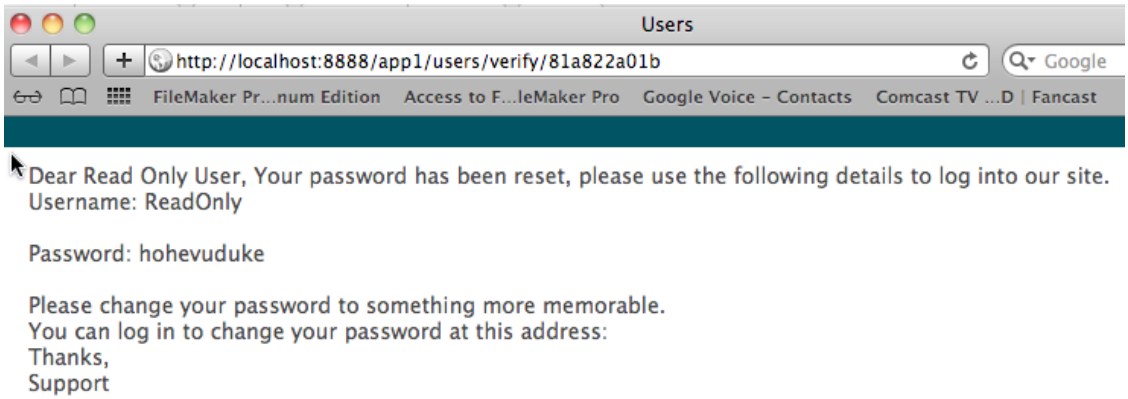
The token will be used for temporary login verification within the verify action.

Reset Password Feature - Step 3 - Click Verify Link



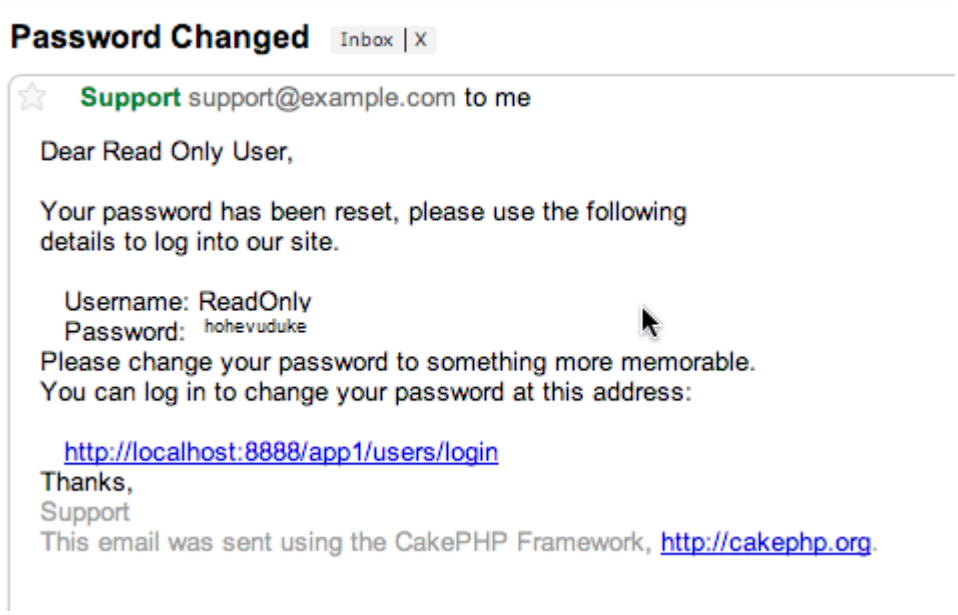
The user clicks the verify link, which passes the token to the verify action.

Reset Password Feature - Step 3 - Password Changed View



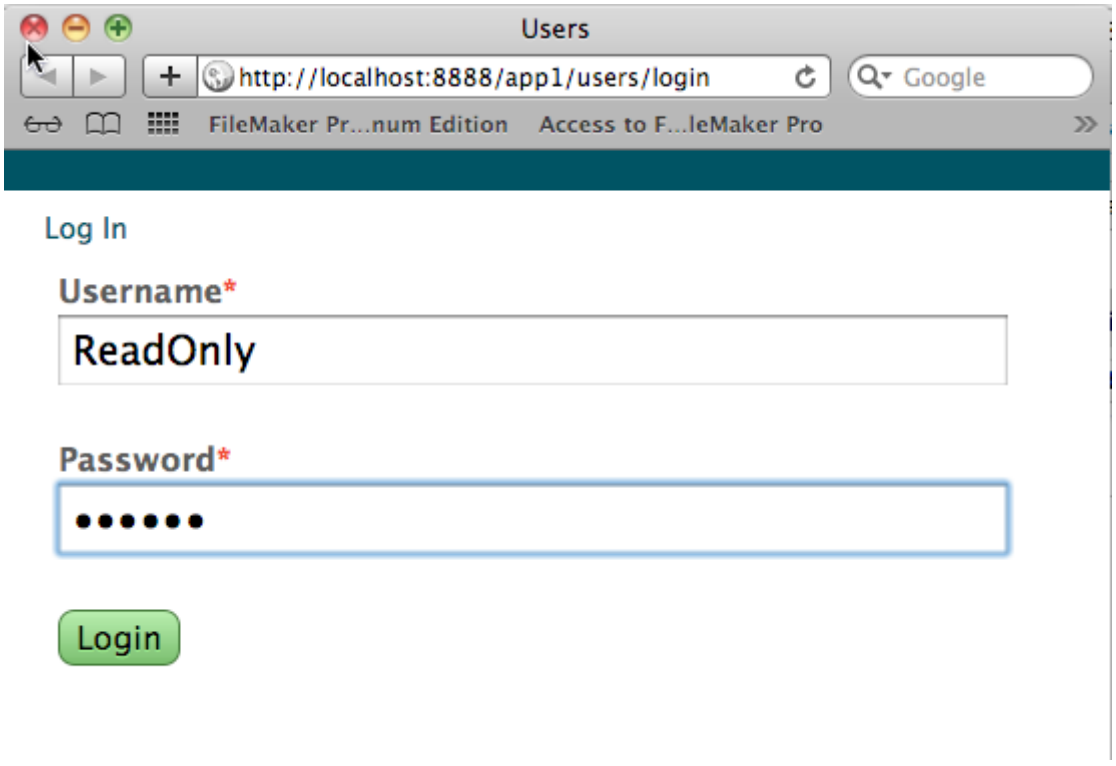
The user is also sent an email with the same temporary password info which is displayed on the verify view page.

Reset Password Feature - Step 4 - Click Email Link



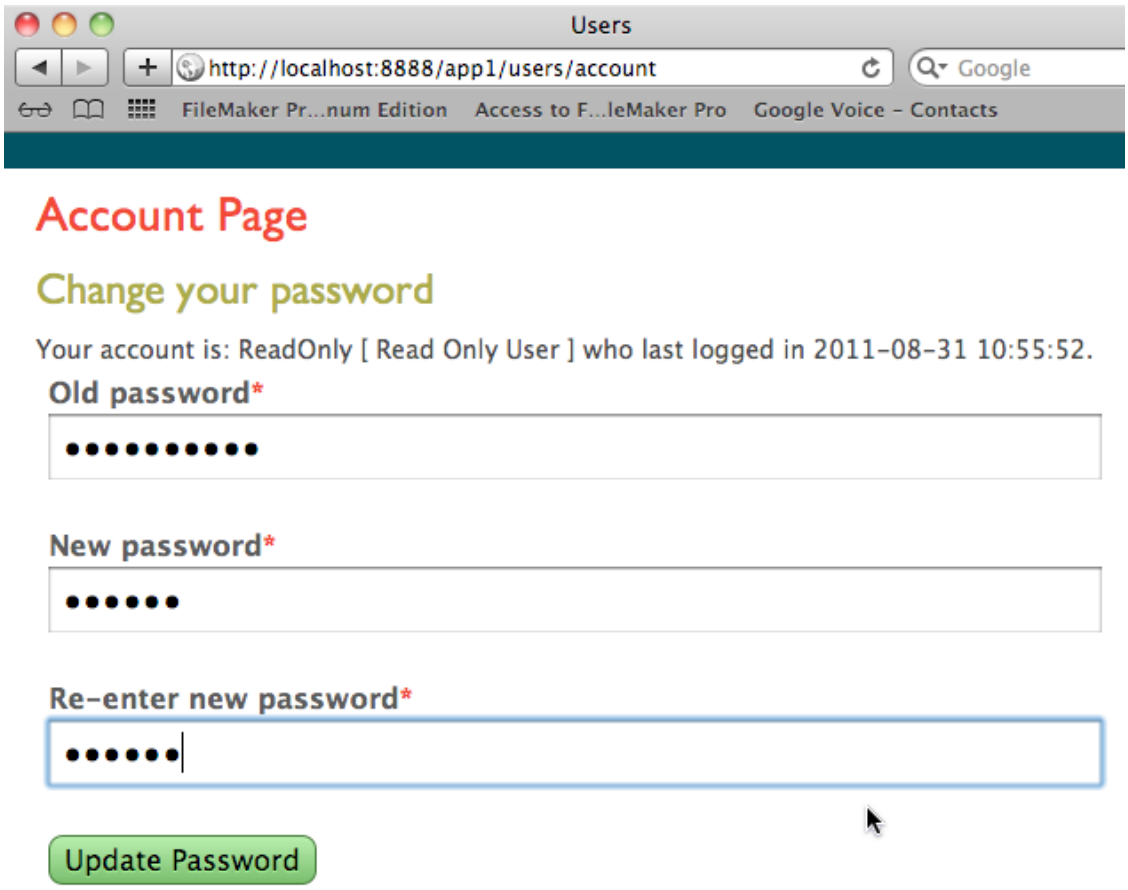
The user may also click on the login link within the email.

Reset Password - Step 5 - Login



The user needs to login using the temporary password.

Reset Password - Step 6 - Change Password



Once the user has logged in with their temporary password, they will then need to change the password.
On the account view, the user needs to **enter the temporary password again** within the Old password field. Then they need to enter their new password twice.
The user will be redirected to the Login screen afterwards, but they are already logged in so they don't really have to log in once again.

Login Redirect

Once a user logs in, they are automatically redirected to the page they were previously viewing or attempting to view.
In some cases, this behavior can potentially lead to confusion if the user changed their password, and will then be automatically redirected to the login view. Once they log in, they will be redirected back to the Change Password view.
This behavior can be changed in the login action code.

Summary of Built-in Roles

There are 4 built-in roles listed in the Roles menu of the admin_edit.ctp file. Any number of additional roles may be added to this menu.
The default roles have names matching built-in FileMaker Privilege Sets. These default roles are:

[Full Access] - This is the role which is required for administration of the User accounts. It is also the role required in order to view/edit records in the Base Table Controllers list shown on the default.ctp home page. The first user account created is automatically granted the [Full Access] role so that it can be used to administer the user accounts. No role is automatically assigned to any other account used by the web application. Roles for additional accounts must be manually assigned to the additional accounts.

[Edit Only] - Edit, View, Query, QueryList actions allowed, no access to Delete, Add actions. No administrative access.

[Read-Only Access] - View, Index, Query, QueryList actions allowed, no access to Edit, Delete, Add actions. No administrative access.

[Data Entry Only] - Index, View, Index, Edit, Delete, Add, Query, QueryList actions allowed. No administrative access.

Note: For each of these roles, any new actions added to the web application will automatically allowed unless the AppController or layout controller isAuthorized() function is manually edited.

Sharing Logins Between Applications

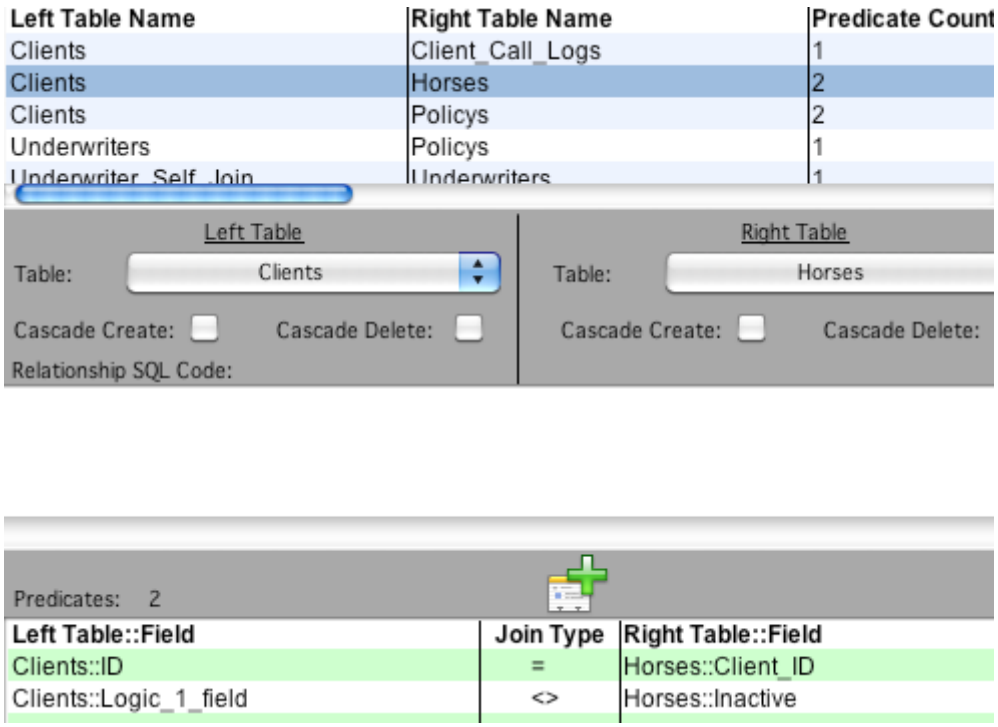
```
/**
 * A random string used in security hashing methods.
 */
    Configure::write('Security.salt', 'MTMxNDU3NzA2OTkzM1N1bmRheSwgQXVndXN0IDI4');

/**
 * A random numeric string (digits only) used to encrypt/decrypt strings.
 */
    Configure::write('Security.cipherSeed', '131457706993392020394895313 94');
```

If you need to share logins between multiple CakePHP projects, you will need to copy the app/config/core.php file to each application. The Security.cipherSeed and Security.salt values within the core.php file are used to encrypt the password text entered by users of the applications.

Manual Tasks - Model Files

There are a few manual changes which may be required to the generated model files in order to duplicate the functionality of the original FileMaker database. Manual changes to model files do not get overwritten when the project is re-generated.

A FileMaker Example Relationship



In this FileMaker database multi-predicate relationship, the first predicate defines the relationship between the records in the two tables, and this info is correctly converted into CakePHP hasMany/belongsTo array entries in the model files.

But the 2nd predicate requires manual modification in order to implement the same functionality. It isn't possible for FmPro Migrator to know exactly what values are contained within the Clients::Logic_1_field global field, because the values may change at any time. But the developer of the FileMaker Pro database knows that the Logic_1_field is really a static value containing the value 1.

Generated client.php Model hasMany Definition

```
'Horse'=>array(
'className'=>'Horse',
'foreignKey'=>'client_id',
'dependent'=>true,
'conditions'=>null,
'fields'=>null,
```

By default, the conditions property is set to null.

Client to Horse - Condition Array Changes

```
'Horse'=>array(
'className'=>'Horse',
'foreignKey'=>'client_id',
'conditions'=>array('Horse.Inactive <> 1'),
),
'InactiveHorseToClient'=>array(
'className'=>'InactiveHorseToClient',
'foreignKey'=>'client_id',
'dependent'=>false,
'conditions'=>array('Horse.Inactive = 1'),
'fields'=>null,
```

Adding the conditions array entry 'Horse.Inactive <> 1' duplicates the functionality of the original relationship. The InactiveHorseToClient hasMany entry has also been manually updated to reflect that only records where 'Horse.Inactive = 1' should be retrieved from the database.

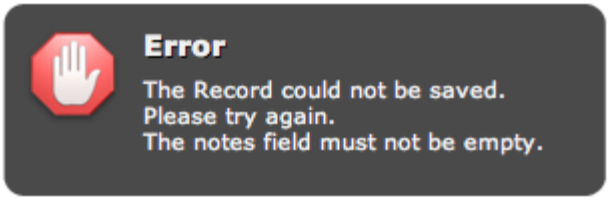
Some performance improvements can occur by reducing the number of table joins done in the database by using conditions instead of joins between tables. These changes could involve using conditions to replace the use of global field joins, and using globally defined variables, which would be set within controller code.

Validation Code - Calculation Formulas

```
'validation_test_max_chars5' => array(
                'rule' => array('maxLength',5),
                'message' => 'The field validation test_Max Chars5 must contain a maximum of 5 characters.'
                ),
//      'validation_test_calculation_formula' => array(
//                'rule' => 'ID = 1', // Table: tbl_assets
//                'message' => 'The field validation test_Calculation_Formula is validated by a Calculation Formula.'
//                ),
                'rule' => 'numeric',
                'message' => 'The value must be Numeric Only'
                ),
```

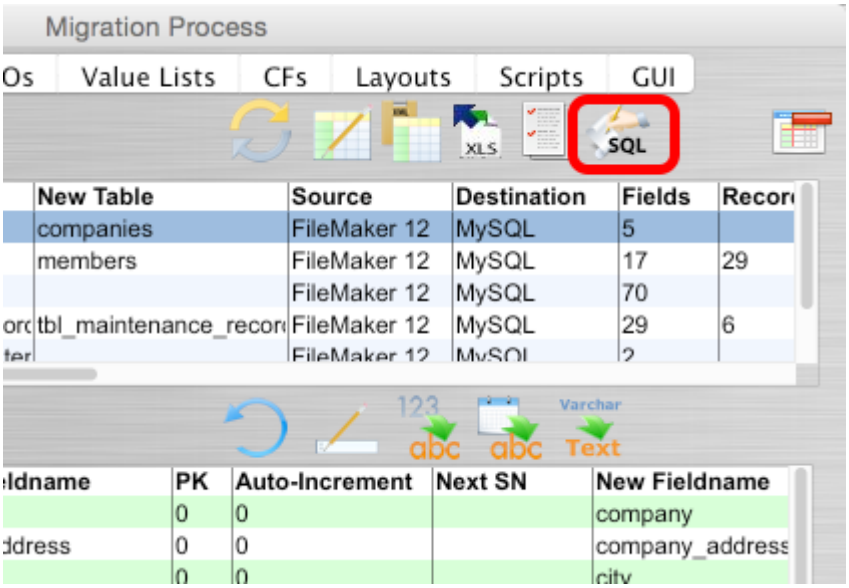
It is not practical to parse and convert FileMaker calculation formulas into PHP code. This is a task which needs to be done manually. Therefore validation tests involving calculations require manual changes to duplicate the same functionality. Calculation formula validations are commented by default in the generated model files.

Validation Text Messages



Validation message text is automatically used within the generated model files. However, some consideration needs to be made concerning whether the error message text provides enough explanation concerning which field is failing the validation test.

Summary and Calculated Fields



Summary and Calculated fields will need to be implemented manually within the model file in order to have the same functionally as the original database.

As an alternative to calculating Summary and Unstored Calculation values within the PHP web application, these values can be implemented with SQL view files in order to improve performance. FmPro Migrator includes a button above the list of tables on the Tables tab of the Migration Process window which generates SQL view files. These files include the original FileMaker calculation code along with a SQL version of the code suitable for modification by a SQL database DBA.

Using a SQL View in a Model file

```
<?php
class TblAsset extends AppModel {
    var $name = 'TblAsset';
    var $actsAs = array('Containable', 'CsvExport');
    var $useTable = 'tbl_assets_view';
    var $primaryKey = 'id';
```

```
  function beforeSave($options = array()) {
    $this->setSource('tbl_assets');
    // Save Global Fields Data to SESSION Memory
    return true;
  } // ----- End of beforeSave
```

We can modify 2 lines of code within any Model file to use the SQL view for reading data and to use the underlying database table for writing data.

Missing Relationships

FileMaker Pro has the ability to infer a relationship between two tables even if a direct relationship does not exist. FmPro Migrator cannot figure out these inferred relationships, therefore they need to be created manually within either the Model or Controller files.

Too Many Relationships

By default, all related tables are referenced in each Model file generated by FmPro Migrator. However, allowing CakePHP to calculate all of these relationships could be too time consuming. The result of this situation is that the application may timeout before completing the SQL database queries, or may display an database error about too many table joins.

These errors will occur on table form which uses the problematic Model files.

Simplifying Relationship Calculations

Therefore the solution to this problem is to simplify the relationships which need calculated. It will be necessary to only calculate related values when they are needed. These changes are generally made in 2 locations:

AppController.php
ToolbarsController.php
<Form>Controller.php

AppController.php Changes

```
function makequery($controller=null,$model=null,$isOmitOperator=null) {
    // use find request info to create query
    App::import('Model', $model);
    $Model = new $model;
    $this->autoRender = false;
    if(isset($_SESSION[$controller]['FindRequestData']))
    {
        $searchCondition = $this->MakeSearchConditions($controller,$model,$isOmitOperator);
        $conditions_of_search=$searchCondition['conditions'];
        if(isset($conditions_of_search))
        {
            unset($_SESSION[$model]['cond']);
            $_SESSION[$model]['cond']=$conditions_of_search;
            App::import('Controller', $controller);
            $controllerClass=$controller."Controller";
            $cntrl = new $controllerClass;
            $binds=$cntrl->binds;
            $Model->bindModel($binds);
            $resultset = $Model->find('all',array('recursive'=>1, 'conditions'=>$conditions_of_
            $_SESSION[$controller]['RecentFindRequest'][]=$_SESSION[$controller]['FindRequestD
            $_SESSION[$controller]['RecentFindNav'][]=$_SESSION[$controller]['FindRequest'];
            if (!empty($resultset)) {
```

1) The list of Models within the AppController.php file should be commented manually. Field based value lists, will automatically load only the Model they require.
2) Within the makequery() function, the follow lines of code need to be added in order to import the

models for performing searches upon the data. These additional lines of code go between the **$_SESSION[$model]['cond']=$conditions_of_search;** and **$resultset = $Model->find('all',array(**... lines of code.

```
App::import('Controller', $controller);
$controllerClass=$controller."Controller";
$cntrl = new $controllerClass;
$binds=$cntrl->binds;
$Model->bindModel($binds);
```

Toolbars Controller Changes

```
function AddFindRecord()
{
// add find request record
    $this->autoRender = false;
    $controllername = $_POST['controller'];
    App::import('Controller', $controllername);
    $controllerClass=$controllername."Controller";
    $cntrl = new $controllerClass;
    $modelNames=$cntrl->useModels;
    $model_name = $_POST['model'];
```

There are 3 changes required within the ToolbarsController.php file [if you are using the Navigation toolbar feature].

1) The first change, is the addition of the same model importing code within the AddFindRecord() function.

```
App::import('Controller', $controller);
$controllerClass=$controller."Controller";
$cntrl = new $controllerClass;
$binds=$cntrl->binds;
$Model->bindModel($binds);
```

2) Replace the following code within the AddFindRecord() function:

```
{
    $pos = strpos($key, $exams);
    if($pos === false)
    {
    }
    else
    {
    array_push($setarray,$key);
    }
}
```

with this new code:

```
        foreach($modelNames as $exams){
            $pos = strpos($key, $exams);
            if($pos === false)
            {
            }
            else
            {
                array_push($setarray,$key);
                break;
            }
        }
    }
```

3) Replace the following code within the AddFindRecord() function:

```
$vb = str_replace($toreplc,"",$k);
$last = str_replace("]","",$vb);
array_push($feindKey,$last);
array_push($findVal,$v);
```

with this new code [This is a new foreeach block within the existing foreach block:

```
foreach($modelNames as $exams){
    if(strpos($k,$exams) >0)
    {
        $toreplc = "data[".$exams."]"."[";
        $vb = str_replace($toreplc,"",$k);
        $last = str_replace("]","",$vb);
        $last=$exams.".".$last;
        array_push($feindKey,$last);
        array_push($findVal,$v);
        break;
    }
}
```

<Form>Controller.php Changes

```
<?php
class AssetsController extends AppController {
    var $name = 'Assets';
    var $uses = 'TblAsset';
    var $helpers = array('Html', 'Form', 'Paginator', 'Js');
    public $components = array('Security');
    public $useModels = array('TblAsset','TblMaintenanceRecord');
    public $binds = array(
                    'hasMany'=>array(
                    'TblMaintenanceRecord'=>array(
                    'className'=>'TblMaintenanceRecord',
                    'foreignKey'=>'asset_id',
                    'conditions'=>null,
                    'dependent'=>true,
                    'fields'=>null
            )
        );
```

Within the web application, it is possible that there could be dozens of relationships within a single Model file. However, there might only be a couple of relationships actually used on each individual form. Therefore it many be necessary to import only the required model files and relationships actually used on the form to improve performance.

Within the controller file for each form:
1) Add the $useModels and $binds arrays at the top of the controller file.
This code will look like the code in the Model files for hasMany/belongsTo relationship definitions. And example is shown below:

```
public $useModels = array('TblAsset','TblMaintenanceRecord');
public $binds = array(
         'hasMany'=>array(
        'TblMaintenanceRecord'=>array(
        'className'=>'TblMaintenanceRecord',
        'foreignKey'=>'asset_id',
        'conditions'=>null,
        'dependent'=>true,
        'fields'=>null
    )
);
```

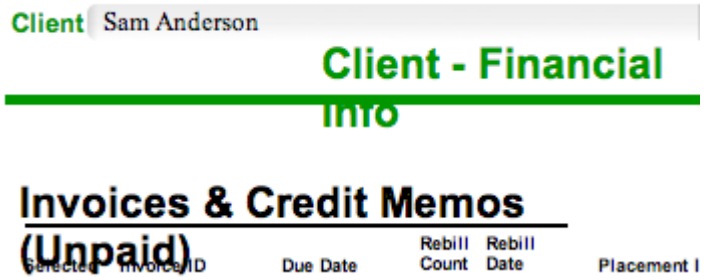
2) Add a call to bind the models:
$this->TblAsset->bindModel($this->binds);

This line of code needs to be added just before the $fields_list variable definition within the view(), and edit() actions of the controller.

3) Comment out the hasMany/belongsTo relationship definitions within each of the Model files.

Note: Make sure to include relationships used for Portals as well as regular fields. Looking at the $fields_list variable contents will show you all of the Models used for fields displayed on the form. Near the bottom of each controller file, you will find JSON data conversion code for each portal, which will reference the related Model.

Text Object Size



FmPro Migrator builds view files using the object coordinates from the original layout. In general, this process works very well. But sometimes, it may be necessary to change the size or position of the objects for use in web browsers.

Text labels will usually need to be lengthened in order to display properly in various web browsers. In this screenshot, the "Client - Financial Info" has wrapped down to the next line.

The "Invoices & Credit Memos (Unpaid) text also should be lengthened so that it does not word wrap.

If the text label is Left Aligned, then lengthen the label by moving the right side border longer by about 20%, for right aligned labels, adjust the left border.

These types of cosmetic changes should be made in the original FileMaker layout, then the layout should be copied back into FmPro Migrator and the project should be regenerated.

If multiple text styles, sizes, attributes are needed within the same text block, break up the text block into multiple individual text labels.

Object Layering

It isn't possible to export information about the front to back layering of the objects on the FileMaker layout. FmPro Migrator defines z-index CSS values for each object in order to attempt to resolve this issue. Objects such as vector graphics objects and images are put at the lowest level of the HTML canvas. Text labels, fields and portals (Grids) are placed above the graphics objects. Some fine tuning of this layering may be necessary.

Multi-Segment Image Objects

On some FileMaker layouts, multiple individual images may have been grouped together to form one single image. In most cases, these images will convert correctly into web forms. But sometimes there will be a small white hairline between the two images. If this situation occurs, then re-create the image as one single image, paste it onto the FileMaker layout, copy the layout into FmPro Migrator and re-generate the project files.

It is generally a good idea to quickly convert an entire project, and then carefully examine the generated web forms afterwards in order to determine if this problem will occur.

Adding FileMaker Layout Part Background Colors in CakePHP View Files

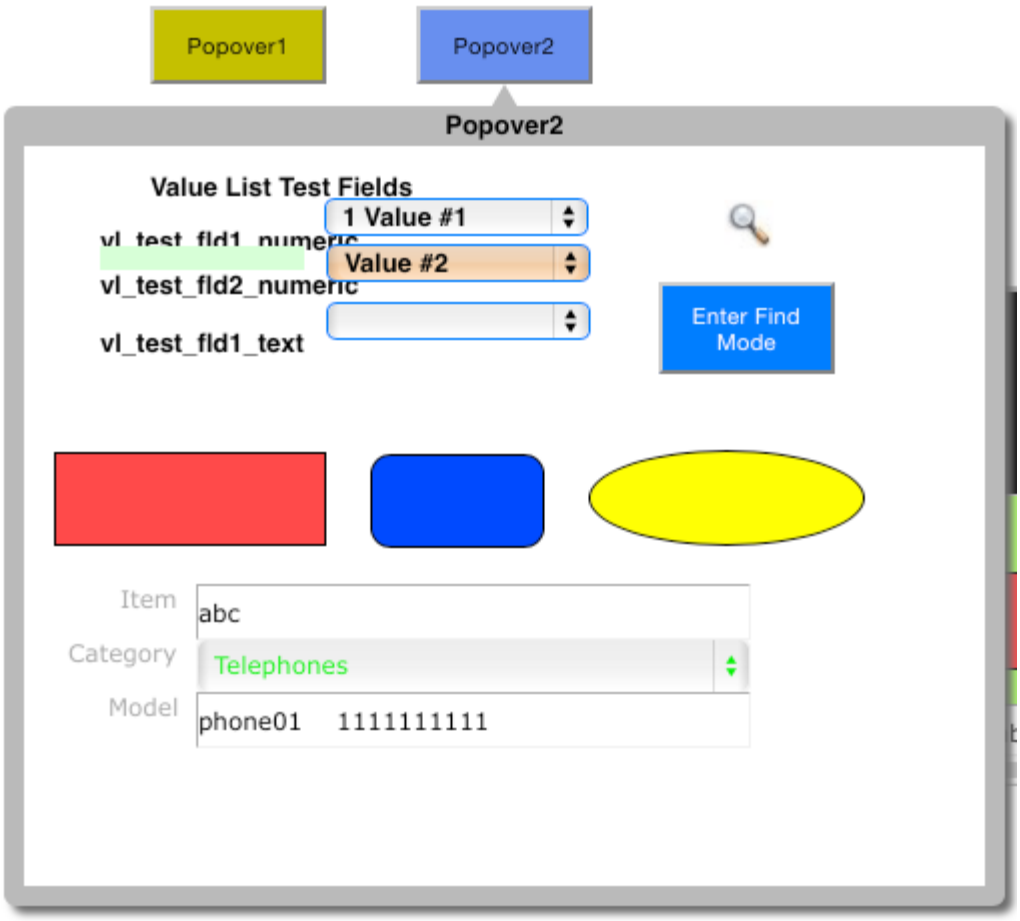
```
<style>
<!--
* {padding:0; margin:0;}
-->
</style>

<body bgcolor="#98CEEF">

<!-- View Contents Go Here -->
<?php echo $this->Form->end();?>

</body>
```

It is not possible to gather information about the layout part positions or background colors from the FileMaker layout information placed onto the clipboard. But the generated CakePHP view files can be modified to accomplish the same result.

1) Add a <body bgcolor > tag just below the </style> tag.
2) Add the closing </body> tag as the last line of the view file after the <?php echo $this->Form->end();?> PHP code.

Make this change to the view.ctp, edit.ctp, add.ctp, query.ctp files.

FileMaker 13 Popover Objects



AFileMaker 13 popover can include the complexity of an entire layout. Therefore the best way to handle this type of object when performing a conversion is to copy the popover contents into a new FileMaker layout. It is probably best to give the new layout a name referencing the original layout for documentation purposes: (example: Layout1_popover1)

Once the project has been converted, open the popover by adding button code to the popover triggering button on the original forms (view.ctp, edit.ctp, query.ctp, add.ctp files).

This code might look like the following examples:

```
var me = $('#button107_btn');
var title = $('#button107_btn').attr("title");

    me.showBalloon({
      position: 'bottom',
      tipSize: 21,
      minLifetime: 0,
      contents: 'Loading....',
```

```
        title: title,
        url: "<?php echo
'http://'.$_SERVER['SERVER_NAME'].":".$_SERVER['SERVER_PORT'].$this->webroot.'Popover2/pop

        css: {
          border: 'solid 10px #aaa',
          padding: '10px',
          fontSize: '100%',
          fontWeight: 'bold',
          lineHeight: '3',
          backgroundColor: '#fff',
          width: '800px',
          opacity: "1"
        }
      }).toggle(function() {me.hideBalloon(); }, function() { me.showBalloon(); });

      me.trigger('click');
```

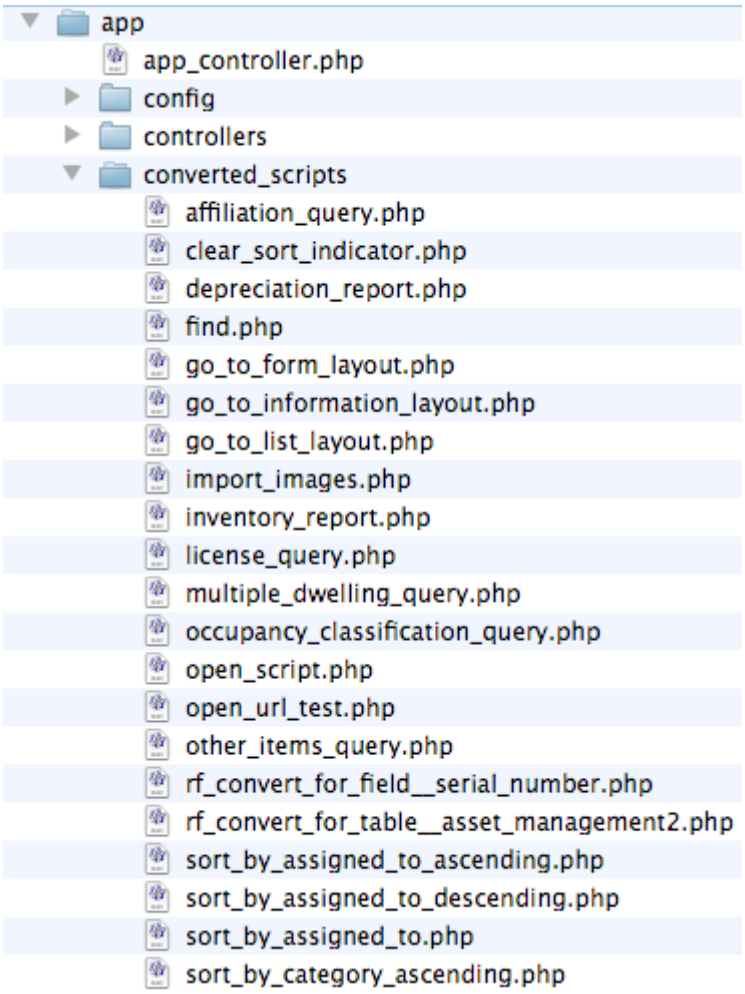
query Popover CSS

In addition to the button code shown above, there is also CSS code added to the view file to support the Popover.
Here is an example of that CSS code:

```
<?php echo $this->Html->css(array('smoothness/jquery-ui-1.10.0.custom.css'));?>

<style>
.ui-content .ui-dialog-titlebar
{
   display:none;
}
.ui-widget-content {
   border: 10px solid #C0C0C0;
}
.ui-dialog{
    padding:10px;
    color: #fff;
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    border-radius: 5px;
    width:190px;
```

```
    margin:30px;
}
.ui-dialog:before {
    content: ' ';
    height: 0;
    position: absolute;
    width: 0;
    left:590px;
    top:145px;
    border: 10px solid transparent;
    border-left-color: #C0C0C0;
}
</style>
```

Converted Scripts



FmPro Migrator converts FileMaker scripts into executable code and Custom Functions into commented PHP code. These files are written into a folder named converted_scripts at the top level of the application.

These scripts are intended as a starting point for further development of the PHP code. Within scripts which start with a Goto Layout script step, FmPro Migrator will use the specified TO Name assigned to the layout as the model name within the following statements. Otherwise, the text "<Missing Model>" will be written out to the generated script to indicate that FmPro Migrator cannot determine which Model to use within the PHP code.

Button Code

FmPro Migrator automatically converts Goto Record First, Previous, Next, Last script steps used for record navigation. There is controller code generated for the edit, view actions to set the $page[] array with the correct record primary key to use when redirecting the user after a navigation button is clicked.

The are a variety of additional layout button script steps which are automatically converted. Please see the section of this manual titled: "Layout Object Script Steps to JavaScript/PHP Conversion" for more details.

Rendering pChart Charts to Image Files

```
/* Render the picture to an image file using Render('../webroot/img/bar_chart_gallery_chart1.png') or send directly to browser using Stroke() */
$myPicture->Stroke('../webroot/img/bar_chart_gallery_chart1.png');
```

By default, FmPro Migrator renders pChart generated charts directly to the web browser. This behavior can be changed by changing the Stroke() function to Render() on the last line of the pChart controller code.

Using the Render() function will render the file to the top level of the webroot/img directory. Rendered chart PNG images can then be displayed by simply creating a static HTML page linking to the static image files. This technique can be used to create charts for high traffic web sites without requiring the user to wait for the chart to be created by the PHP code, thus improving the perceived performance of the web application and reducing the computational load on the web server.

The files don't have to be rendered into the webroot/img directory. The pathname could be changed to reference any directory which is accessible to the web server.

pChart FoundSet Usage

Within FileMaker Pro, the Y-Series database column for a chart will often be a Summary field. Since SQL database servers work differently from FileMaker Pro, the Summary & Grouping features are implemented within the PHP chart creation code. FmPro Migrator automatically retrieves the Tablename::Fieldname referenced within the Summary field and uses the source field as the Y-series field.

If the source of the Y-Series data was the Members::Company_Count column (a Summary field), then the column name would be changed to use the Members::Company column in the PHP code. Actually to be more accurate, the column name will use the CakePHP Model name followed by the column name (Member.company). The model name will be based upon the original TO Name on the RelationshipGraph.

The generated pChart scripts don't make use of the foundset Session array. So either the foundset record ID array could potentially be used, or a specialized query could be used in the find() statement to retrieves a specific set of records for chart processing.

pChart - Multi-Series Charts - If the Foundset with Group/Sort property is selected, the generated chart code will use:
X-Axis field - Will be used as the grouping field for the query.
Y-Axis field - If a Y-Axis field uses a Summary field which uses the X-Axis field as is source, the

count of values will be used for the data set.
Otherwise, the Y-Axis data values will be used for the found records. Asum of grouped values will be calculated during the query, but it is left available to the developer to use if needed.

If a Foundset is specified without the Group/Sort property or the Current Record field values is specified, the actual record values will be used for the data set created for the chart.

pChart Incorrect Data Values

Using incorrect data values for the Y-Axis will result in unusual looking (and incorrect) charts. FileMaker 11 will ignore text values for the Y-Axis, but pChart builds an unusual looking chart having negative bar values.

pChart Chart Differences Compared to FileMaker Charts

(pChart 2.1.1)
Legend areas cannot have a gradient blend, only a transparent background or solid background.
If the Legend border is not displayed, the legend background color is not displayed - the background is transparent.
To change font style (bold, italic, underline) - manually change the TrueType font filename in the generated PHP code.
Only a limited number of TrueType fonts are included with pChart, therefore a default font 'Verdana' is specified by default. This can be changed in the PHP Conversion Preferences.

pChart - Multi-Series Charts - X-axis titles don't get drawn with pChart 2.11. These titles appear to have worked previously, by reviewing the sample charts in the pChart documentation. But there are other issues with versions prior to 2.11, so reverting back to an earlier version is not recommended.

Also, Y-Axis text overwrites the Y-Axis values (possible bug?).
Note: There is no 3D Bar chart or Area Chart feature available in pChart 2.11 to match the FileMaker 3D chart implementation.

pChart Curve Fitting - Curve fitting is not supported by pChart but this feature might be supported in the future. The following forum post is from the developer:
12/10/2010 "As of today, pChart can only draw Bezier curves based on your datasets. The curve path has the constraint to go through each point without any interpolation."

Therefore FileMaker curve-fitting line charts are being drawn as line charts in pChart.

pChart Object Positioning

FmPro Migrator attempts to estimate the correct size and location required for pChart to render chart titles as well as the size and position of the chart. These values are estimated based upon the amount of text required for the label, as well as the whether titles and legends are drawn on the chart. It is likely that manual modification will be needed to reposition either the title text or the chart rendering area.

Fusion Charts - Foundset Usage

Within FileMaker Pro, the Y-Series database column for a chart will often be a Summary field. Since SQL database servers work differently from FileMaker Pro, the Summary & Grouping features are implemented within the PHP chart creation code. FmPro Migrator automatically retrieves the Tablename::Fieldname referenced within the Summary field and uses the source field as the Y-series field.

If the source of the Y-Series data was the Members::Company_Count column (a Summary field), then the column name would be changed to use the Members::Company column in the PHP code. Actually to be more accurate, the column name will use the CakePHP Model name followed by the column name (Member.company). The model name will be based upon the original TO Name on the RelationshipGraph.

The generated Fusion Chart scripts don't make use of the foundset Session array. So either the foundset record ID array could potentially be used, or a specialized query could be used in the find() statement which retrieves proper records for chart processing.

Fusion Charts - Multi-Series Charts - If the Foundset with Group/Sort property is selected, the generated chart code will use:
X-Axis field - Will be used as the grouping field for the query.
Y-Axis field - If a Y-Axis field uses a Summary field which uses the X-Axis field as is source, the count of values will be used for the data set.
Otherwise, the Y-Axis data values will be used for the found records. Asum of grouped values will be calculated during the query, but it is left available to the developer to use if needed.

If a Foundset is specified without the Group/Sort property or the Current Record field values is specified, the actual record values will be used for the data set created for the chart.

Fusion Charts - Differences Compared to FileMaker Charts

(Fusion Charts 3.2.1)
Legend areas cannot have a gradient blend, only a transparent background or solid background. The FmPro Migrator generated legend background code would create a gradient blend for the legend, if this feature was supported by Fusion Charts.

All JavaScript rendered 3D Pie charts (iPad), are rendered as 2D flat charts, and in some cases the charts will be drawn past the edge of the chart boundary. JavaScript Pie charts do not display legend info, but do display label/data info and gradients.

Data labels cannot have text Bold/Underline properties set via styles - but this is not an issue

because FileMaker doesn't allow this feature to be set anyhow.
Background radial gradients are not supported.
Fusion Charts requires Flash 8 or higher to be installed, otherwise the JavaScript fallback technique will be used automatically (implemented in Highcharts).
Flash 10 or higher is required in order for clients to export the chart image.
If you want to display lines from the Pie chart slices to the labels - change smartLineAlpha from 0 to 100 (or any value in between).

Setting drop shadow doesn't seem to work (bug?).
showShadow='1' use3DLighting='1'

Setting label/data value color doesn't work using Style definition (bug?).

Fusion Charts - Multi-Series Combi Charts - X-axis labels don't seem to have an angle property. All labels are drawn horizontally. But rotation of labels is available for other line/bar charts.

Fusion Charts - curve fitting is not supported by Fusion Charts, but is supported with the MSSpline.swf file included in the Power Charts package, an additional $400 product. Therefore FileMaker curve-fitting line charts are being drawn as line charts in Fusion Charts.

Dynamic Value List - Include Only Related Records

```
function define_item_list_value_list() {
    // Value List: item_list
    $TblAsset = new TblAsset;
    $item_list_value_list=$TblAsset->find('list',array("fields"=>array("TblAsset.item","TblAsset.item"),"conditions"=>array("TblAsset.item <>"=>null),"group"=>array("TblAsset.item"),"order"=>array("TblAsset.id ASC")));
    return $item_list_value_list;
    }
```

Dynamic value lists retrieve and display either 1 or 2 fields from the specified database table, depending upon the design of the original value list.
However, the "Include only related records" feature is not supported. Implementation of this feature could be done by adding an additional condition to the query. But as with FileMaker databases, if the specified table is not related to the current controller model, unexpected results could occur.

Dynamic Value List Sorting/Grouping

Values are displayed and entered into the web form field from either the first, second or both fields (just like FileMaker).
The resulting records are always grouped (duplicates removed) and an ascending sort is performed on the values.

Columns of Radio Buttons And Checkboxes

FileMaker displays Radio Buttons and Checkboxes in multiple columns on the layout if the field object is tall and wide enough. FmPro Migrator attempts to simulate this same visual behavior for Radio Button and Checkbox Groups taller than 20 pixels and wider than 94 pixels. Depending upon the average widths of the value list items, it may be necessary to manually change the number of columns which have been estimated by FmPro Migrator. This change can be made in the calls to the prepareInputGroup() function located within the form controller file for any form.

An example of this form controller code follows:

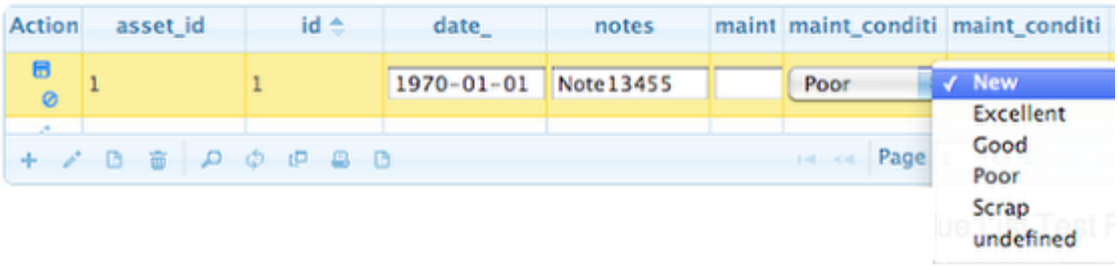
$this->set("tbl_assets__checkbox_test_fld110_group",$this->prepareInputGroup("checkbox","tbl_as
','2'));

The last parameter represents the number of columns to display, and in this example 2 columns will be displayed.

Tab Control Tab Label Text

The labels of tab controls will always be created using black text, using the default text font and size.
Note: It has not been possible to reliably retrieve this information from the layout XML style info for the tab panel.

jqGrid Edit Mode Menus



jqGrid objects created for edit.ctp view files include record editing capability. Part of this editing capability includes having menus available for fields which were configured with a menu on the FileMaker layout.
Since the jqGrid is implemented as a separate file of PHP code, these menus are not directly integrated with the CakePHP controller code.

jqGrid - Adding Value List Definition


```
63  $condition_list_value_list_grid = "";
64  $condition_list_value_list = "";
65  if (!empty($condition_list_value_list)) {
66              foreach($condition_list_value_list as $key=>$value) {
67              $condition_list_value_list_grid = $condition_list_value_list_grid.$key.':'.$value.';';
68              }
69          }
70  $grid->setColProperty('maint_condition1', array("edittype"=>"select", "editoptions"=>array("value"=>" $condition_list_value_list_grid"),"label"=>"maint_condition1","width"=>67));
71  $condition_list_value_list_grid = "";
72  $condition_list_value_list = array("New"=>"New","Excellent"=>"Excellent","Good"=>"Good","Poor"=>"Poor","Scrap"=>"Scrap");
73  if (!empty($condition_list_value_list)) {
74              foreach($condition_list_value_list as $key=>$value) {
75              $condition_list_value_list_grid = $condition_list_value_list_grid.$key.':'.$value.';';
76              }
77          }
78  $grid->setColProperty('maint_condition2', array("edittype"=>"select", "editoptions"=>array("value"=>" $condition_list_value_list_grid"),"label"=>"maint_condition2","width"=>67));
```

It is easy to manually copy static value list definition code from the app_controller.php file into the generated grid PHP file.
Line # 64 shows the unmodified line of code, with an empty value list definition.
Line #72 shows the value list definition array, which has been copied from the app_controller.php file.

Dynamic value lists will require more work, because the jqGrid does not make use of the models defined within the CakePHP application.

jqGrid - Minimum Vertical Size

Web based Grid objects require more vertical space compared to FileMaker portals. Therefore, if a FileMaker portal is displayed too small, the web based grid (created by jqGrid), will not always be usable. The minimum usable size for a grid is generally about 100 pixels tall, due to the header row at the top of the grid and the footer row of controls at the bottom of the grid. Even at

this size, the grid may only display about 1 row of related row data.

jqGrid - List of Grid Fields

It may not always be possible for FmPro Migrator to figure out exactly which fields are contained within the portal, especially if the fields were actually created before the portal object. Therefore some converted portals may require manual code changes within the generated php code.

jqGrid - Checkbox Sets, Radio Button Sets, Image Fields

The jqGrid does not support radio buttons, checkboxes, multi-line portal rows, images, buttons objects, or vector graphic objects drawn within the rows of the portal. Therefore radio button and checkbox fields will be created as regular text fields. Other objects will be omitted.

jqGrid - Multi-Table Joins

SQL code for multi-table joins needs to be completed manually within the jqGrid portal files. The names of the fields and tables will be generated automatically, but the joins required in the WHERE clause will need to be written manually. Since the jqGrid cannot be directly integrated within the CakePHP controller code, the automatic table joins done within the CakePHP models cannot be used for gathering records for the jqGrid.

jqGrid - Column Labels

There is no practical way to automatically transfer the text labels above a FileMaker portal into the grid object. These column labels will need to be manually entered into the jqGrid .php file. By default, FmPro Migrator uses the column name as the label for each column of the grid.

jqGrid - Supported Database Servers

The following database servers are supported for Read/Write usage with the jqGrid object. The jqGrid makes a direct connection to the following database servers in order to provide Read/Write access to the records in the grid.

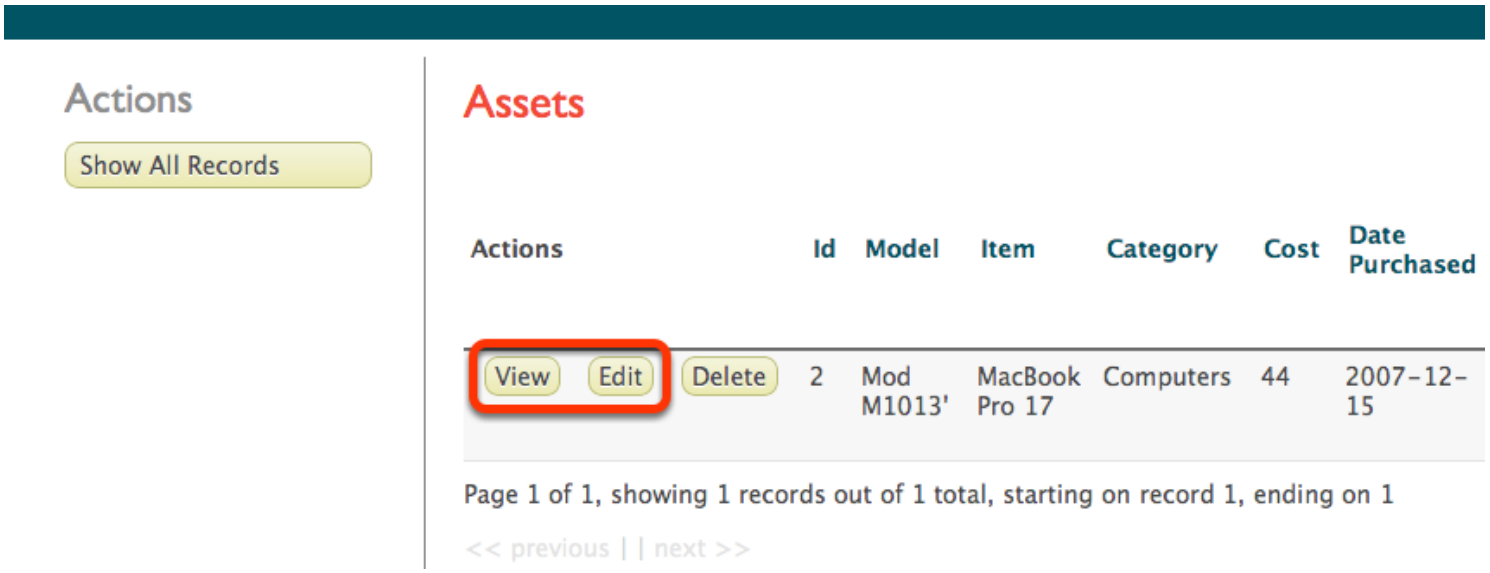
MySQL 5.x+ (PDO)
PostgreSQL versions 7.x+ (PDO)
SQLite versions 3.x+ (PDO)
Microsoft SQL Server 2005+
Oracle 8+
MySQL (through Mysqli)
DB2 (IBM DB2 functions)
MongoDB

ODBC connectivity to FileMaker or any other ODBC data source is not supported because the jqGrid cannot have enough control over the ODBC database connection to work with large data sets.

jqGrid - Date Picker Columns

It is possible to set a jquery DataPicker for jqGrid columns in edit mode. This code can be configured manually within the jqGrid setup code.

jqGrid - Foundset Mode



After doing a query, the web application will be running in Foundset mode, with a list of the primary key values passed thru the foundset Session variable. As long as the foundset Session variable is present, the user will be re-directed back to the querylist.ctp view file.

Global fields, Stored and Unstored Calc fields will require manual changes after the automated conversion process.

Global Fields - Defined Within App Controller

```
function beforeFilter() {
    if ($this->Session->check('GlobalFields')==false) {
    // Initialize global fields for the current session - only if not already set
    $this->Session->write('GlobalFields.tbl_assets.template_information_global', '');
    $this->Session->write('GlobalFields.tbl_assets.hilitelibrary', '');
    $this->Session->write('GlobalFields.tbl_assets.hilitesortedby', '');
    $this->Session->write('GlobalFields.tbl_assets.sc_url_prefix', 'http://10.1.0.101:8020';
    }
}
```

Global fields are defined within the app_controller.php file within the beforeFilter() function. These values are initialized (if necessary) and stored within SESSION variables for each user. This means that global fields which are changed by scripts will have their values persist for the user of the web application. The initial value for each of these global fields should be manually updated within the beforeFilter() before the application is run the first time. Otherwise they will get initialized to an empty value and not reset afterwards. If this occurs, just comment out the if statement and its enclosing brace, let the app run once, then uncomment these lines.

Global Fields - Defined Within Model Files

```
function beforeSave($options) {
    // save global fields data to SESSION memory
    $this->Session->write('GlobalFields.tbl_assets.template_information_global', $this->data['TblAsset']['template_information_global']);
    $this->Session->write('GlobalFields.tbl_assets.hilitelibrary', $this->data['TblAsset']['hilitelibrary']);
    $this->Session->write('GlobalFields.tbl_assets.hilitesortedby', $this->data['TblAsset']['hilitesortedby']);
    $this->Session->write('GlobalFields.tbl_assets.sc_url_prefix', $this->data['TblAsset']['sc_url_prefix']);
    // calculate stored calculation fields before saving
    //$this->data['TblAsset']['book_value'] = If(GetAsNumber(Depreciation Life) <> 0; Round( Remaining Life / Depreciation Life * Cost ; 2 ); 0);
    //$this->data['TblAsset']['remaining_life'] = If((Date Purchased + Depreciation Life * 365 - Int(Today)) / 365 > 0; (Date Purchased + Depreci
    $this->data['TblAsset']['depreciation'] = $this->data['TblAsset']['cost']-$this->data['TblAsset']['book_value'];
    //$this->data['TblAsset']['date_due'] = If(IsEmpty(Check Out Days)=1 or IsEmpty(Date Check Out)=1; GetAsDate(""); Date Check Out + Check Out
    //$this->data['TblAsset']['overdue'] = If(IsEmpty(Check Out Days)=1 or IsEmpty(Date Check Out)=1 or IsEmpty(Date Check In)=0; "";  If(GetAsNu
    //$this->data['TblAsset']['assigned_display'] = If( Length(Assigned To) > 20;  Left(LeftWords(Assigned To; 1); 20) & "...";  Assigned To);
    $this->data['TblAsset']['sample_calc'] = "100 years old";
    return true;
}
```

```
function afterFind($results) {
    $base = null;
    $this->Session=new CakeSession($base);
    // replace global fields data with SESSION memory
    foreach ($results as $key => $val) {
      $results[$key]['TblAsset']['template_information_global'] = $this->Session->read('GlobalFields.tbl_assets.template_information_global');
      $results[$key]['TblAsset']['hilitelibrary'] = $this->Session->read('GlobalFields.tbl_assets.hilitelibrary');
      $results[$key]['TblAsset']['hilitesortedby'] = $this->Session->read('GlobalFields.tbl_assets.hilitesortedby');
      $results[$key]['TblAsset']['sc_url_prefix'] = $this->Session->read('GlobalFields.tbl_assets.sc_url_prefix');
    }
    return $results;
}
```

Global fields are incorporated within the web application in a manner which makes the data appear to come from the database. Globals can easily be used within the PHP web application by simply reading from or writing into the model. Therefore using a global field on a layout will result in it being usable in the same manner within the web application. To implement this functionality, the contents of global fields are automatically saved back into SESSION memory by the beforeSave() function whenever a record is added or updated.

Also, the afterFind() function replaces any globals read from the database table with the latest current data stored within SESSION memory for the current user.

Stored Calculation Fields - Defined Within Model Files

```
// calculate stored calculation fields before saving
//$this->data['TblAsset']['book_value'] = If(GetAsNumber(Depreciation Life) <> 0; Round( Remaining Life / Depreciation Life * Cost ; 2 ); 0);
//$this->data['TblAsset']['remaining_life'] = If((Date Purchased + Depreciation Life * 365 - Int(Today)) / 365 > 0; (Date Purchased + Depreciation Life * 365 - Int(Today)) / 365; 0);
$this->data['TblAsset']['depreciation'] = $this->data['TblAsset']['cost']-$this->data['TblAsset']['book_value'];
```

Stored Calculation fields get calculated before a record is saved into the database within the beforeSave() model function.
Simple arithmetic or concatenation calculations are converted automatically into PHP code. Calculations containing function calls (or even a single open parenthesis character) are always commented out, requiring manual conversion by the PHP developer.

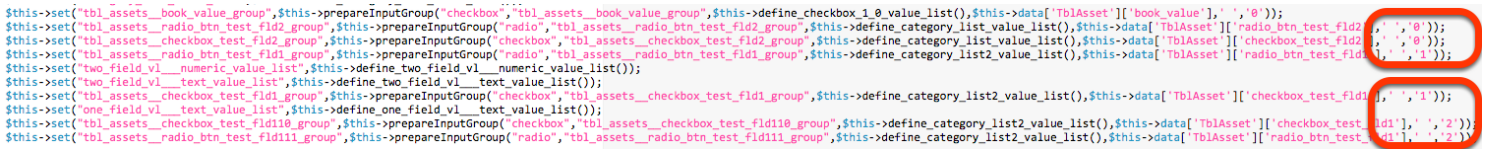
Unstored Calculation Fields - Defined Within View Controller Files

```
// unstored calculation fields
$this->data['TblAsset']['sc_full_url_readwrite'] = $this->data['TblAsset']['sc_url_prefix']."/SuperContainer/Files/App1/".$this->data['TblAsset']['id']."/Notes1Data?style=showdelete+upload+info+bold+left";
$this->data['TblAsset']['sc_full_url_readonly'] = $this->data['TblAsset']['sc_url_prefix']."/SuperContainer/Files/App1/".$this->data['TblAsset']['id']."/Notes1Data?style=info+bold+left";
$this->data['TblAsset']['sc_full_url_readwrite'] = $this->data['TblAsset']['sc_url_prefix']."/SuperContainer/Files/App1/".$this->data['TblAsset']['id']."/Notes1Data?style=showdelete+upload+info+bold+left";
```

Unstored Calculation fields are calculated before the value is used within the view controller file, within the view and edit actions. Simple arithmetic or concatenation calculations are converted automatically into PHP code. Calculations containing function calls (or even a single open parenthesis character) are always commented out, requiring manual conversion by the PHP developer.

This example code shows the conversion of a SuperContainer URL which concatenates a global field, the primary key column field and static text defining the SuperContainer options.

Vertical vs Horizontal Checkbox and Radio Button Sets



Radio Buttons and Checkboxes which are taller than 20 pixels, will be assumed to be vertically oriented objects - and will created with a "
" HTML tag separating each value. This feature can be changed within the controller code for of the view.

The same code also offers the option for an additional column which defines the number of columns to display.

Layout Names - Spaces and Special Characters

CakePHP does not allow controllers having more than one underscore character between letters of the controller name. FmPro Migrator automatically removes extra spaces, underscore or special characters during the conversion process. This change will result in controller names which look a little different from the original layout names.

Object Rotation on Layouts

Web browsers don't reliably support object rotation properties, therefore this feature is not implemented, by design. In fact, none of the major web browsers currently supports the CSS3 rotation properly. If an image object is rotated incorrectly in the generated web form, it may be possible to change the contents of the image object within the app/webroot/img directory so that it will be automatically changed within all views of the generated project. This workaround will not solve the problem of an image object used in multiple places, with different rotation properties.

Otherwise, rotated image objects should be created as graphic objects place directly onto the layout.

Merge Fields

```
<p id="button3_btn" style="position: absolute; top: 335px; left: 402px; width:
189px; height: 27px; text-align: left; vertical-align: top; color: ;
border-width: 0 px; border-color:; z-index: 3; font: -2pt ; background-color:
#F8F8F8"><?php echo htmlentities("Loaned To ", ENT_QUOTES,
'UTF-8').$this->data['TblAsset']['assigned_display']; ?></p>
```

Merge fields are embedded within text labels within the original layout. When a merge field is found within a text label, the field definition is added within the text of the text label. Merge fields embedded within blocks of text including Unicode characters will require manual changes to the

text and field output to wrap the text within an htmlentities() function for proper display of unicode characters.

Layout Symbol Objects

```
<!-- Text: text59_lbl -->
<p id="text59_lbl" style="position: absolute; top: 145px; left: 907px; width:
22px; height: 14px; text-align: left; vertical-align: top; color: #000000;
border-width: 0 px; border-color:; z-index: 3; font: 10pt Helvetica;
background-color: none"><?php echo $this->Paginator->current(); ?></p>
```

The following layout symbol objects are supported (for FileMaker 12 and earlier versions):
{{CurrentDate}} or // - Current date, in mm/dd/yyyy format, displayed based upon the locale of the server where the PHP code is executed.
{{CurrentTime}} or :: - Current time, in 24hr HH:MM:SS format, displayed based upon the locale of the server where the PHP code is executed.
{{UserName}} or || - Current logged in username. Note: An error will be generated if the isAuthorized() function has not been uncommented within the app_controller.php file.
{{RecordNumber}} or @@ - Current record number. This symbol is displayed as the actual position of the record within the current foundset. The code within the layout controller which retrieves the $record_position value is commented out unless there is a record number symbol on the layout in order to improve performance. This value has to be calculated by retrieving the list of primary key values from the database for every record, since there is no record position value available from SQL database servers. This value is calculated from foundsets automatically based upon using the list of primary keys already stored in the foundset session variable.
{{PageNumber}} or ## - Current page number. This symbol is generally only applicable to printing, and within FileMaker will be displayed as a "?" placeholder if not in print preview mode. This symbol is replaced with the CakePHP Paginator page# of the current page of records being displayed.