



Learn FileMaker Pro 16

The Comprehensive Guide to Building
Custom Databases

—
Mark Conway Munro

Apress®

www.allitebooks.com

Learn FileMaker Pro 16

The Comprehensive Guide to Building
Custom Databases



Mark Conway Munro

Apress®

Learn FileMaker Pro 16: The Comprehensive Guide to Building Custom Databases

Mark Conway Munro
Lewisburg, Pennsylvania, USA

ISBN-13 (pbk): 978-1-4842-2862-3
DOI 10.1007/978-1-4842-2863-0

ISBN-13 (electronic): 978-1-4842-2863-0

Library of Congress Control Number: 2017950736

Copyright © 2017 by Mark Conway Munro

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Cover image designed by Freepik

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Technical Reviewer: Brian Sanchez
Coordinating Editor: Jessica Vakili
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-2862-3. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

Contents at a Glance

About the Author	xxxvii
About the Technical Reviewer	xxxix
Foreword	xli
Acknowledgments	xliii
Introduction	xliv
■ Part I: Using FileMaker	1
■ Chapter 1: Introducing FileMaker.....	3
■ Chapter 2: Exploring the Application.....	19
■ Chapter 3: Exploring a Database Window	45
■ Chapter 4: Working with Records.....	61
■ Chapter 5: Transferring Data	99
■ Part II: Defining Data Structures.....	117
■ Chapter 6: Working with Database Files	119
■ Chapter 7: Working with Tables.....	155
■ Chapter 8: Defining Fields	171
■ Chapter 9: Connecting Tables with Relationships	203
■ Chapter 10: Managing Containers	233
■ Chapter 11: Managing Value Lists.....	243

■ Part III: Writing Formulas and Using Functions	255
■ Chapter 12: Writing Formulas	257
■ Chapter 13: Text Functions	291
■ Chapter 14: Text Functions (JSON)	313
■ Chapter 15: Text Formatting Functions	321
■ Chapter 16: Number Functions	327
■ Chapter 17: Date, Time, and Timestamp Functions	335
■ Chapter 18: Container Functions	343
■ Chapter 19: Aggregate, Summary, and Repeating Functions	357
■ Chapter 20: Financial and Trigonometric Functions	379
■ Chapter 21: Logical Functions	385
■ Chapter 22: Get Functions	401
■ Chapter 23: Design Functions	445
■ Chapter 24: Mobile Functions	457
■ Chapter 25: Creating Custom Functions	461
■ Chapter 26: Introducing ExecuteSQL	477
■ Part IV: Designing Interfaces	499
■ Chapter 27: Introducing Layout Mode	501
■ Chapter 28: Getting Started with Layouts	519
■ Chapter 29: Exploring Layout Palettes	559
■ Chapter 30: Creating Layout Objects	589
■ Chapter 31: Manipulating Objects	661
■ Chapter 32: Using Themes and Styles	681
■ Chapter 33: Customizing Menus	693

■ **Part V: Automating Tasks with Scripts** **719**

■ **Chapter 34: Introduction to Scripting**..... **721**

■ **Chapter 35: Common Scripting Tasks** **749**

■ **Chapter 36: Debugging Scripts**..... **793**

■ **Chapter 37: Using Script Triggers** **801**

■ **Chapter 38: Extending Features with Plugins** **815**

■ **Part VI: Sharing and Networking** **819**

■ **Chapter 39: Introducing Network Sharing**..... **821**

■ **Chapter 40: Securing Database Access**..... **851**

■ **Chapter 41: Advanced Sharing and Analysis Tools** **879**

■ **Chapter 42: Getting Connected**..... **895**

Index..... **899**

Contents

About the Author	xxxvii
About the Technical Reviewer	xxxix
Foreword	xli
Acknowledgments	xlili
Introduction	xliv
■ Part I: Using FileMaker	1
■ Chapter 1: Introducing FileMaker.....	3
The History of FileMaker in a Nutshell	3
Nashoba Nutshell	3
Early FileMaker.....	4
FileMaker 4.....	4
Claris.....	4
FileMaker Inc.	4
Exploring Database Basics.....	5
Defining a Database	5
Database Structure.....	6
Relational Databases.....	7
User Interface	7
Scripted Procedures	8
Identifying Different Database Architectures	8
The Traditional Multitier Database Architecture.....	9
FileMaker’s Integrated File Architecture.....	10

- Taking a Bird’s Eye Anatomical and Functional Overview..... 11**
 - Application Resources 13
 - Database File Resources 13
- The FileMaker Product Family..... 14**
 - FileMaker Pro 15
 - FileMaker Pro Advanced 15
 - FileMaker Server 16
 - FileMaker Go..... 16
 - FileMaker WebDirect 17
- Summary 17**
- Chapter 2: Exploring the Application..... 19**
 - Installing and Launching FileMaker Pro 19**
 - Launching the Installer 19
 - Running the Installation Process 20
 - Launching the Application 21
 - Default Windows 21**
 - Get Started Screen 21
 - Launch Center 22
 - Application Preferences 23**
 - General Settings 23
 - Layout Settings..... 25
 - Memory 26
 - Plug-Ins 26
 - Permitted Hosts 27
 - Default Menu Bar 28**
 - The Application Menu 29
 - File Menu 30
 - Edit Menu..... 34
 - View Menu 35
 - Insert Menu 36

Format Menu	36
Records Menu.....	37
Scripts Menu	38
Tools Menu	38
Window Menu.....	39
Help Menu	40
Contextual Menus.....	40
Contextual Menus for Fields.....	42
Contextual Menus for Record Content Area.....	42
Contextual Menus for Web Viewers	42
Contextual Menus for Window Components.....	43
Contextual Menus for Calculation Formulas.....	43
Summary.....	43
■ Chapter 3: Exploring a Database Window	45
Using a Starter Solution	45
Listing FileMaker's Starter Solutions.....	45
Creating a Database from a Starter Solution.....	46
Defining the Database Window	47
Identifying Window Areas	47
Defining Window Modes.....	48
Defining Content Views.....	49
Exploring the Window Header	49
Status Toolbar (Browse Mode).....	50
Formatting Bar.....	54
Window Ruler	55
Managing Multiple Windows	56
Creating a New Window	56
Closing a Window	57
Selecting a Window from the Menu.....	57
Hiding and Showing Windows	57

- Opening an Existing Database..... 58
 - Opening a Database through the Launch Center 59
 - Managing Favorite Databases 60
- Summary..... 60
- Chapter 4: Working with Records..... 61**
- Entering Data..... 61
 - Opening a Record 61
 - Understanding Field Focus 62
 - Moving to Another Field with Tab Order..... 62
 - Modifying Field Contents 63
 - Formatting Field Contents 64
 - Closing a Record..... 64
- Managing Records 65
 - Creating a New Record..... 65
 - Deleting a Record 65
 - Deleting All Records 66
 - Duplicating a Record 66
- Searching for Records..... 67
 - Performing Fast Searches 67
 - Using Find Mode..... 70
- Working with Found Sets 84
 - Defining a Found Set 84
 - Changing the Records in the Found Set 84
 - Sorting Records in the Found Set..... 85
 - Modifying Field Values for Records in the Found Set 87
 - Find and Replace 90
 - Spell Checking..... 91
- Printing..... 93
 - Page Setup 93
 - Using Preview Mode..... 93
 - Print Dialog Options..... 96
- Summary..... 97

- **Chapter 5: Transferring Data** **99**
- Importing and Exporting Records **99**
- Supported File Types **100**
- Downloading Sample Data **100**
- Importing Records **101**
- Exporting Records **110**
- Exporting Field Contents **115**
- Summary **116**
- **Part II: Defining Data Structures** **117**
- **Chapter 6: Working with Database Files** **119**
- Creating a New Database File **119**
- Configuring a Sandbox Table **120**
- Changing the Default Table Name **120**
- Creating Data Entry Fields **121**
- Creating a Calculation Field **123**
- Creating a Summary Field **123**
- Setting Up Layouts **125**
- Database File Options **130**
- Open Settings **131**
- Icon Settings **132**
- Spelling Settings **133**
- Text Settings **133**
- Script Triggers Settings **135**
- Maintaining Healthy Files **135**
- Database Maintenance Functions **135**
- Avoiding Design and Training Deficiencies **140**
- Restraining File Size **143**
- Understanding File Damage **144**

Specifying File Paths.....	147
Formatting Paths	148
Building Dynamic Paths.....	152
Summary.....	153
Chapter 7: Working with Tables.....	155
Modeling Tables	155
Introducing the Manage Database Dialog (Tables).....	156
Table Properties.....	157
Naming Tables.....	157
Managing Tables	159
Selecting Tables.....	159
Adding Tables	160
Modifying a Table Name	169
Deleting Tables	169
Summary.....	170
Chapter 8: Defining Fields	171
Defining Field Data Types.....	172
Entry Fields.....	172
Display Fields	176
Exploring Field Content Types	176
Introducing the Manage Database Dialog (Fields).....	177
Naming Fields	178
Highlighting the Visibility of Field Names	178
Defining Standard Fields.....	179
Ideas for Standard Fields.....	179
Naming Standard Fields	180
Managing Fields.....	182
Selecting Fields	182
Adding Fields.....	183
Modifying a Field Name.....	185

Modifying a Field Type	185
Deleting Fields.....	186
Setting Field Options	186
Options for Entry Fields	187
Options for Display Fields.....	196
Expanding the Learn FileMaker Object Model.....	199
Defining Standard Fields	199
Defining Custom Fields.....	200
Summary.....	201
■ Chapter 9: Connecting Tables with Relationships	203
Introducing Relationships.....	203
Defining Relationships.....	203
Defining Uses for Relationships.....	204
Defining Connection Types	205
Organizing Relationships.....	209
Planning the Learn FileMaker Relational Object Model.....	212
Managing Data Sources	212
Introducing the Manage External Data Sources Dialog	212
Exploring the Edit Data Source Dialog.....	214
Introducing the Manage Database Dialog (Relationships).....	215
Working with Table Occurrences.....	216
Selecting Table Occurrences	216
Interacting with Table Occurrences	217
Arranging Table Occurrences.....	218
Viewing Options.....	220
Formatting Table Occurrences.....	220
Editing Table Occurrences	221
Adding Table Occurrences.....	222
Deleting Occurrences	223
Printing the Relationship Graph.....	224

Building Relationships	224
Adding Relationships.....	224
Manipulating Relationships.....	226
Adding Notes to the Graph	230
Finalizing the Learn FileMaker Relational Model	231
Summary	232
■ Chapter 10: Managing Containers	233
Inserting Files into Containers	233
Insert Menu.....	234
Drag and Drop.....	234
Copy and Paste.....	235
Understanding Container Storage Options	235
Storing the Actual File.....	235
Storing a Reference to a File.....	235
Using Managed External Storage	237
Defining Base Directories.....	237
Defining a Field to Store Container Data Externally.....	240
Extracting Files from Containers	242
Summary	242
■ Chapter 11: Managing Value Lists	243
Using Value Lists	243
Exploring the Value List Interface	244
Introducing the Value Lists Dialog.....	244
Introducing the Editing Value List Dialog.....	245
Creating Value Lists	246
Using Custom Values.....	246
Using Values from Another File.....	246
Using Values from a Field.....	247
Building Value Lists from Field Values.....	249
Summary	254

■ Part III: Writing Formulas and Using Functions.....	255
■ Chapter 12: Writing Formulas	257
Introducing Formulas	258
How Formulas Work.....	259
Defining Formula Components.....	260
Comments	260
Constants.....	261
Field References.....	262
Functions	263
Operators.....	265
Reserved Words.....	272
Variables.....	274
Exploring the Calculation Interface	277
Exploring the Specify Calculation Dialog.....	278
Writing Formulas.....	279
Getting Started with Simple Formulas.....	279
Inserting Components into Formulas.....	282
Creating Repeating Calculation Fields.....	286
Using Space to Your Advantage	287
Adding Calculations to the Learn FileMaker File	289
Company Contact Count	289
Contact Address Label.....	290
Summary.....	290
■ Chapter 13: Text Functions	291
Text Function Reference.....	291
Char	292
Code.....	293
Exact.....	293
Filter	294
FilterValues.....	294
Furigana.....	295

GetAsCSS	295
GetAsDate	295
GetAsNumber	296
GetAsSVG	297
GetAsText	297
GetAsTime	298
GetAsTimestamp	298
GetAsURLEncoded	299
GetValue	299
Hiragana	300
KanaHankaku	300
KanaZenkaku	300
KanjiNumeral	300
Katakana	300
Left	300
LeftValues	301
LeftWords	301
Length	302
Lower	302
Middle	302
MiddleValues	303
MiddleWords	303
NumToJText	303
PatternCount	303
Position	304
Proper	305
Quote	305
Replace	306
Right	306
RightValues	306
RightWords	307
RomanHankaku	307

RomanZenkaku	307
SerialIncrement	307
SortValues	308
Substitute	309
Trim.....	310
TrimAll.....	310
UniqueValues.....	311
Upper	311
ValueCount.....	311
WordCount.....	312
Summary.....	312
■ Chapter 14: Text Functions (JSON)	313
Introducing JSON	313
Defining the JSON Format	313
Referring to JSON Elements	315
JSON Function Reference	317
JSONDeleteElement.....	317
JSONFormatElements.....	317
JSONGetElement	318
JSONListKeys	318
JSONListValues.....	319
JSONSetElement.....	319
Summary.....	320
■ Chapter 15: Text Formatting Functions	321
Text Formatting Function Reference	321
RGB.....	322
TextColor.....	322
TextColorRemove	323
TextFont	323
TextFontRemove	324
TextFormatRemove.....	324

TextSize	325
TextSizeRemove.....	325
TextStyleAdd	326
TextStyleRemove	326
Summary.....	326
■ Chapter 16: Number Functions.....	327
Number Function Reference	327
Abs.....	328
Ceiling.....	328
Combination	328
Div.....	329
Exp.....	329
Factorial.....	330
Floor.....	330
Int	330
Lg.....	331
Ln.....	331
Log.....	331
Mod.....	332
Random	332
Round	333
SetPrecision.....	333
Sign	333
Sqrt.....	334
Truncate.....	334
Summary.....	334
■ Chapter 17: Date, Time, and Timestamp Functions	335
Date Function Reference.....	335
Date	336
Day.....	336
DayName	336

DayNameJ	337
DayOfWeek	337
DayOfYear	337
Month	337
MonthName	338
MonthNameJ	338
WeekOfYear	338
WeekOfYearFiscal	339
Year	339
YearName	339
Time Function Reference	340
Hour	340
Minute	340
Seconds	341
Time	341
Timestamp Function Reference	341
Timestamp	341
Summary	342
■ Chapter 18: Container Functions	343
Container Function Reference	343
Base64Decode	343
Base64Encode	344
Base64EncodeRFC	344
CryptAuthCode	345
CryptEncrypt	346
CryptEncryptBase64	346
CryptDecrypt	347
CryptDecryptBase64	347
CryptDigest	347
GetContainerAttribute	348
Examples	350

GetHeight	351
GetThumbnail	352
GetWidth	352
HexEncode	353
HexDecode	353
TextEncode	354
TextDecode	354
VerifyContainer	355
Summary	355
Chapter 19: Aggregate, Summary, and Repeating Functions	357
Aggregate Function Reference	357
Average	358
Count	360
List	362
Max	363
Min	365
StDev	367
StDevP	369
Sum	370
Variance	372
VarianceP	374
Summary Functions	376
GetSummary	376
Repeating Function Reference	377
Extend	377
GetRepetition	378
Last	378
Summary	378
Chapter 20: Financial and Trigonometric Functions	379
Financial Function Reference	379
FV	379

NPV	380
PMT	380
PV	380
Trigonometric Function Reference	381
Acos	381
Asin	382
Atan	382
Cos	382
Degrees	382
Pi	383
Radians	383
Sin	383
Tan	384
Summary.....	384
■ Chapter 21: Logical Functions.....	385
Logical Function Reference.....	385
Case	386
Choose	386
Evaluate.....	387
EvaluationError	388
ExecuteSQL.....	389
GetAsBoolean	389
GetAVPlayerAttribute	389
GetField	390
GetFieldName	391
GetLayoutObjectAttribute	392
GetNthRecord	394
If	394
IsEmpty	395
IsValid	395
IsValidExpression	395

Let	396
Lookup	397
LookupNext	398
Self	399
Summary	399
■ Chapter 22: Get Functions	401
Get Function Reference	401
Get (AccountExtendedPrivileges)	404
Get (AccountGroupName)	405
Get (AccountName)	405
Get (AccountPrivilegeSetName)	405
Get (ActiveFieldContents)	406
Get (ActiveFieldName)	406
Get (ActiveFieldTableName)	406
Get (ActiveLayoutObjectName)	406
Get (ActiveModifierKeys)	407
Get (ActivePortalRowNumber)	407
Get (ActiveRepetitionNumber)	408
Get (ActiveSelectionSize)	408
Get (ActiveSelectionStart)	408
Get (AllowAbortState)	409
Get (AllowFormattingBarState)	409
Get (ApplicationArchitecture)	409
Get (ApplicationLanguage)	410
Get (ApplicationVersion)	410
Get (CalculationRepetitionNumber)	411
Get (ConnectionAttributes)	411
Get (ConnectionState)	411
Get (CurrentDate)	412
Get (CurrentExtendedPrivileges)	412
Get (CurrentHostTimestamp)	412

Get (CurrentPrivilegeSetName) 413

Get (CurrentTime) 413

Get (CurrentTimestamp) 413

Get (CurrentTimeUTCMilliseconds) 413

Get (CustomMenuSetName) 414

Get (DesktopPath) 414

Get (Device) 415

Get (DocumentsPath) 415

Get (DocumentsPathListing) 416

Get (EncryptionState) 416

Get (ErrorCaptureState) 417

Get (FileMakerPath) 417

Get (FileName) 417

Get (FilePath) 417

Get (FileSize) 418

Get (FoundCount) 418

Get (HighContrastColor) 418

Get (HighContrastState) 419

Get (HostApplicationVersion) 419

Get (HostIPAddress) 419

Get (HostName) 420

Get (InstalledFMPlugins) 420

Get (LastError) 420

Get (LastExternalErrorDetail) 421

Get (LastMessageChoice) 421

Get (LayoutAccess) 421

Get (LayoutCount) 422

Get (LayoutName) 422

Get (LayoutNumber) 422

Get (LayoutTableName) 422

Get (LayoutViewState) 423

Get (MenubarState) 423

Get (ModifiedFields)	423
Get (MultiUserState)	424
Get (NetworkProtocol)	424
Get (NetworkType)	424
Get (PageNumber)	425
Get (PersistentID)	425
Get (PreferencesPath)	425
Get (PrinterName)	426
Get (QuickFindText)	426
Get (RecordAccess)	426
Get (RecordID)	427
Get (RecordModificationCount)	427
Get (RecordNumber)	427
Get (RecordOpenCount)	427
Get (RecordOpenState)	428
Get (RegionMonitorEvents)	428
Get (RequestCount)	428
Get (RequestOmitState)	429
Get (ScreenDepth)	429
Get (ScreenHeight)	429
Get (ScreenScaleFactor)	430
Get (ScreenWidth)	430
Get (ScriptAnimationState)	430
Get (ScriptName)	430
Get (ScriptParameter)	431
Get (ScriptResult)	431
Get (SortState)	432
Get (StatusAreaState)	432
Get (SystemDrive)	432
Get (SystemIPAddress)	433
Get (SystemLanguage)	433
Get (SystemNICAddress)	433

Get (SystemPlatform)	434
Get (SystemVersion)	434
Get (TemporaryPath)	434
Get (TextRulerVisible)	435
Get (TotalRecordCount)	435
Get (TouchKeyboardState)	435
Get (TriggerCurrentPanel)	436
Get (TriggerExternalEvent)	436
Get (TriggerGestureInfo)	437
Get (TriggerKeystroke)	437
Get (TriggerModifierKeys)	437
Get (TriggerTargetPanel)	438
Get (UserCount)	438
Get (UserName)	438
Get (UseSystemFormatsState)	439
Get (UUID)	439
Get (WindowContentHeight)	439
Get (WindowContentWidth)	440
Get (WindowDesktopHeight)	440
Get (WindowDesktopWidth)	440
Get (WindowHeight)	440
Get (WindowLeft)	441
Get (WindowMode)	441
Get (WindowName)	441
Get (WindowOrientation)	442
Get (WindowStyle)	442
Get (WindowTop)	442
Get (WindowVisible)	443
Get (WindowWidth)	443
Get (WindowZoomLevel)	443
Summary	443

■ Chapter 23: Design Functions.....	445
Design Function Reference	445
DatabaseNames	446
FieldBounds	446
FieldComment	447
FieldIDs	447
FieldNames	448
FieldRepetitions	448
FieldStyle	449
FieldType	449
GetNextSerialValue	450
LayoutIDs	450
LayoutNames	451
LayoutObjectNames	451
RelationInfo	452
ScriptIDs	453
ScriptNames	453
TableIDs	453
TableNames	454
ValueListIDs	454
ValueListItems	455
ValueListNames	455
WindowNames	456
Summary.....	456
■ Chapter 24: Mobile Functions.....	457
Mobile Function Reference	457
Location	457
LocationValues	458
RangeBeacons	459
Summary.....	460

■ Chapter 25: Creating Custom Functions	461
Defining Custom Functions	461
Exploring the Custom Function Dialogs.....	462
The Manage Custom Functions Dialog	462
The Edit Custom Function Dialog.....	464
Building Custom Functions.....	465
Creating a Simple Custom Function	465
Adding Parameters to a Custom Function	467
Accessing Fields from Custom Functions.....	471
Building Recursive Custom Functions	472
Summary.....	475
■ Chapter 26: Introducing ExecuteSQL	477
Defining the ExecuteSQL Function	478
How the ExecuteSQL Function Works.....	478
Understanding Some Limitations of ExecuteSQL	480
Creating SQL Queries	481
Defining SELECT Statements	481
Using the SELECT Statement.....	483
Inserting Literal Text in the Field List.....	489
Concatenating Results.....	490
Using the WHERE Clause	491
Using the ORDER BY Clause	492
Using the JOIN Clause	492
Using the GROUP BY Clause	493
Using the UNION Clause.....	494
Limiting the Results of a Query	494
Accessing the Database Schema	496
Selecting FileMaker_Tables.....	496
Selecting FileMaker_Fields	497
Exploring Other SQL Features	498
Summary.....	498

■ Part IV: Designing Interfaces	499
■ Chapter 27: Introducing Layout Mode	501
Emphasizing the Importance of Layout Design	501
Using Layout Mode	502
Identifying Layout Mode	502
Status Toolbar (Layout Mode)	504
Menu Changes (Layout Mode)	509
Summary	518
■ Chapter 28: Getting Started with Layouts	519
Planning Layouts	519
Classifying Layouts into Type Categories	519
Sketching a Layout Flow Chart	523
Working with Layout Parts	524
Exploring Layout Regions and Controls	524
Defining Part Types	526
Managing Parts	527
Adding Layouts	530
Creating a New Layout	530
Configuring Layouts	550
Deleting Layouts	556
Using the Manage Layouts Dialog	556
Optimizing Layout Performance	558
Summary	558
■ Chapter 29: Exploring Layout Palettes	559
Exploring the Field Picker	559
Exploring the Inspector	561
Panel One — Defining Position Settings	562
Panel Two — Style Settings	568
Panel Three — Appearance Settings	568
Panel Four — Data	576

Exploring the Layout Objects Palette.....	585
Using the Shortcuts in the Layout Objects Palette Contextual Menu.....	586
Summary.....	588
■ Chapter 30: Creating Layout Objects	589
Introducing Layout Objects.....	589
Inserting an Object onto the Layout.....	590
Configuring an Object.....	590
Working with Field Objects.....	591
Clarifying Field Terminology	591
Field Basics	592
Configuring Field Control Style	596
Exploring the Benefits of Pop-Up Menus for Two-Field Value List.....	610
Bypassing Value List Entry Restrictions.....	613
Using Placeholders	616
Working with Text.....	619
Static Text.....	620
Dynamic Placeholder Symbols	620
Merge Fields.....	620
Merge Variables.....	621
Working with Button Controls	621
Working with Buttons	621
Popover Button	628
Button Bar.....	632
Working with Panel Controls	635
Tab Control.....	635
Slide Control	638
Working with Portals	639
Illustrating How Portals Work	640
Exploring the Portal Setup Dialog.....	640
Adding Objects to Portal Rows	642
Creating Records in a Portal.....	642

Deleting Portal Rows	644
Filtering Portal Records	646
Working with Web Viewers.....	650
Exploring the Web Viewer Setup Dialog.....	650
Interacting with Web Viewer Content	652
Building a Web Page Using Data from Fields.....	653
Working with Charts.....	655
Creating Charts.....	657
Summary.....	660
■ Chapter 31: Manipulating Objects	661
Selecting Objects	661
Resizing Objects.....	662
Moving Objects.....	662
Layout Positioning Helpers	663
Arranging & Aligning Objects	667
Align.....	668
Resize To.....	668
Distribute or Space	669
Rotate	670
Group	670
Lock	671
Object Stacking	671
Hiding Objects	672
Conditional Formatting.....	672
Condition Formula Options	674
Conditional Format Settings	675
Conditionally Formatting an Invoice Status Field	675
Understanding Tab Order.....	677
Changing the Tab Order	677
Using Functions of the Set Tab Order Dialog	678

Naming Objects	679
Summary	679
■ Chapter 32: Using Themes and Styles	681
Introducing Themes and Styles	681
Defining Styles.....	681
Defining Themes.....	685
Using Themes.....	686
Choosing a Layout's Theme	687
Managing Themes	688
Using Styles.....	689
Changing an Object's Style Assignment	689
Editing a Style's Settings.....	690
Summary.....	691
■ Chapter 33: Customizing Menus.....	693
Exploring the Capabilities of Customize Menus	693
Getting Started with Custom Menus Basics.....	694
Exploring Custom Menu Set Options	695
Understanding Customization Exceptions	696
Exploring the Manage Custom Menus Dialog.....	697
Creating a Custom Menu Set.....	698
Adding Copies of Standard FileMaker Menus.....	699
Customizing Menu Items.....	704
Exploring the Edit Custom Menu Dialog	704
Modifying Standard Menu Items.....	707
Adding a Custom Menu	709
Installing a Menu Set.....	712
Exploring the Link Between Commands and Menus.....	713
Creating a Status-Based Custom Menu	716
Summary.....	717

■ Part V: Automating Tasks with Scripts	719
■ Chapter 34: Introduction to Scripting	721
Identifying the Benefits of Scripting	721
Exploring Scripting Basics	722
Calling Other Scripts	723
Understanding the Importance of Context	727
Exploring the Script Workspace	728
Workspace Toolbar	729
Script Pane	729
Script-Editing Pane	731
Script Step Pane	732
Script Workspace Menu Changes	734
Writing Scripts	738
Script Step Basics	739
Summary	748
■ Chapter 35: Common Scripting Tasks	749
Allowing User Abort	749
Setting Variables	750
Creating Navigation Scripts	751
Go to Layout	751
Go to Record/Request/Page	753
Go to Related Record	754
Interacting with Fields	756
Go to Field	757
Set Field	757
Set Field by Name	758
Set Selection	759
Set Next Serial Value	759
Get Directory	761

Working with Records	762
Import Records	762
Export Records	763
Using Show Custom Dialog	764
Configuring a Dialog	764
Configuring a Dialog for Field Input	766
Capturing the Button Click.....	767
Conditionally Branching with If Statements	767
Building Simple If Statements	768
Building Compound If Statement.....	769
Building Nested If Statement.....	769
Performing Searches.....	770
Working with Found Sets	771
Iterating with a Loop Statement.....	772
Looping Through a Found Set of Records.....	772
Looping Through Lists of Data.....	774
Confirming a Dialog Field Value.....	775
Scripting Portal Functions	777
Navigating to a Portal Row's Record	777
Adding a Portal Row	778
Managing Windows	779
Creating a New Window	780
Introducing the Card Window	788
Managing Errors.....	790
Summary.....	791
■ Chapter 36: Debugging Scripts.....	793
How Script Debugging Works.....	793
Uses for the Script Debugger	794
Defining Basic Debugging Concepts.....	794
Activating the Debugger	795

Exploring the Debugging Interface	795
Debugging Options Under the Tools Menu	795
Exploring the Script Debugger Window	796
Exploring the Data Viewer Window	797
Using Custom Breakpoints	798
Summary	799
■ Chapter 37: Using Script Triggers	801
Script Trigger Reference	801
File Option Triggers	802
Layout Triggers	804
Layout Object Triggers	809
Working with Script Triggers	811
Finding Script-Triggered Objects	812
Remembering Event Precedence	812
Using Caution with Script Parameters	813
Exceptions to Event Triggers	814
Summary	814
■ Chapter 38: Extending Features with Plugins	815
Purchasing Plug-Ins	815
360 Works	815
Productive Computing	815
Trio Automatisering	816
Finding Other Plug-ins	816
Locating Installed Plug-Ins	816
Installing Plug-Ins	817
Accessing Plug-In Functions	817
Summary	818

■ Part VI: Sharing and Networking	819
■ Chapter 39: Introducing Network Sharing	821
How Network Sharing Works	821
Limitations to Sharing	823
Setting Up Peer-to-Peer Database Sharing	823
Using a FileMaker Server	825
Understanding Client License Options and Limitations	825
Exploring FileMaker Server Basics	827
Hosting Databases with FileMaker Cloud	840
Accessing Databases from Outside FileMaker	841
Sharing Bookmarks with Snapshot Links	841
Using the FileMaker URL	842
Using AppleScript to Automate FileMaker	845
Summary	849
■ Chapter 40: Securing Database Access	851
Exploring the Benefits of Security	851
Defining Default Security	851
Exploring the Security Dialog	852
Exploring the Interface Modes	853
Creating Accounts	856
Creating a New Local Account	856
Creating a New External Account	858
Exploring Privilege Sets	861
Default Privilege Sets	862
Creating Custom Privilege Sets	862
Exploring Extended Privileges	871
Defining Built-In Extended Privileges	871
Using Custom Extended Privileges	872
Controlling File Access	876
Summary	877

- **Chapter 41: Advanced Sharing and Analysis Tools 879**
 - Database Design Report..... 879
 - Generating a Design Report..... 879
 - Exploring a Design Report 881
 - Introducing Professional DDR Tools 884
 - Developer Utilities 885
 - Exploring the Developer Utilities Dialog..... 885
 - Specifying Solution Options..... 887
 - Summary 894
- **Chapter 42: Getting Connected 895**
 - Connect with Mark Munro 895
 - Learn FileMaker Pro 16 Official Sites 895
 - FileMaker Inc. 896
 - Database Design Report (DDR) Analysis Tools..... 896
 - Plug-Ins 896
 - Communities 896
 - Technology Sites 897
 - Function Background Information..... 897
- Index..... 899**

About the Author

Mark Conway Munro is an author, software developer and entrepreneur. A 1988 introduction to FileMaker led Mark to pursue a career as a developer. Soon after, he was using AppleScript and FileMaker for information management and process automation. In 1994, Mark founded Write Track Media, a computer consultancy to develop custom database and workflow automation solutions for clients.

Today, Mark continues in that mission, using AppleScript and FileMaker Pro in combination with other tools and technologies to build time-saving systems for a diverse list of clients. By fully understanding the virtue of using technology to free humans from repetitive computer tasks so they can pursue a higher level of productivity, Mark's passion for the work is unwavering.

Mark is also the author of *AppleScript: Developer Reference* (2010, Wiley).

About the Technical Reviewer

Brian Sanchez is a FileMaker developer, a consultant with wide experience, and a founding member of aACE Software. Brian's experience is wide reaching. For nearly 20 years, he has designed and implemented custom apps for inventory tracking, digital catalogs built with FileMaker Go, plus asset management and pricing tools. His clients range from catering businesses to TV casting agencies to European vacationing services. Brian holds multiple FileMaker certifications. His development philosophy is to build systems with such integrity in the code and architecture that the client never needs to call again — unless they're looking to upgrade the solution. Brian is the technical editor of *Learn FileMaker Pro 16* (2017, Apress).

Foreword

In the 1990s and 2000s, I ran TECSoft, an AppleScript and FileMaker Pro development shop in New York City. Our specialty was database publishing – the powerful ability of Apple’s AppleScript technology to intelligently extract data from FileMaker databases and to format it ready-for-print in page layout applications such as QuarkXPress and Adobe InDesign.

We worked closely with Apple and Claris, the predecessor of FileMaker Inc., to showcase the abilities of their technologies. In 1994, growing – and needing additional developer talent, we asked our pal Eric Silver, the then New York area sales rep at Claris, to be on the lookout for us. He told me about a FileMaker wizard he knew named Mark Munro working at Jack Morton, an AV production company in New York.

Mark was responsible for organizing Morton’s production project data and managing their workflow. We met and he explained their workflow and showed me how he had organized it in FileMaker so that management was able to track projects from start to finish, track the various assets involved, and track and accurately bill their clients for the time and materials involved.

Mark’s work was amazing and I immediately realized we had come across FileMaker gold. All aspects of his company’s workflow were logically organized in a functional workflow management tool – and most impressively of all – was clearly present in an easy-to-follow interface. It is mind-bending to realize that Mark had accomplished this before FileMaker was a relational database.

We began working with Mark as a freelance FileMaker Pro and AppleScript gun slinger. We threw our most complicated and demanding projects at him – a huge workflow management and catalog publishing project at Sony Music, a very complex and mission-critical graphic file creation project with drop-dead deadlines for Associated Press, to name just a couple.

In short order, Mark had more than enough work to go out on his own as a full-time FileMaker and AppleScript developer. As TECSoft’s focus turned to working with Apple delivering AppleScript seminars and training sessions, we created the TECSoft Developers Consortium (TDC) to handle the huge demand for FileMaker and AppleScript integration projects our seminars generated. With Mark as TDC developer #1, we were able to create an international referral network of FileMaker Pro and AppleScript developers that became the largest FileMaker Pro and AppleScript brain trust on earth.

Mark has now been running his own successful development company, Write Track Media, for over 20 years. What Mark possesses, more than any other developer I’ve worked with, is a true talent for organization. Mark’s passion makes him eminently qualified to pass along his mastery of FileMaker’s powerful organization tool. He just had to write this book.

John Thorsen Jr.
Connected Hearth, LLC

Acknowledgments

This book was an enormous challenge and would not have been possible without the contributions and encouragement of many.

Some long overdue thanks to Peter Samelson and Tony Spina who played a role in my early introduction to FileMaker decades ago. To Sylvia Griffiths for hiring me at Jack Morton and to Laura Shuler for asking me to develop a budget management database. To John Thorsen, Jr., for development opportunities he offered me and for being such a prolific source of advice, knowledge, and humor.

Continuing thanks to all my incredible clients for finding new challenges that push me to find new technological solutions to improve their database and workflow systems.

To the whole team at Apress for providing me with this opportunity and for all their hard work in bringing everything together.

Most of all, a huge thanks to two close friends. To Brian Sanchez for his work reviewing the material with a keen eye for technical accuracy. His feedback led to many key improvements that greatly enhanced the quality of the material. Also, to Walker Stevenson for reading a draft of the manuscript and providing some additional technical notes and suggestions for improvement.

Introduction

Back in late-1988 while working at Tannen's Magic Store in Manhattan, I was introduced to FileMaker. A discussion arose about using computers to improve the efficiency and accuracy of the pen and paper-based ordering system. To do that, we first needed to enter the inventory into a database. My boss handed me what was then a state of the art personal computer; a Macintosh SE, which was a small box with a tiny 9-inch black and white screen. It was loaded with a copy of the newly released FileMaker II by Claris Corporation. That version was expressed in Roman numerals and this was two years before the "Pro" suffix would be added and the versioning reset to 1.0.

With the computer, a small rolling cart and a long extension cord, I headed into the stock room to begin computerizing the inventory. I barely used a computer prior to that day and it was my first glimpse at FileMaker. I spent about ten minutes becoming acquainted with the software, then quickly created a database file and set up a handful of fields. This rapid adoption is not so much a testament to my intuitive ability with software but more attributable to the extremely primitive nature of that early version. The database I created was extraordinarily crude but it allowed us to begin entering product information and to print on paper "labels" that could be taped onto product boxes to improve the hand-written names and prices.

For the next few weeks, with the help of another employee, I entered batches of products into the database. There was no network at the store at the time, so we periodically shuttled the database by 3.5" floppy disk to the main office computer to print out box labels displaying the information. These were affixed to boxes as the next batch of products were entered. Finally, we had a complete database of every available product. From there, I developed an order management system that pulled product information from that inventory file and installed a small network of computers each with access to a server. The project changed order fulfillment at the store and the course of my life.

Fast forward almost three decades, about twenty significant version increments later and we reach FileMaker Pro 16, an astonishingly powerful piece of software. Keeping with tradition, it is packed with features for users and developers in an integrated application. It contains a development environment *and* a customizable runtime. It balances an ease of use making it approachable by beginners *and* contains powerful standard technologies and robust customization options demanded by professionals.

The age of the software and its multi-decade evolution shows in an esthetic hodgepodge of styles throughout the development interface. The design of some dialogs look a little dated when compared to the newer additions; palettes and workspaces sporting a modern design. This seems to hint at a steadfast dedication to stability and a desire to avoid the frequent pitfall of many software titles; changing things for the sake of changing things. This gradual evolution of components contributes to the stable nature of the software across time even as it routinely takes leaps to evolve and keep pace with new technologies.

The complexity of different modes and perspectives so tightly integrated in a single application, and the sheer number of functions, tools and customization options available proved a challenge when deciding how to best present the material in a book. In the end, I divided the material into the following six parts, logically arranged in a hierarchy so that each chapter builds on the previous:

- Part 1 — Using FileMaker
- Part 2 — Defining Data Structures
- Part 3 — Writing Formulas and Using Functions
- Part 4 — Designing Interfaces
- Part 5 — Automating Tasks with Scripts
- Part 6 — Sharing and Networking

My hope is that the arrangement of material might serve *both* those who want to read it cover to cover and those who use it as a reference guide to look up specific topics and examples. I would enjoy hearing from you about your experience with the book and am open to suggestions for how to improve future editions. If you want to send feedback, ask a question, or read exclusive online content, visit my website for the book:

www.learn-filemaker.com

Unlike those early versions I used decades ago, FileMaker Pro 16 is a mature application with a solid track record, evolving feature set, substantial user base that should appeal to professional developers. And, while no one today should expect to learn it in ten minutes, it is still approachable for beginners and casual programmers. I hope this book convinces people to give it a try, both seasoned professionals and newcomers who are beginning their adventure into the world of custom database development.

Mark Conway Munro
May 2017 / Lewisburg, PA

PART I



Using FileMaker

FileMaker Pro combines a runtime user interface with a development environment into a single application. To become familiar with the software, start by reviewing the basics of using the software from a user perspective.

These five chapters are intended primarily for newcomers so readers who have prior experience using databases built with FileMaker may find some of this material relatively basic. However, experienced users and expert developers who are looking to brush up on the latest features may appreciate this as a solid foundation for the remaining portions of the book.

The topics covered in these chapters include:

1. *Introducing FileMaker* — Includes the history of FileMaker, database basics and architectures, a bird's eye overview of FileMaker's unique architecture and the entire FileMaker product family.
2. *Exploring the Application* — Includes FileMaker Pro installation process, default windows, application preferences, the default Browse mode menu bar, and various contextual menus throughout the interface.
3. *Exploring a Database Window* — Using a FileMaker-provided Contacts starter solution, this includes the anatomy of a database window, how to use windows, and how to open an existing database.
4. *Working with Records* — Explores the process of entering, managing, searching, and printing records.
5. *Transferring Data* — Provides an overview of the various methods of importing and exporting records.

CHAPTER 1



Introducing FileMaker

FileMaker Pro is a relational data management system published by Apple subsidiary FileMaker Inc.

Since its origin, the product has expanded and evolved with improved capabilities to keep pace with standard technologies. The current incarnation is a powerful, feature-rich and easy-to-use integrated development tool, available in fifteen different languages. FileMaker databases can be shared across a network with multiple simultaneous users working on Mac and Windows computers as well as iOS devices.

FileMaker is popular with millions of developers and users, including independent consultants, employees of small businesses, and members of teams working at medium to large businesses, non-profits, and government agencies. Beginners are empowered by the ease of use and approachable interface. Professionals are won over by its rapid development capabilities for prototyping and the ability to build powerful user-friendly solutions. Business owners who don't have time to learn to program can easily find a professional consultant to develop a system tailored to meet their individual needs.

Whatever your skill set, if you are frustrated with an existing database system or are looking to develop something new, FileMaker is an excellent choice.

In this chapter, we introduce FileMaker by discussing topics such as the following:

- The history of FileMaker in a Nutshell
- Exploring database basics
- Identifying different database architectures
- Taking a bird's eye anatomical and functional overview
- The FileMaker product family

The History of FileMaker in a Nutshell

The early history of FileMaker is a zigzag between various publishers, and it employed a rather haphazard naming and numbering system before finally settling into its consistent and logical modern track.

Nashoba Nutshell

FileMaker started its life in the early 1980s as *Nutshell*, an MS-DOS computer program that was developed by Nashoba Systems in Concord, Massachusetts, and distributed by electronics marketer Leading Edge.

Early FileMaker

When the Macintosh computer was introduced in early 1984, Nashoba saw an opportunity. They worked to combine the database engine with a graphic user interface and created a new forms-based database product. When Leading Edge announced that they wished to remain a DOS-only vendor, Nashoba began working with a new distributor. Forethought, Inc. FileMaker v1.0 was released in April 1985 for the Macintosh platform. In 1986, it was renamed FileMaker Plus to match the release of the Macintosh Plus.

By today's standards, the early versions of FileMaker were incredibly crude and limited in capabilities. However, for the time, the software filled an important need with an easy-to-use design and steadily became popular with the do-it-yourself crowd.

FileMaker 4

When Microsoft purchased Forethought, they tried to negotiate a purchase of FileMaker, which was outselling their own Microsoft File database application. However, Nashoba declined and decided to begin self-publishing the program, now named FileMaker 4.

Claris

In 1987, Apple formed Claris Corp. as a wholly owned subsidiary to develop and publish Macintosh software titles such as MacWrite and MacPaint. In 1988, Claris purchased Nashoba Systems to acquire FileMaker. By this time Leading Edge and Nutshell had disappeared as other DOS and Windows database products dominated the market.

In 1988, FileMaker II was released by Claris to match the naming scheme of their other products. After a few minor updates, the product was rebranded in 1990 under its modern naming format when FileMaker Pro version 1.0 was released.

Claris upgraded the product in 1992 with Windows support, making FileMaker Pro 2.0 the first cross-platform version. They began publishing a server application in 1994. It wasn't until 1995 with the release of version 3.0 that FileMaker became fully relational.

By 1997, with version 4.0, FileMaker was a widely popular product and was outselling all other Claris products. Apple absorbed those products in-house and renamed the subsidiary after their only remaining product.

FileMaker Inc.

FileMaker Inc., the newly renamed subsidiary, released FileMaker Pro 4.1v2 in June of 1999. Since that time, they have steadily improved and expanded the product. Some highlights are included in Table 1-1.

Table 1-1. A summary of major new features in recent FileMaker Pro versions

Year	Version	Feature Improvements
2001	5.5	Native support for Mac OS X
2004	7.0	Multiple table file architecture, multiple windows, relationship graph, calculation variables
2005	8.0	Tabs on layouts, script variables
2006	8.5	Web Viewer, layout object names
2007	9.0	SQL support, conditional formatting of layout objects
2009	10	Script triggering
2010	11.0	Charting, snapshot link, filtered portals, first version of FileMaker Go for iOS released
2012	12.0	Themes, floating and modal windows, ExecuteSQL function introduced, enhanced container field, improved charting
2013	13.0	WebDirect and HTML5 features; enhanced interface, hide layout object calculation, enhanced ExecuteSQL function, Perform Script on Server script step
2015	14.0	Script workspace, Button Bar layout tools, Launch Center
2016	15.0	Enhanced script workspace, concealed edit box
2017	16.0	Layout Objects window, copy/paste value lists, JSON support, enhanced security, enhanced script steps, built-in function changes, variable support in data sources, extended privilege changes

Since its start in 1985 as a modest, single-user database, the product has greatly improved. What began as a simple, easy-to-use tool for non-technical people has matured into a considerably more sophisticated platform. Today, FileMaker offers an astonishing array of technical features and capabilities that are sure to fill the most advanced development needs. By integrating this power with an intuitive graphical interface, it continues to be approachable by those new to programming.

Exploring Database Basics

Before getting into the specifics of FileMaker, let's review a few basic database concepts.

Defining a Database

A *database* is a structured collection of information efficiently stored in a generic format that makes it easily accessibility for reuse in a variety of different ways.

Most applications are focused on one specific data type with predefined properties and procedures. A calendar application allows a user to create and manage events with properties of date, time, attendees, notes, and alerts. Similarly, an email application allows management of messages with predefined properties of sender, recipients, subject, and body. These could each be thought of colloquially as an “event-base” and a “message-base” since, at their *base*, they store and provide access to *events* and *messages*.

From a wider perspective, both events and messages are types of information; they are both forms of *data*. A calendar app manages an event *database*. A mail application manages a message *database*. Unlike these, an open-ended database application like FileMaker Pro has no predefined data type; at its base is data, *any data*. It allows a developer-user to create a custom management system for *any* kind of data depending on a specific custom need.

Specific applications, such as the calendar or mail app mentioned above, provide a metaphorical filing cabinet that is locked and preconfigured to fit only certain predefined folders containing a specific type of information. A user is free to enter information into this predefined framework but have little or no control over the framework itself. By contrast, a database is the metaphorical equivalent of that same filing drawer completely empty and unlocked so that *you* can define the content it accepts, how that information is stored, related, displayed, used and shared. Every project starts fresh with the same blank slate waiting for you to define the framework and establish the capabilities it will make available to the user.

Databases can store information about *anything* a developer defines them to manage: companies, messages, notes, people, products, tasks, etc.

Database Structure

Fundamentally, a database is made up of only a few basic components. These include tables, relationships, a user interface, and scripted procedures.

A *table* can be visualized as a set of rows and columns in a spreadsheet tab that is dedicated to a single type of entity. The simple table example, shown in Figure 1-1, is modeled after People.

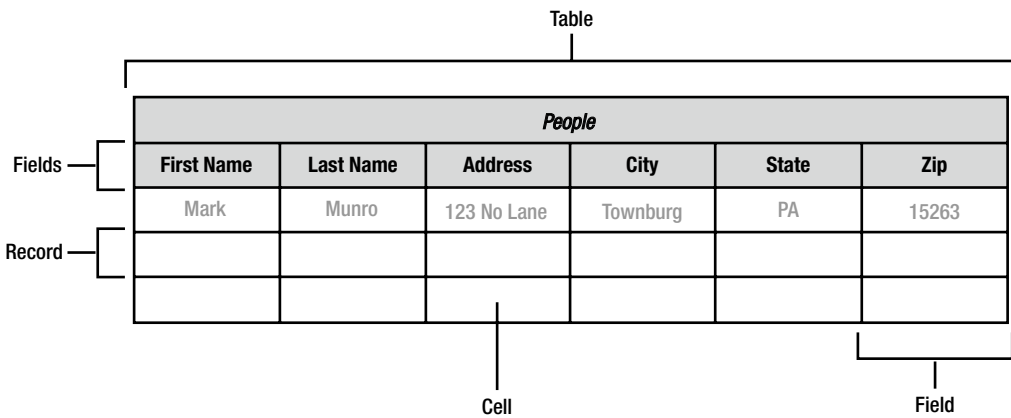


Figure 1-1. The basic anatomy of a database table

A *field*, represented by a column in the table, is a defined container in which one piece of information about the entity modeled will be stored. In the example shown, each component of a name and address is an individual field.

A *record*, represented by a row in the table, is one individual entity stored within a table. While the table in the example represents people in general, a row, or record, represents one person. The record is one instance of the table’s defined field set containing information about a specific person.

A *cell* is the formal name of an intersection of a field and record. In FileMaker, these are commonly referred to as *fields* with the understanding that each is an instance of a field for a record (cell), which is distinct from the field definition itself (field).

Relational Databases

A database is considered *relational* when its tables can be interconnected to form relationships between the records they contain based on one or more key fields. While any field can act as a key, most often a unique identification number is assigned to each record, which is used to locate matching values in a field in another table.

A relationship allows information from one table to be accessed from the other through the connection. They allow data to be stored in separate tables but used elsewhere and help to eliminate redundant data entry.

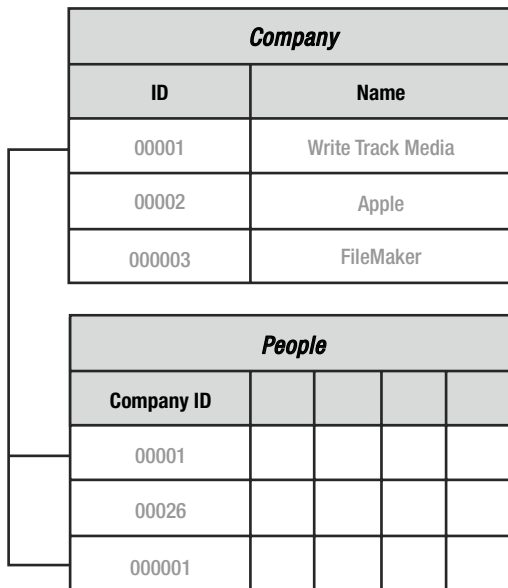


Figure 1-2. An example relationship established between two tables

A *relationship*, like the one shown in Figure 1-2, is a connection between two tables established by linking one or more fields from one table to one or more fields in another. In this example, the unique ID field in the Company table is linked to the Company ID field in the People table forming the relationship. We see that there are two matches, meaning two people have been linked to the same company. Using the relationship, the two-related people can be displayed on a layout for the company and used in calculations and scripted procedures from within that context.

User Interface

The *user interface* is made up of the layouts, menus, and windows that allows the user to view and interact with the data in a graphical, human-friendly format.

A *layout* is a visual display of fields and controls from the perspective of a single table's records. Tables can have many layouts for different purposes. For example, a *list* layout might show a couple fields for all the records for a table to allow faster navigational scrolling. Another layout might show every field for a single record, optimized for *data entry*.

The *current layout* is whichever layout is being displayed in the database window at a given time. This layout can be changed to another by the user manually selecting a different one or by a script. Figure 1-3 shows a database with two tables, each with a list and entry layout. The window is shown currently displaying the People Entry layout.

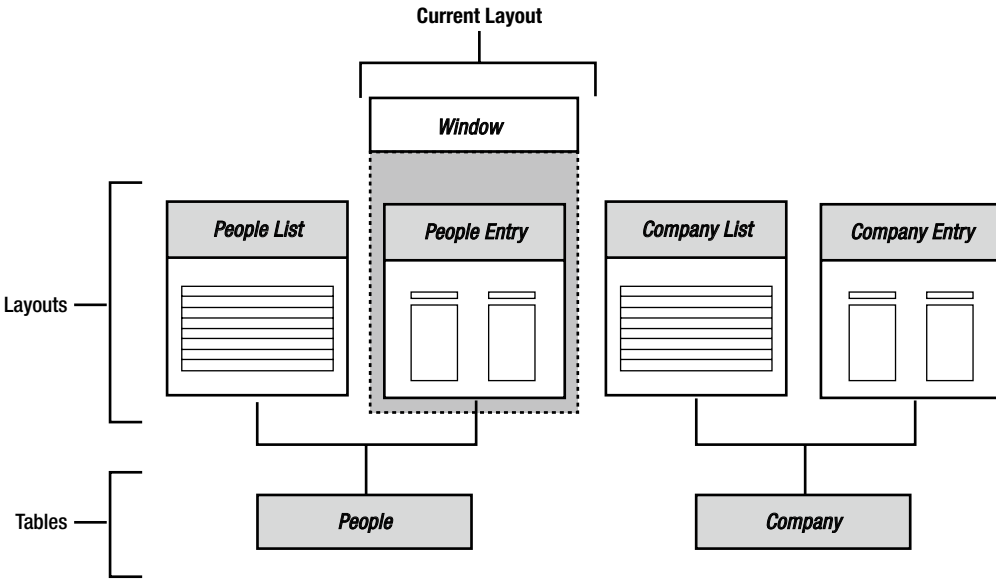


Figure 1-3. Illustrating layouts displaying data from their corresponding tables with the window displaying the current layout

A *found set* of records is a subset of the total records contained in the table being displayed in a window, typically after a search. The *current record* is whichever record from the found set is displayed within a window showing one record or whichever record is selected within a window showing a list of records.

Scripted Procedures

A *script* is a stored set of actions that can be automatically performed in response to user activity or on a schedule. Scripts can generally perform any function with which a user has access.

Identifying Different Database Architectures

FileMaker employs a uniquely *integrated architecture* that skillfully manages to maintain an intuitive environment that allows most users to acclimate regardless of their level of development experience. However, it may be somewhat foreign to experienced developers who are familiar with traditional multitier architecture of database systems like SQL. Ironically, this may cause those with *more* database development experience to have a greater learning curve with certain FileMaker concepts.

The following comparison provides an overview sketch of the situation and may help any developer new to FileMaker better understand the integrated nature of the application and the database file structure it is used to create.

The Traditional Multitier Database Architecture

Most databases are built using a multitier architecture. For database software engineers, a *multitier architecture*, illustrated in Figure 1-4, is an arrangement in which three functions — presentation front-end interface, process logic middleware, and back-end data — are usually physically separated and often built with different languages.

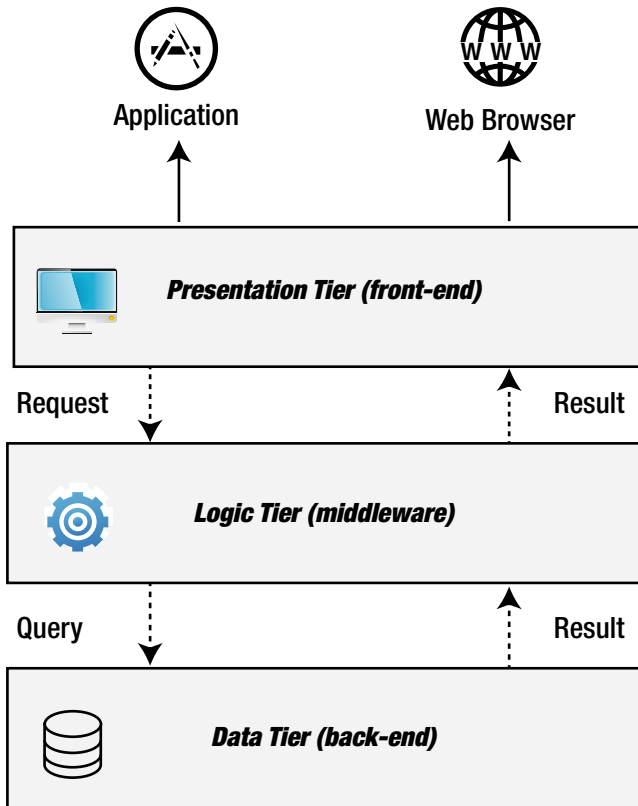


Figure 1-4. Showing an overview of the traditional multitiered database architecture with data, interface, and script as separate but interacting systems

Presentation Tier (Front End)

The *Presentation Tier* is a front-end interface. This top layer is where information is displayed in human readable form specifically for user interaction; data is viewed and edited on layouts and scripted procedures are accessible through menus and buttons.

Examples of a presentation tier include the following:

- A web page that includes database elements.
- A desktop or mobile application that presents data via windows and layouts.

Process Logic Tier (Middleware)

The *Process Logic Tier* is the middle layer that contains business logic: code that does much of the work to process commands, make decisions, evaluate requests, format results, and generally facilitate processes between the interface and the stored data.

Examples of the logic tier include these:

- PHP
- Java

Stored Data Tier (Back End)

The *Stored Data Tier* is a back-end data system. This bottom layer is generally hidden out of sight, which handles information storage and retrieval from a database or file system.

Examples of the stored data tier include these:

- Oracle databases
- MySQL database

FileMaker’s Integrated File Architecture

Unlike the distributed, multitiered design, FileMaker is an *integrated file-based architecture*, illustrated in *Figure 1-5*. An *integrated architecture* is one in which all components that make up a database are built and stored together into a single file or system that is accessed all at once.

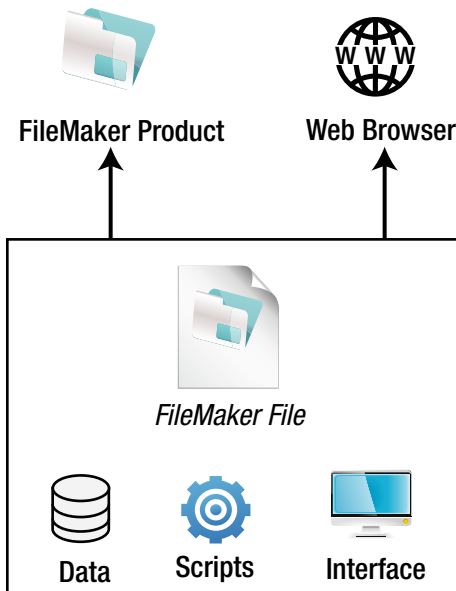


Figure 1-5. An overview of FileMaker’s integrated database architecture with data, interface, and script combined in a single file

While FileMaker’s integrated file structure is most likely inherited from its early development as a simple, end-user, file-building tool — this characteristic has been retained as it has evolved into the modern powerhouse it is today. Newer features that allow external data sources, including SQL tables, to be pulled into a FileMaker database and used as if they were internal, result in a flexibility that provides the best of both worlds.

Some benefits of this approach include the following:

- Develop the entire database system from within a single, easy-to-use application.
- Databases are files and can be hosted from a central server or a desktop, or used and shared like self-contained, portable packages just like any other document on a computer.
- Since a database can include and use external data sources, it is possible to approximate a multitiered structure. For example, some developers store data tables in a file separate from the user interface and scripted processes.

Taking a Bird’s Eye Anatomical and Functional Overview

FileMaker’s unique integration doesn’t stop at the file structure. Each version of the desktop application — FileMaker Pro and FileMaker Pro Advanced — provides a slight variation on a single, complex but remarkably intuitive and uncluttered interface that ingeniously blends all functions for the development *and* use of feature-rich database files, including the following:

- Developing back-end data structures
- Developing front-end interfaces
- Developing scripted processes
- Providing front-end user access to the database

This seemingly impossible feat is accomplished by the skillful use of different window modes for specific functions, context-sensitive menus, and separate windows for certain development functions. This seamless and almost artfully designed environment obscures an enormous complexity, which can present a challenge for both new users and authors of books such as this one. Therefore, a bird’s eye overview of the many components involved and topics to cover herein may be beneficial.

The application resources and defined file components that work together to organize, store, and manage information are shown in Figure 1-6. Although there are points of overlap, the components can be roughly divided into five categories, split between the application and the database file.

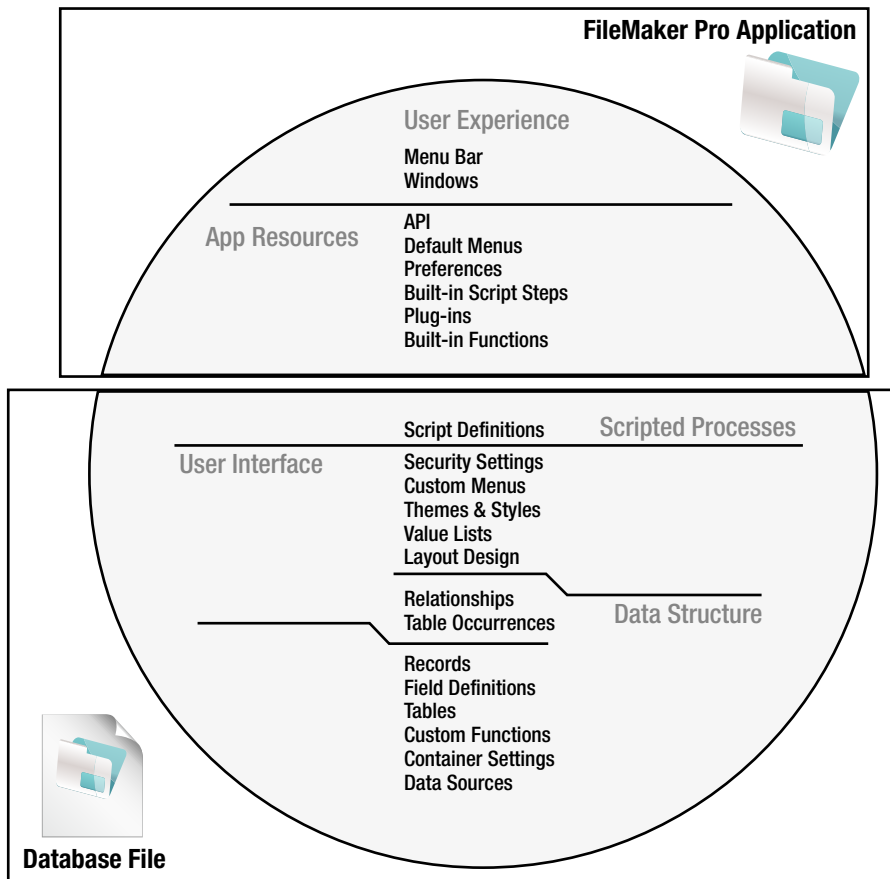


Figure 1-6. The component breakdown between the FileMaker application and a database file, grouped by primary role

The five major categories of components are these:

Application

- Application resources
- User experience

Database File

- Data structure
- User interface
- Scripted processes

Application Resources

The FileMaker application contains basic resources that interpret settings and information in a database file to generate a user experience.

Application Resources

Application resources are those things that make up the application programming interface (API) used to develop custom databases and create the development-user and end-user experience, including the following:

- Preferences (Chapter 2)
- Plug-ins (Chapter 38)
- Default menus (Chapter 2)
- Built-in functions (Part 3 — or — Chapters 13-24)
- Built-in script steps (Chapters 34 and 35)
- Sharing and networking (Chapters 39-41)

User Experience

The *user experience* covers what the user sees and how they interact with a database in general use. This includes the menus (default or custom) and the windows that the application presents to the user in the interface. The user can then take actions, including these:

- Viewing an interacting with database windows (Chapter 3)
- Creating, opening, and navigating a database file (Chapter 3)
- Working with records (Chapter 4)
- Importing and exporting data (Chapter 5)

Database File Resources

A *database file* contains everything unique to your project: all the settings for the data structure, user interface and scripted processes, as well as the records of data entered to date.

Data Structure

The *data structure* portion of a database contains all the settings that define what is stored in the file, how it connects, and what functions can be performed on it, and the data itself, including the following:

- Tables (Chapter 7)
- Fields (Chapter 8)
- Data sources, table occurrences, and relationships (Chapter 9)
- Calculation formulas (Chapter 12)
- Custom functions (Chapter 25)

- SQL queries (Chapter 26)
- Container settings (Chapter 10)
- Value lists (Chapter 11)

User Interface

The *user interface* of a database contains all the settings that define the way things are displayed on layouts in windows, including the following:

- Layouts (Chapters 27-31)
- Themes and styles (Chapter 32)
- Custom menus (Chapter 33)
- Security settings (Chapter 40)

Scripts

The *scripts* of a database define the procedures that can be performed automatically or at a user's request. (Chapters 34-37).

The FileMaker Product Family

The FileMaker Pro 16 product family is made up of five different applications, shown in Figure 1-7.

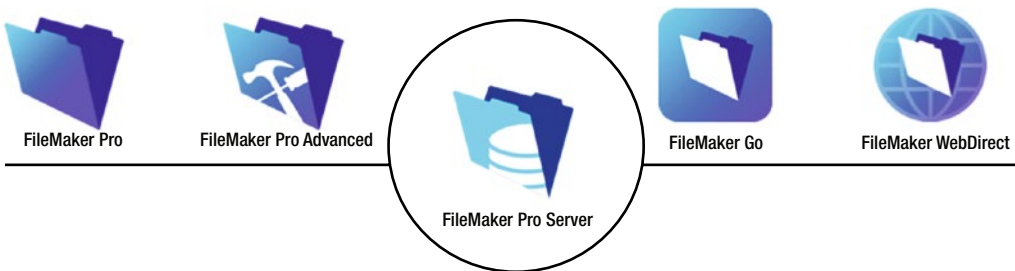


Figure 1-7. The FileMaker 16 product family lineup

Each of these applications has a unique feature set and is either purpose or device specific in its ability to develop, use, and host databases as listed in Table 1-2.

Table 1-2. The supported devices and fundamental features for each product in the FileMaker family

Product Name	Platform(s)	Develop	Use	Host
<i>FileMaker Pro</i>	Mac, Windows	Yes	Yes	Yes
<i>FileMaker Pro Advanced</i>	Mac, Windows	Yes	Yes	Yes
<i>FileMaker Server</i>	Mac, Windows	No	No	Yes
<i>FileMaker Go</i>	iOS Devices	No	Yes	No
<i>FileMaker WebDirect</i>	Web Browser	No	Yes	No

FileMaker Pro

The standard desktop application is named *FileMaker Pro*. This application provides general development features, including the following:

- *User Access* — provides intuitive front-end access to database features.
- *Import/Export* — easily exchange data between a FileMaker database and CSV, Tab, XML, ODBC, Microsoft Excel files, other FileMaker databases, and more.
- *Customize* — a graphical development environment makes it easy to create custom databases.
- *Reporting* — easily build layouts for summarizing data in reports or charts.
- *Automate* — create scripts to repeat a sequence of actions automatically.
- *Sharing* — easily share a database with other users by turning on peer-to-peer networking or uploading a database to a FileMaker Server.
- *Secure* — create accounts and passwords to limit user access and activity.
- *Integrate* — connect external SQL data sources and interact with them; create custom workflows with scripting languages that allow interaction with other applications using AppleScript (macOS) and Dynamic Data Exchange (Windows).

FileMaker Pro Advanced

The developer version of the desktop application is called *FileMaker Pro Advanced*, which has all the features of the standard version plus a handful of additional features intended for advanced developers, including the following:

- *Custom Functions* — allow the creation of your own functions.
- *Custom Menus* — allow the FileMaker menu to be overridden and replaced with custom menus and items that perform custom scripts.
- *Data Viewer* — easily monitor fields, variables, and calculations while troubleshooting your custom database.
- *Database Design Report* — generate a detailed report of all the elements that make up a database.

- *Database Encryption* — enable AES 256-bit encryption to protect a shared database.
- *Multiple Table Import* — enhanced ability to import multiple tables from one database to another.
- *Script Debugger* — helps to identify script problems and evaluate performance problems.

■ **Tip** To follow along as we explore many of the advanced subjects discussed in this book, you will want a copy of FileMaker Pro Advanced installed on your computer.

FileMaker Server

FileMaker Server is database-hosting software that is based on industry-standard security technology to offer user-scalable, secure, and reliable round-the-clock access to databases for authorized users. Its features include the following:

- *Host Databases* — dedicated server allows simultaneous, multi-user access to databases from across a network by users of FileMaker Pro, FileMaker Go, and FileMaker WebDirect.
- *Media Streaming* — container field content uses progressive downloading technology where applicable to begin streaming content for immediate display. For example, a user will not have to wait for a movie stored in a container field to fully download before it begins playing.
- *Scheduled Events* — allow various events to be scheduled for automatic execution including database backups, run system-level scripts (shell script or batch file), and run FileMaker script in a FileMaker database.
- *Perform Script on Server* — databases can opt to have specific scripts run from the server rather than on a user's computer to offload tasks and save time.
- *Logging* — administrators can access event and error logs to troubleshoot problems and to help optimize database performance.
- *Web Direct* — authorized users can access databases securely across the Web through a browser window.

■ **Note** FileMaker Cloud is another option for hosting databases on the Internet for wide area sharing. The service combines Amazon Web Services (AWS) and FileMaker Server software (Chapter 39).

FileMaker Go

FileMaker Go is a free iOS app available for both the iPad and iPhone that allows users to access database resources when away from their desktop computer. With the app, users can access databases stored locally on their iOS devices or from a FileMaker Server across a network. The app does not include an API, so databases must be developed on the desktop application prior to use on a device.

■ **Note** Accessing a database hosted on a FileMaker Server from FileMaker Go or with FileMaker WebDirect requires an available user connection license for each connection (Chapter 39: Understanding Client License Options and Limitations).

FileMaker WebDirect

Although FileMaker offers other options for web hosting, deploying a database on the Web is a lot easier with *FileMaker WebDirect*. Once a database has been built with the desktop application, it can be easily enabled for web sharing and hosted on a FileMaker Server. Users can then access the database through a web browser. Layouts are automatically rendered in the browser and, except for certain incompatible script steps and the ability to work in multiple windows, most functionality is identical to that on a desktop or iOS device.

Summary

In this chapter, we discussed an overview of FileMaker and basic database concepts. We looked at the big picture history, architecture differences, anatomical overview of components, and we reviewed the FileMaker product line.

CHAPTER 2



Exploring the Application

In this chapter, we will start exploring the FileMaker Pro Advanced application, covering the following topics:

- Installing and launching FileMaker Pro
- Default windows
- Application preferences
- Default menu bar
- Contextual menus

Installing and Launching FileMaker Pro

If you haven't already done so, take a moment to install the FileMaker Pro application. To ensure access to every developer feature and everything discussed in this book, install FileMaker Pro Advanced.

Launching the Installer

Once you have purchased and downloaded a copy of FileMaker Pro, locate and double-click on the package file, shown in Figure 2-1. This file will have a .pkg extension. On a Macintosh computer, this file should be found in a Mac_Installer folder inside of the User's downloads folder located at:

~/Downloads/

Double-clicking the package file, to open the installer wizard.

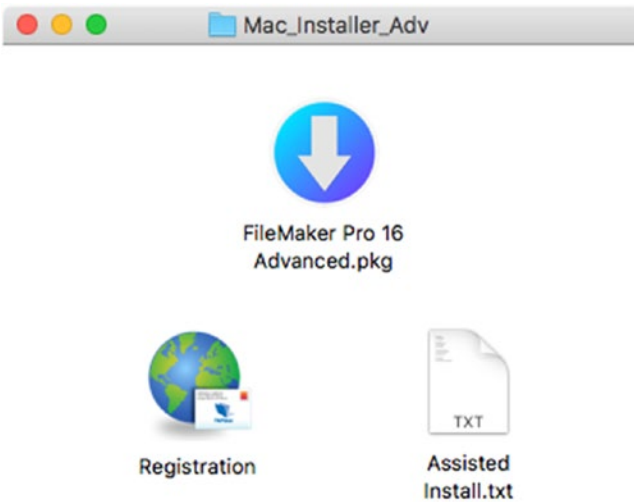


Figure 2-1. The FileMaker Pro Advanced 16 installer disk image, installer disk, and installer window

Running the Installation Process

Double-click on the FileMaker Pro 16 Advanced.pkg file. This will launch the package into the macOS Installer application, shown in Figure 2-2.

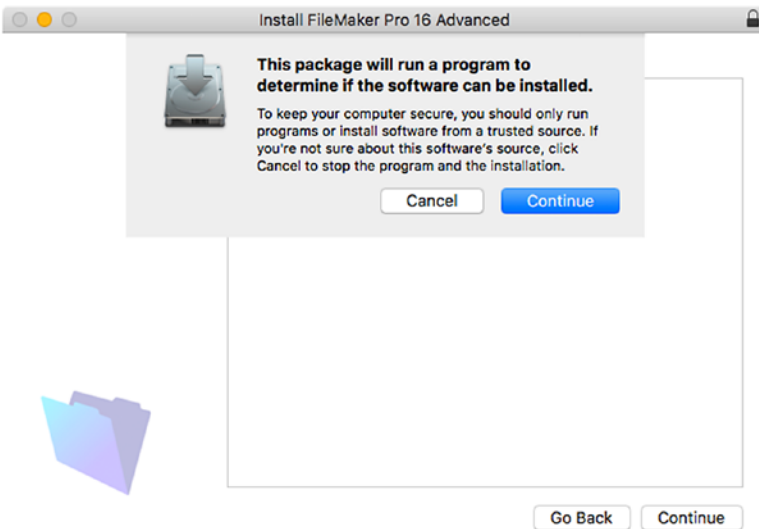


Figure 2-2. The FileMaker Pro 16 Advanced installer

To begin installing the application, click the **Continue** button on the dialog attached to the installer window and then on the **Continue** button on the window itself. Then follow these steps:

1. Enter your name, organization, and FileMaker license key, then click the **Continue** button.
2. On the software license agreement panel, click **Continue** and then click the **Agree** button to agree to the stated terms.
3. Choose the installation type and location; the default **Standard Install** options are recommended. Then click the **Install** button.
4. Authenticate as an administrative user of the computer by either using a **Touch ID** or by entering a user name and password, then click the **Install** button.
5. Fill out the registration information.
6. When the installation process is complete, a window will appear notifying you of this fact. Click the **Close** button to close and complete the process.
7. The installer should offer you a choice of what to do with the installer package. You can **Keep** the installer in the downloads folder or choose to **Move** to **Trash**.

Launching the Application

Once finished installing, locate and launch the FileMaker Pro application. If you performed the **Standard** installation, the application should be in the `/Applications/` folder. Double-click it to launch the application. It may be available from the dock and can be launched with a single click.

Default Windows

FileMaker has two default windows, one or both of which may appear when the application is launched.

Get Started Screen

When launched for the first time, the application will welcome you with a **FileMaker Get Started** screen. This scrollable window, shown in [Figure 2-3](#), summarizes several functions into a convenient window especially suited for beginners or for anyone wanting to create a database with one of the starter solutions provided by FileMaker. The options in this window include the following:

- Creating a new copy of one of FileMaker’s prebuilt basic or advanced starter solution templates.
- Creating a new custom solution.
- Converting an existing solution.
- Taking a tour of FileMaker Pro.
- Quick link to videos and other learning resources.
- Quick link to FileMaker support website.

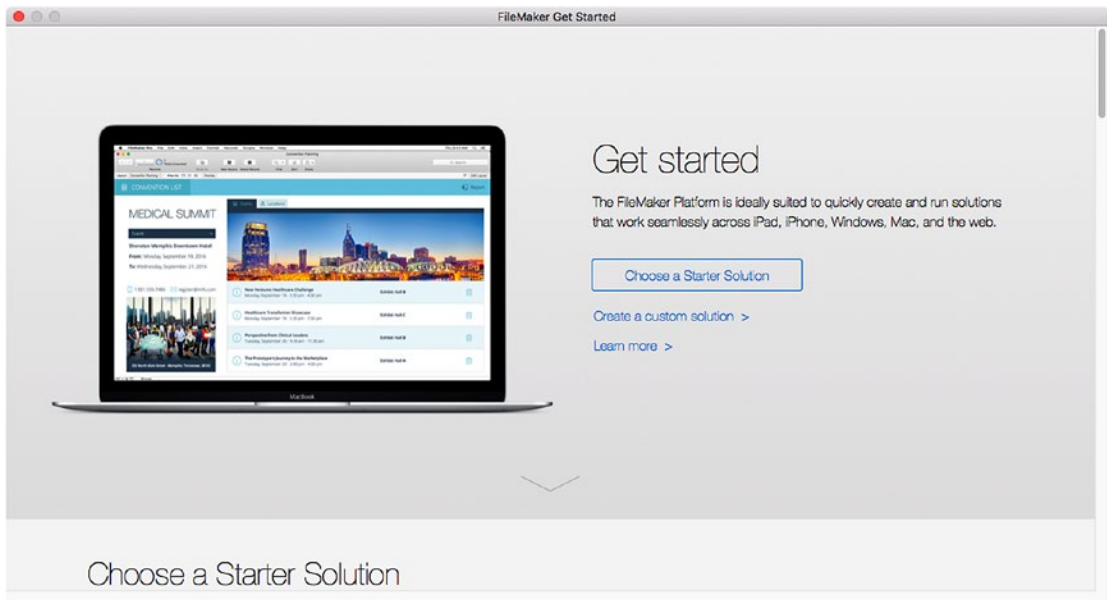


Figure 2-3. The FileMaker Get Started window

Once closed, the FileMaker Get Started window will not reappear. However, it can be manually opened by selecting the File > Get Started menu.

Launch Center

After the first launch, the default window that FileMaker will open is the Launch Center, shown in Figure 2-4.

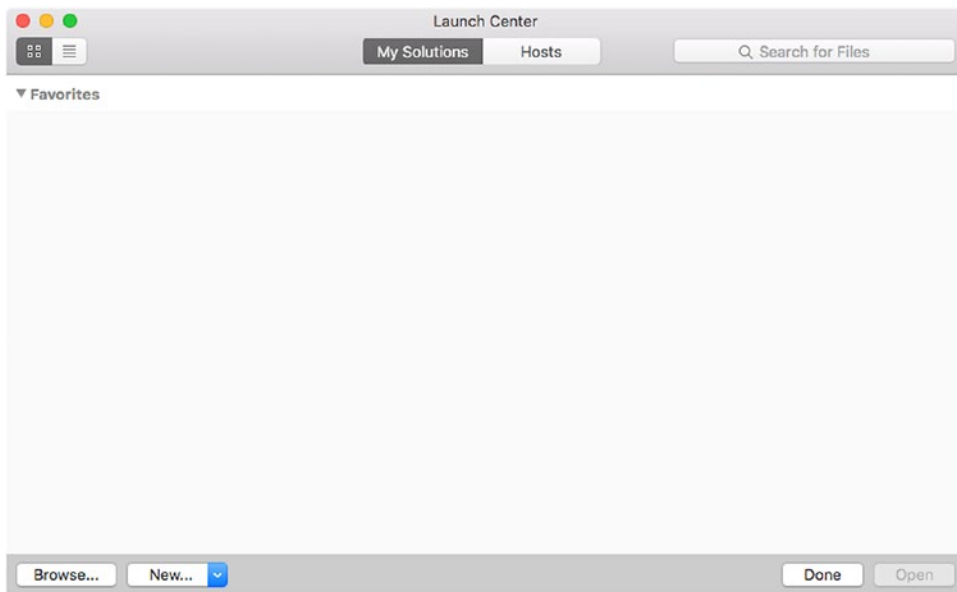


Figure 2-4. The FileMaker Launch Center window

This window provides a personalized interface for opening or creating FileMaker database files with functions including the following:

My Solutions Tab

- Favorite and recently used database files are displayed in a searchable list and are available for quick opening by double-clicking.
- The Browse button is used to locate, select, and open a database file from any directory folder or networked FileMaker Server.
- Recently opened databases can be quickly marked as favorites.
- The New button menu allows creation of a new database file.

Hosts Tab

- Favorite and local FileMaker Servers are available in a searchable list.
- The currently selected FileMaker Server's database files are available in a searchable list.
- FileMaker Server host computers can be quickly marked as favorites.

The Launch Center window will appear each time you launch the application and can be accessed manually by selecting the File > Open menu.

For more information about using the Launch Center window, see Chapter 3: Opening an Existing Database.

Application Preferences

The FileMaker application has preferences that control behaviors and allow users to personalize certain aspects of the experience on their computer. To open the preference dialog, select the applicable menu item: FileMaker Pro Advanced > Preferences (macOS) or Edit > Preferences (Windows).

Preference settings are separated into five tabs:

- General
- Layout
- Memory
- Plug-ins
- Permitted Hosts

■ **Note** Some preferences are denoted with the author's recommendation in parentheses after the description.

General Settings

The *General Preference tab*, shown in Figure 2-5, contains settings that control the user interface, user name, and notifications for updates.

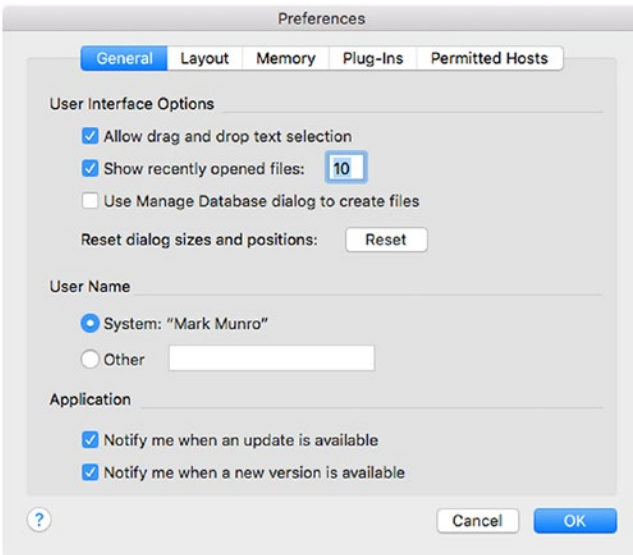


Figure 2-5. The General tab of the Preference dialog

User Interface Options

- **Allow drag-and-drop text selection** — Check this box to allow drag and drop between fields, between layouts, and between fields and content from other applications (recommended on).
- **Show recently opened files** — Check this box to enable a display of recently opened files in the File menu and the Launch Center. The text box next to it will become enabled and allow you to specify how many recent files are shown.
- **Use Manage Database dialog to create files** — Check this box to automatically open the Manage Database dialog window after creating a new file. When not selected, a new database opens in Layout mode with the Field Picker open (recommended on).
- **Reset dialog sizes and positions** — Click this button to force all dialogs to revert to their default size and position.

The following Windows only options are not shown in Figure 2-5:

- **Enlarge window contents to improve readability** — Check this box to make text easier to read with an increase to the size of layout objects.
- **User Interface Language** — A menu of available languages with a default selection of English.
- **Font smoothing style** — A menu of options to make text sharper and less fuzzy with these options. In macOS, font smoothing is controlled in the operating system preferences.

User Name

This setting identifies the name of the user. The behavior varies slightly by platform:

- In *macOS*, the user name defaults to “System,” which is set in the operating system preferences; choosing “Other” allows entry of a different user name.
- In *Windows*, a user name must be entered in the space provided.

Application

Two check boxes control whether FileMaker checks for and notifies the user when an update or new version is available:

- Notify me when an update is available.
- Notify me when a new version is available.

Layout Settings

The *Layout Preference tab*, shown in Figure 2-6, contains settings that control certain behavior when a window is in Layout mode.

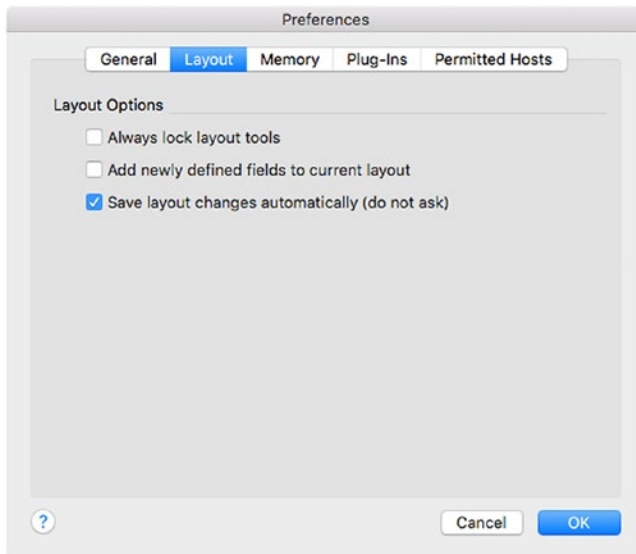


Figure 2-6. The *Layout tab* of the *Preference dialog*

Layout Options

- **Always lock layout tools** — When this box is checked, a selection of a layout tool will be locked until another layout tool is selected. The default behavior is for the tool to be released back to the cursor after creating or editing an object (recommended off).

- Add newly defined fields to current layout — When this box is checked, any newly defined field will appear at the bottom of the current layout (recommended off).
- Save layout changes automatically (do not ask) — When this box is checked, any changes made while in Layout mode will be automatically saved when switching back to browse mode. If left unchecked, a dialog will confirm whether or not changes should be saved.

Memory

The *Memory Preference tab*, shown in Figure 2-7, contains settings that control the limitations of the file cache.

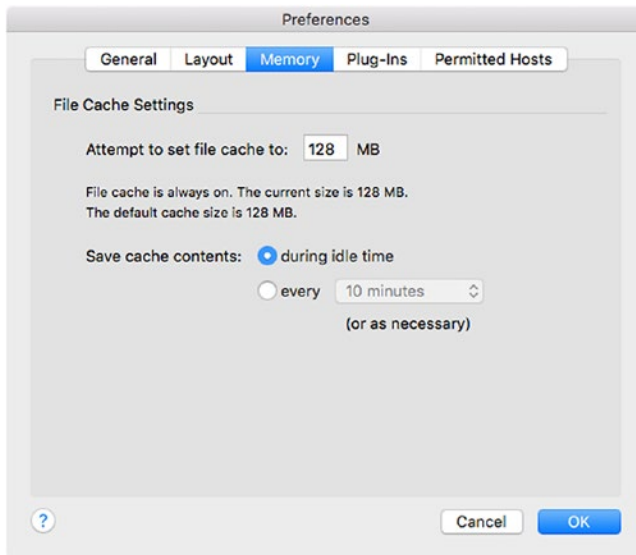


Figure 2-7. The *Memory tab* of the *Preference dialog*

File Cache Settings

As you work in a database, FileMaker stores changes in the RAM cache and periodically saves the accumulated data to the hard disk. File Cache settings control how often the cache is saved.

- Attempt to set file cache to — a number representing the amount of memory allocated to the file cache. For improved performance, set this number to a higher value.
- Save cache contents — allows control over when the cache is saved back to disk; either during idle time or at regular specified intervals.

■ **Caution** Changes to these settings take effect after you quit and re-launch the application.

Plug-Ins

The *Plug-In Preference tab*, shown in Figure 2-8, contains a list of installed plug-ins and various controls. For more information on plug-ins, see Chapter 38.

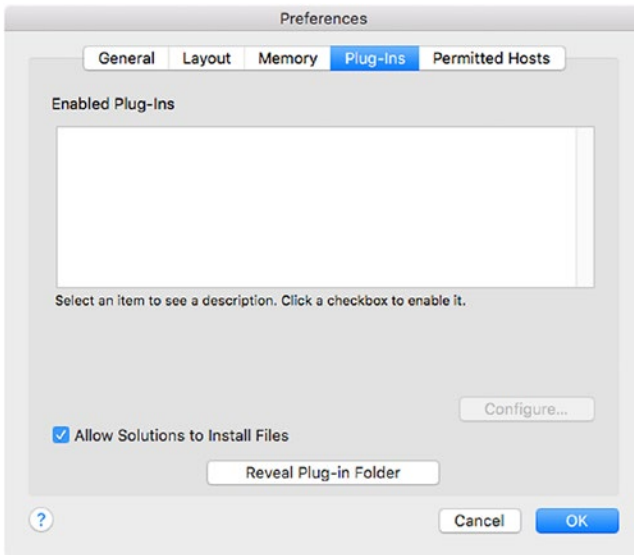


Figure 2-8. The *Plug-Ins* tab of the Preference dialog

Permitted Hosts

The *Permitted Hosts Preference tab*, shown in Figure 2-9, will contain a list of any host computers whose SSL certificates cannot be verified. When this happens, FileMaker may notify the user before connecting and, if permitted, add the host to the list shown on this tab in preferences. To delete a host, once added to the list, select it and type backspace.

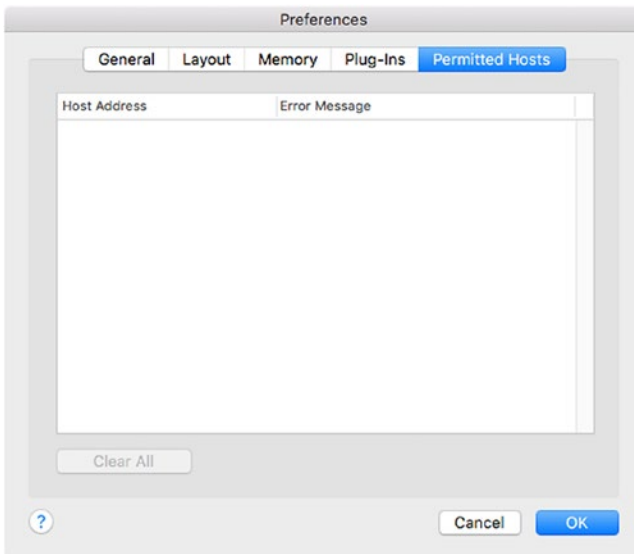


Figure 2-9. *Permitted Hosts* tab of the Preference dialog

Default Menu Bar

The FileMaker Pro default menu bar, shown in Figure 2-10, contains commands for a variety of functions.



Figure 2-10. *The Default Menu Bar*

Specific menu items may be enabled or disabled depending on various factors. Some menu items are disabled when:

- No database is open.
- A database script is running or paused.
- A data has a modal window open.
- The user's current access privileges don't allow certain functions.
- The user's current action or context doesn't allow certain functions.

Similarly, some or all menus and/or items may be completely different when:

- The Get Started screen is open showing a limited menu set.
- A database is in Find mode with a modified menu set (Chapter 4).
- A database is in Preview mode with a modified menu set (Chapter 4).
- A database is in Layout mode with a modified menu set (Chapter 27).
- A modifier key such as Option is held down.
- A database using custom menus is open.
- A developer dialog is open.

This chapter provides an overview of the default menu bar set when viewed with a database open in Browse mode with no modifier keys held down.

The Application Menu

The *Application menu*, shown in Figure 2-11, contains general application functions and some macOS functions.

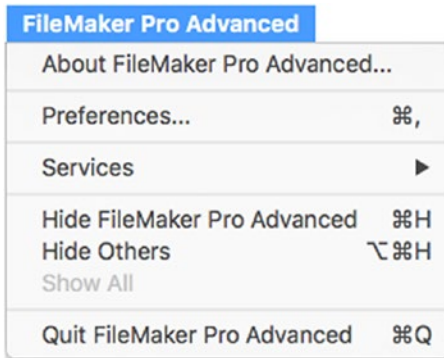


Figure 2-11. *The Application Menu*

- **About FileMaker Pro Advanced** — opens a dialog window showing the application version and other information.
- **Preferences** — opens the preferences dialog window.
- **Services** — displays any macOS or third-party application services available based on the current selection.
- **Hide FileMaker Pro Advanced** — hides the FileMaker application and all open database windows.
- **Hide Others** — hides every application except FileMaker.
- **Show All** — brings all applications to the front, including FileMaker.
- **Quit FileMaker Pro Advanced** — closes all open databases and quits the application.

File Menu

The *File menu*, shown in *Figure 2-12*, contains many file and developer-related functions.

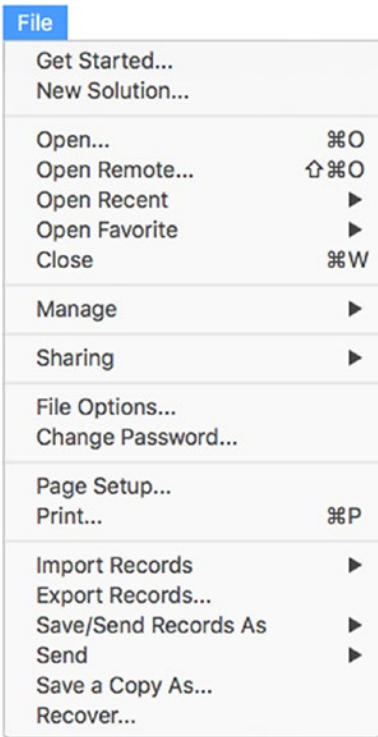


Figure 2-12. The File Menu

- **Get Started** — opens the FileMaker Get Started default window.
- **New Solution** — opens a New File dialog window, allowing creation of a new empty database file.
- **Open** — opens the Launch Center for access to recent files, favorites.
- **Open Remote** — opens the Launch Center for quick access to hosted files.
- **Open Recent** — displays a submenu of recently open database files.
- **Open Favorite** — displays a submenu of favorite database files.
- **Close** — closes the current window.
- **Manage** — displays a submenu of developer functions (see details below).
- **Sharing** — displays a submenu of network sharing functions (see details below).
- **File Options** — opens a dialog window giving access to preferences for the current database file (see details below).

- **Change Password** — opens a dialog that allows the user to change the password setting for the current account for the current database file.
- **Page Setup** — opens a dialog for adjusting settings prior to printing.
- **Print** — opens a dialog for printing the front window.
- **Import Records** — displays a submenu of import options (Chapter 5).
- **Export Records** — opens a dialog to begin an export process (Chapter 5).
- **Save/Send Records As** — displays a submenu of options for saving records from the front window in Excel, PDF, or Snapshot link format.
- **Send** — displays a submenu allowing the option to send an email or a Snapshot link (Chapter 39; Sharing Bookmarks with Snapshot Links).
- **Save a Copy As** — opens a dialog for saving a copy of the file in various formats (see details below).
- **Recover** — opens a dialog to begin a database recovery process (Chapter 6).

File > Manage Submenu

The *Manage* submenu of the *File* menu, shown in Figure 2-13, contains most developer-related functions for the current database file.

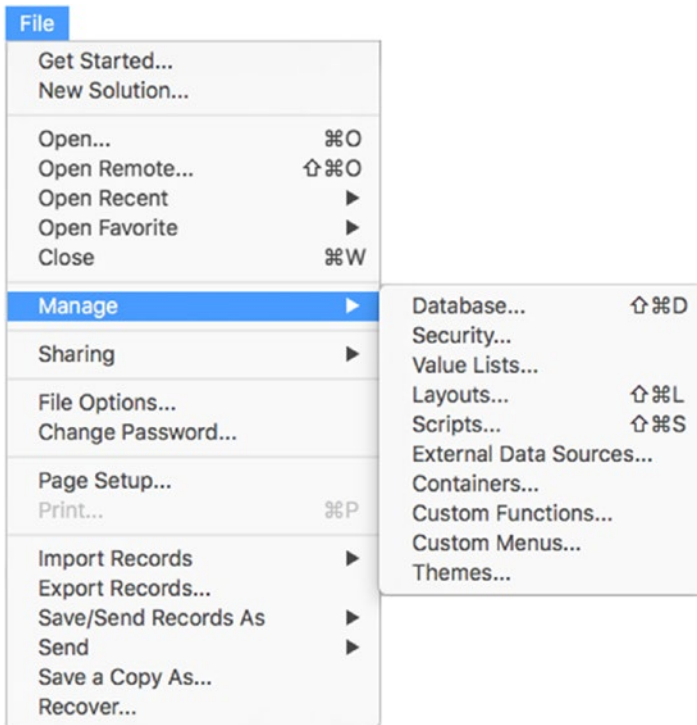


Figure 2-13. The *File > Manage Database* submenu

- Database — opens the Manage Database window, which is used to create, edit, and import tables, fields, and relationships for the current database file (Chapters 7-9).
- Security — opens the Manage Security dialog window, which is used to create and edit accounts and privilege settings for the current file (Chapter 40).
- Value Lists — opens the Manage Value Lists window, which is used to create and edit lists of values (Chapter 11) that can later be assigned to fields on layouts (Chapter 30).
- Layouts — opens the Manage Layouts dialog, which is used to create, edit and organize layouts (Chapter 28).
- Scripts — opens the Script Workspace window, which is used to manage scripts (Chapter 34).
- External Data Sources — opens the Manage External Data Sources window, which is used to create and edit connections to other FileMaker and ODBC databases (Chapter 9).
- Containers — opens the Manage Containers window, which is used to define the location and method of storing container field content (Chapter 10).
- Custom Functions — opens the Manage Custom Functions window, which is used to create and edit with custom functions (Chapter 25).
- Custom Menus — opens the Manage Custom Menus window, which is used to create and edit custom menus (Chapter 33).
- Themes — opens the Manage Themes window, which is used to manage custom themes (Chapter 32).

File > Sharing Submenu

The *Sharing submenu of the File menu*, shown in Figure 2-14, contains functions used to share the current database file on a network (Chapter 39).

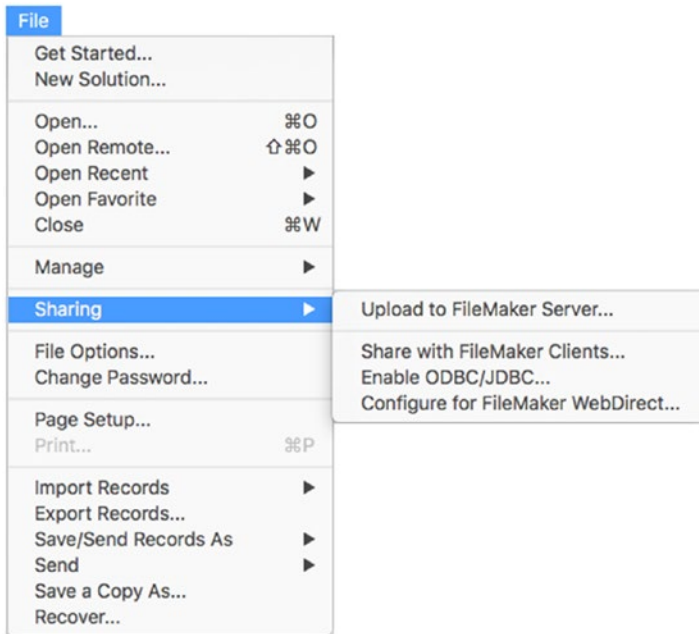


Figure 2-14. The File > Sharing submenu

- Upload to FileMaker Server — opens a dialog used to upload a local database to a FileMaker Server for hosting.
- Share with FileMaker Clients — opens a dialog used to configure a database for peer-to-peer sharing or hosting on a FileMaker Server.
- Enable ODBC/JDBC — opens a dialog used to configure a database for sharing using these protocols.
- Configure for FileMaker WebDirect — opens a dialog used to configure a database for sharing on the Web.

Edit Menu

The *Edit menu*, shown in *Figure 2-15*, contains a suite of typical editing features that are enabled when editing the contents of a field.

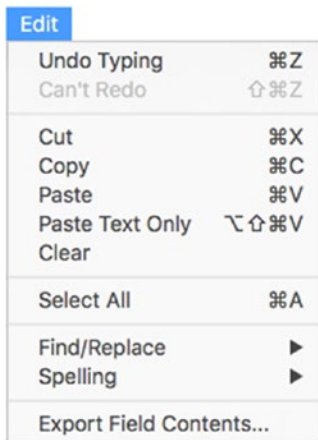


Figure 2-15. The *Edit menu*

- **Undo** — reverts the contents of the field back one step.
- **Redo** — reapplies reverted change change to contents of a field.
- **Cut** — puts the selected contents of a field into the clipboard and removes it from the field.
- **Copy** — copies the selected contents of a field into the clipboard and leaves it in the field.
- **Paste** — inserts the contents of the clipboard at the selection point of the current field.
- **Paste Text Only** — inserts the contents of the clipboard at the selection point of the current field as plain text.
- **Clear** — deletes the selected text from the current field.
- **Select All** — sets the selection to the entire contents of the current field.
- **Find/Replace** — displays a submenu of features to perform a find and replace of text values within a field (Chapter 4).
- **Spelling** — displays a submenu of functions related to spell checking the contents of one or more fields (Chapter 4).
- **Export Field Contents** — opens a dialog used to specify a name and location of a file in which to save the contents of the current field.

View Menu

The *View menu*, shown in *Figure 2-16*, contains functions that control the view of the front database window, many of which are explained in *Chapter 4*.

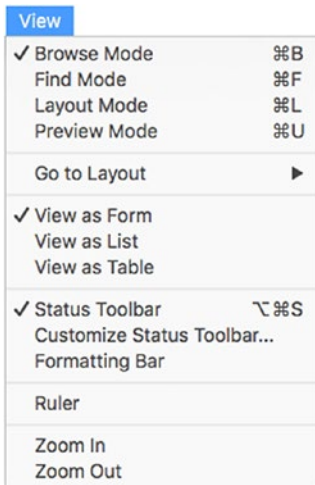


Figure 2-16. *The View Menu*

- **Browse Mode** — switches the mode of the current window to regular database usage: browsing and editing records.
- **Find Mode** — switches the mode of the current window to begin a search process.
- **Layout Mode** — switches the mode of the current window for layout editing (*Chapter 27*).
- **Preview Mode** — switches the mode of the current window to display a print preview of the current record(s) as formatted on the current layout.
- **Go to Layout** — displays a submenu of available layouts for the front database to which the user can navigate with a click.
- **View As Form** — switches the current window view to form view.
- **View As List** — switches the current window view to list view.
- **View As Table** — switches the current window view to table view.
- **Status Toolbar** — toggles the visibility of the status toolbar at the top of the current database window.
- **Customize Status Toolbar** — opens a panel attached to the current window with toolbar customization options (*Chapter 3*).
- **Formatting Bar** — toggles the visibility of the formatting bar between the toolbar and the content area of the current database window.

- Ruler — toggles the visibility of a ruler along the top of the content area of the current window.
- Zoom In — increases the magnification of the current window by 50%.
- Zoom Out — decreases the magnification of the current window by 50%.

Insert Menu

The *Insert menu*, shown in [Figure 2-17](#), contains contextually enabled functions for inserting content into the currently selected field in the current window (Chapter 4, “Insert Functions” section).

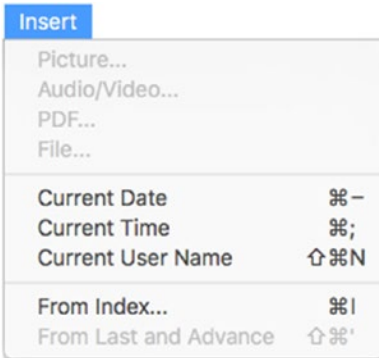


Figure 2-17. *The Insert Menu*

Format Menu

The *Format menu*, shown in [Figure 2-18](#), contains standard text styling functions.

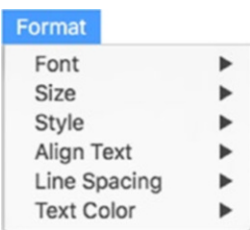


Figure 2-18. *The Format menu*

Records Menu

The *Records menu*, shown in [Figure 2-19](#), contains record functions, many of which are explained in [Chapter 4](#).

Records	
New Record	⌘N
Duplicate Record	⌘D
Delete Record...	⌘E
Delete All Records...	
Go to Record	▶
Refresh Window	⇧⌘R
Show All Records	⌘J
Show Omitted Only	
Omit Record	⌘T
Omit Multiple...	⇧⌘T
Modify Last Find	⌘R
Saved Finds	▶
Sort Records...	⌘S
Unsort	
Replace Field Contents...	⌘=
Relookup Field Contents	
Revert Record...	

Figure 2-19. *The Records menu*

- **New Record** — creates a new empty record in the front window.
- **Duplicate Record** — creates a duplicate of the current record in the front window, with a confirmation dialog.
- **Delete Record** — deletes the current record in the front window, with a warning dialog.
- **Delete All Records** — deletes all records in the found set in the front window, with a warning dialog.
- **Go to Record** — displays a submenu of options to navigate to the next or previous record or to enter a number that identifies the desired record to navigate to within the found set.
- **Refresh Window** — refreshes the front window, redrawing any layout elements that require updating.
- **Show All Records** — converts the found set to include all records in the table of the current window.
- **Show Omitted Only** — converts the found set to display only those records that are not part of the current found set.

- **Omit Record** — excludes the current record from the found set in the current window.
- **Omit Multiple** — displays a dialog allowing the entry of a number indicating how many records to exclude from the current found set, starting from the current record.
- **Modify Last Find** — enters find mode with the last find criteria re-created.
- **Saved Finds** — displays a submenu containing commands to save the current find, edit saved finds or perform a recent or saved find.
- **Sort Records** — opens a dialog allowing the user to specify a sort order for the records in the found set of the current window.
- **Unsort** — reverts the sort order of the records in the found set of the current window to their creation order.
- **Replace Field Contents** — opens a dialog allowing the user to specify criteria to change the value in the current field for every record within the found set.
- **Relookup Field Contents** — refreshes the current field's contents for every record in the found set based on its lookup settings in field definitions (Chapter 8).
- **Revert Record** — closes the current record without saving any changes.

Scripts Menu

The *Scripts menu*, shown in *Figure 2-20*, provides access to the Script Workspace window and displays a list of individual scripts that have been configured to appear here (Chapter 34).

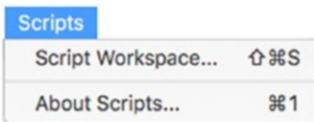


Figure 2-20. The *Scripts menu*

Tools Menu

The *Tools menu*, shown in *Figure 2-21*, is only available in FileMaker Pro Advanced and contains the following commands:

- Script debugging features (Chapter 36)
- Access to the data viewer window (Chapter 36)
- Custom menu setup (Chapter 33)
- Database design report (Chapter 41)
- Developer utilities (Chapter 34)

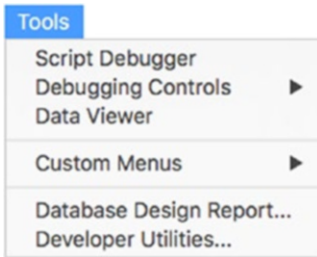


Figure 2-21. The *Tools* menu

Window Menu

The *Window* menu, shown in [Figure 2-22](#), contains many window-related functions. For more information, see [Chapter 3](#).

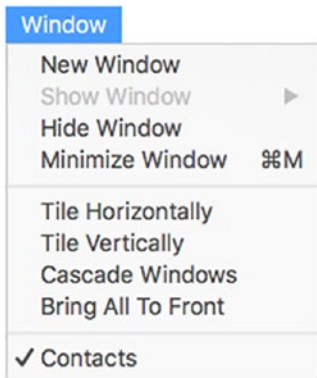


Figure 2-22. The *Window* menu

- **New Window** — opens a new window defaulting to the same size, layout and found set as the current window.
- **Show Window** — displays a submenu of all windows that are currently hidden that can be brought to the front with a click; the selected window will be repositioned on screen if it is hidden off-screen.
- **Hide Window** — hides the current window.
- **Minimize Window** — minimizes the current window into the macOS dock.
- **Tile Horizontally** — repositions all open windows horizontally across the screen.
- **Tile Vertically** — repositions all open windows vertically across the screen.
- **Cascade Windows** — repositions all open windows in a cascade, one overlapping the next.
- **Bring All To Front** — brings all windows to the front, although some may remain off-screen.

Help Menu

The *Help menu*, shown in [Figure 2-23](#), contains help and other application related functions.

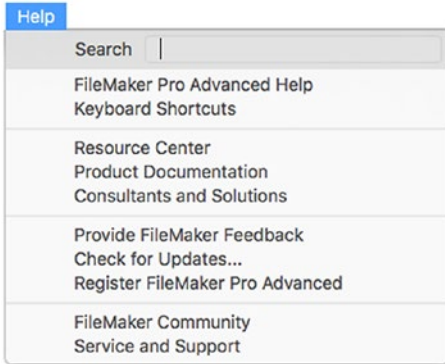


Figure 2-23. *The Help menu*

Contextual Menus

A *contextual menu* is a pop-up menu that appears within a specific context of a graphical user interface when the user right-clicks the cursor on an area possessing one, like the one shown in [Figure 2-24](#). These menus can increase the speed of accomplishing certain tasks. A user can quickly locate and trigger the command they are looking for directly from the object or location upon which the command will act, much faster than locating the command in the more cluttered menu bar.

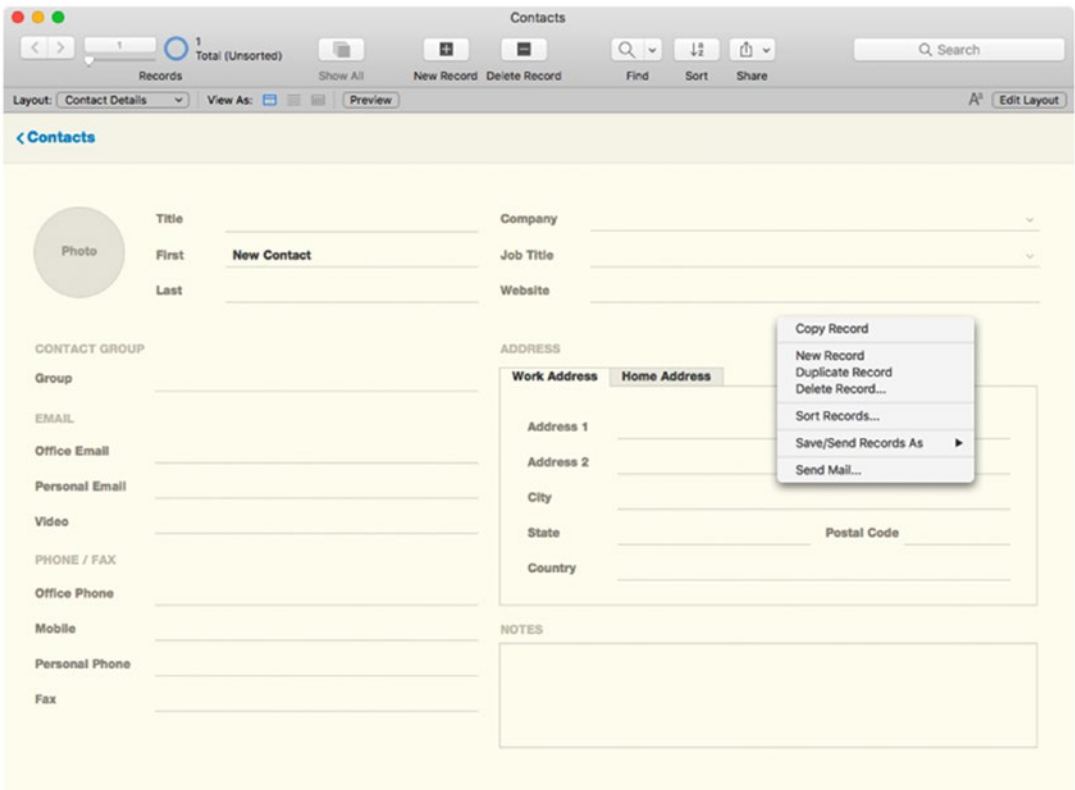


Figure 2-24. A window with a contextual menu visible

FileMaker has numerous different contextual menus, summarized here into four categories:

- Fields
- Record content area
- Web viewer
- Window

Contextual Menus for Fields

There are six different contextual menus for fields, depending on type of the field or current mode of the window:

- **Text-Based Field (Browse Mode)** — contains functions for cut-copy-paste, formatting, inserting content, sorting or searching for records by that field, chart creation, and exporting the contents of the field.
- **Text-Based Field (Find Mode)** — This contains functions for cut-copy-paste and insert operator.
- **Container Field (Browse Mode)** — contains functions for cut-copy-paste, insert file, and exporting the contents of the field.
- **Summary Field (Browse Mode)** — contains functions for cut-copy-paste, formatting, chart creation, and exporting the contents of the field.
- **Toolbar > Quick Search Field (Browse Mode)** — contains functions for cut-copy-paste, spell and grammar check, text substitutions, and transformations and speech functions.
- **Any Field (Layout Mode)** — contains functions for cut-copy-paste, text orientation, object style, specifying the field, opening button setup, opening the Conditional Formatting dialog, opening the Script Triggers dialog, formatting and arranging.

Contextual Menus for Record Content Area

There are four different contextual menus for the content area of a window where record data or layout components are displayed, depending on the window mode:

- **Browse Mode** — contains functions for copying, creating, duplicating, deleting, sorting, and saving/sending records.
- **Find Mode** — contains functions for copying, adding, and deleting find requests.
- **Preview Mode** — contains functions for copying the page, page setup, and printing.
- **Layout Mode** — contains functions for pasting, setting the style of the background, and opening the Change Theme, Layout Setup, or Part Setup dialogs.

Contextual Menus for Web Viewers

There are two contextual menus for Web Viewers depending on window mode:

- **Browse Mode** — contains a single function to reload the displayed content.
- **Layout Mode** — contains functions for cut-copy-paste, setting the style of the object, button setup, configuration of the viewer, opening the Conditional Formatting dialog, opening the Script Triggers dialog, formatting and arranging.

Contextual Menus for Window Components

There are three contextual menus for a window component:

- **Layout Part** — contains functions for opening the **Part Definition** dialog and setting the style or fill color of the part.
- **Ruler** — contains functions for switching the units to points, inches, or centimeters.
- **Toolbar** — contains functions for opening the **Customize Toolbar** dialog; switching the toolbar icon, text, or combination mode; or hiding the toolbar completely.

Contextual Menus for Calculation Formulas

There is one contextual menu available in every instance of the **Specify Calculation** dialog:

- **Specify Calculation Formula** — contains functions for manipulating text in various ways.

Summary

In this chapter, we discussed the basics of the FileMaker Pro application, including its preferences and default Browse mode menus.

CHAPTER 3



Exploring a Database Window

Over the next three chapters, we will focus on the *user experience* of working in a database file. This will provide a familiarity with interface and general FileMaker concepts. Readers already familiar with the basic appearance and behaviors of FileMaker may skip ahead but may find a review of these topics useful.

For this process, we will create a database from a starter solution provided by FileMaker. It is important to note that the window of the solution we create contains both default window resources and custom sample database layouts. Although that solution is created by FileMaker, Inc. and does provide a nice example for this orientation tour, it is not necessarily indicative of how a custom solution will or should be designed. In later chapters, we will create a new database and discuss how to develop a custom solution with design custom layouts.

In this chapter, we will discuss topics such as the following:

- Using a starter solution
- Defining the database window
- Exploring the window header
- Managing multiple windows
- Opening an existing database

Using a Starter Solution

A *starter solution* is a template for a common type of database, provided by FileMaker Inc., intended as an example of basic database structures and functions or for use as a starting point upon which to build out a custom solution.

Listing FileMaker's Starter Solutions

FileMaker provides both basic and advanced starter solutions:

Simple solutions

- Contacts
- Inventory
- Content management
- Tasks

Advanced solutions

- Assets
- Contacts
- Content management
- Estimates
- Event management
- Expense report
- Inventory
- Invoices
- Meetings
- Personnel records
- Product catalog
- Projects
- Research notes
- Resource scheduling
- Tasks
- Time billing

Creating a Database from a Starter Solution

To focus on the basics of user interface concepts, we will use the basic Contacts template. You can freely explore the details of the advanced templates at your leisure.

To create a copy of the Contacts template, follow these steps:

1. Select the File > Get Started menu.
2. In the Get Started window, click on the Choose a Starter Solution button; this will scroll you down to the start of the list of available starter solutions starting with Contacts.
3. Click the Create from This Starter Solution button.

A copy of the starter solution should appear on your desktop and then automatically open in a window as shown in Figure 3-1.

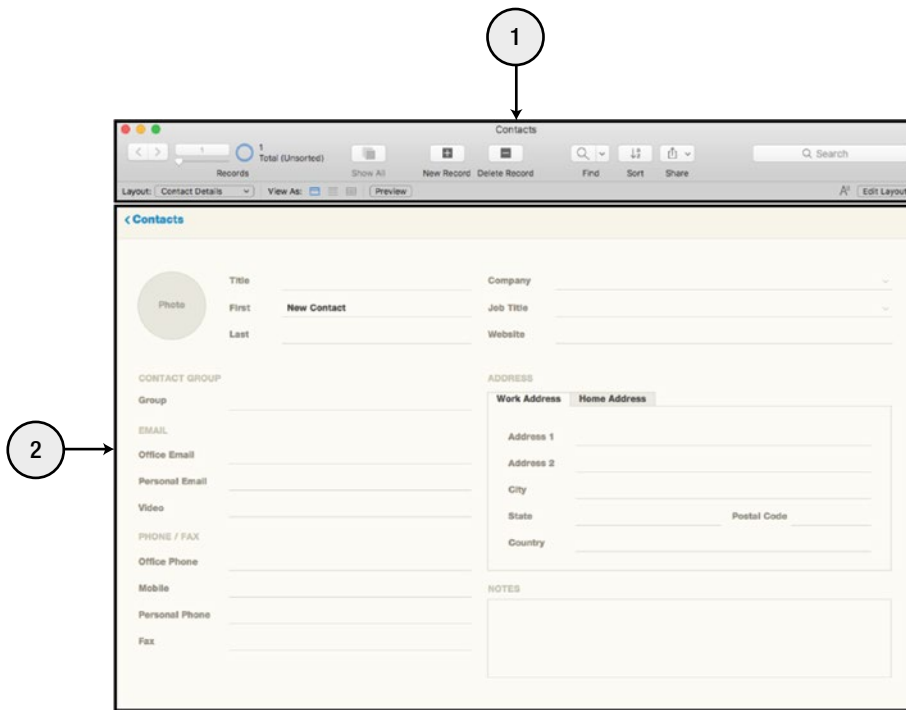


Figure 3-1. The default window of the new database created from the Contacts starter solution template

Defining the Database Window

A *database window* is where records are created, edited, and viewed. A window provides a single perspective on a layout representing a table of data. A user can navigate to other records within that table or view other layouts for the same or for other tables, one at a time.

Identifying Window Areas

A database window can be broken down into two general areas:

1. *Status Toolbar* — running along the top of the window, this area contains navigation, buttons and controls for managing records, switching modes, and changing between available views. This area is the same for every layout in a file (depending on the window mode) but can be customized by the user on their computer. It can be hidden by the user or hidden and/or locked with a script.
2. *Content Area* — the bottom, expandable area of the window contains the user interface for the current layout as designed by a developer. In this example, the window is preconfigured with interface elements for the contacts table as designed by FileMaker Inc.

Defining Window Modes

A *window mode* is a state in which everything displayed within the entire window is optimized for a functional purpose. FileMaker has four possible window modes:

- Browse mode
- Find mode
- Preview mode
- Layout mode

A user can change the mode of a window by:

- Clicking a mode button in the status toolbar.
- Selecting a different mode from the View menu or typing the equivalent key command.
- Running a script or clicking a button that uses an Enter <mode> script step such as Enter Browse Mode or Enter Find Mode.

Each mode changes various elements of the window including the following:

- Status toolbar control set
- Content area appearance and function
- Menu availability and functionality

Browse Mode

Browse mode, the default and most common window state, is used to view and perform data entry interactions with records. In this mode, the content area of the window represents the actual data stored in the current layout's table. A user can create, view, edit, delete, duplicate, sort, and omit records, click buttons and other controls, and run scripts. For more information on working with records, see Chapter 4.

Find Mode

Find mode is a window state in which the layout transforms to display a blank record where the user enters criteria used to search for a desired set of records. In this mode, a record is considered a request. Users can interact with requests just as they do with records, but requests only exist temporarily as a means of collecting the criteria prior to performing the search. For more information on searching for records, see Chapter 4.

Preview Mode

Preview mode is a window state that displays the current layout as a preview of how it will appear on a printed page. The layout will look generally the same except that fields will not be editable, and objects may slide and compress, summarize, and change formatting depending on the layout settings. For more information on previewing and printing records, see Chapter 4.

Layout Mode

Layout mode is a window state that displays the current layout in a design mode. This allows users with development access to edit which fields and controls will appear in Browse, Find, and Preview modes and configure their formatting and functional settings. For more information on designing layouts, see Chapters 27-31.

Defining Content Views

A *content view* is a view format that determines how records are displayed in the content area of a Window. A user can select to view records as one of three formats depending on the layout settings:

- Form view
- List view
- Table view

The content view of a window can be changed by:

- Selecting the corresponding menu item under the View menu.
- Clicking the corresponding View As icon in the toolbar.
- Running a script or clicking a button that uses the View As script step.

Form View

The *Form view* format displays records one record at a time as a form for data entry. When in this view, the user must navigate forward or backward using the navigation controls in the toolbar to see other records instead of scrolling through a list within the window frame.

List View

The *List view* format displays all the records in the current found set as a continuous list, repeating the layout elements once for each record. When in this view, the user can scroll up or down to see other records.

Table View

The *Table view* format displays all the records in the current found set in a spreadsheet-style arrangement of columns and rows with no rich interface formatting.

■ **Note** For additional information about Content Views see Chapter 28, “Planning Layouts”.

Exploring the Window Header

There are three horizontal bars that can be displayed or hidden depending on the user’s access privileges, preferences, or the settings established by the developer. These are:

- Status toolbar
- Formatting bar
- Ruler

Status Toolbar (Browse Mode)

The *status toolbar* is an area running along the top of the window containing controls pertinent to the current window mode.

Unless locked by a script, the visibility of the toolbar can be toggled by:

- Selecting the View > Status Toolbar menu item.
- Running a script or clicking a button that uses the Show/Hide Toolbars script step.

For each window mode, the toolbar options can be either the default set for that mode or a user-customized set that applies to all databases on the user’s computer. The default set of toolbar options for Browse mode is discussed in this chapter. Find and Preview modes are discussed in Chapter 4. Layout mode is discussed in Chapter 27.

■ Note Every toolbar control is connected to basic FileMaker commands and is automatically enabled or disabled based on the respective command’s representation and enabled in the menu bar. See Chapter 33, “Exploring the Link Between Commands and Menus.”

Default Status Toolbar Items (Browse Mode)

The default status toolbar for Browse mode is shown in Figure 3-2.

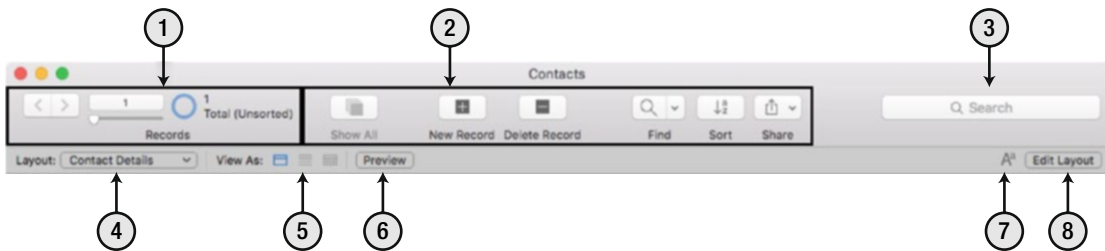


Figure 3-2. The default status toolbar of a database window in Browse mode

The Browse mode toolbar can be divided into the following areas:

1. Record navigation controls
2. Function buttons
3. Quick search field
4. Layout menu
5. View buttons

6. Preview button
7. Formatting bar button
8. Edit layout

Record Navigation Controls

At the top left of the toolbar are the navigation controls, shown in Figure 3-3.

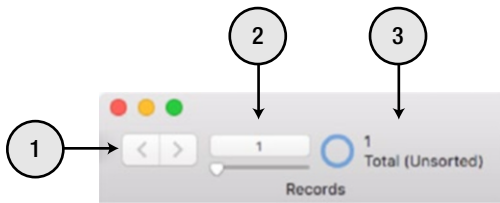


Figure 3-3. The default toolbar navigation controls in Browse mode

The available navigation controls are the following:

1. *Record Navigation Arrows* — click to navigate backward or forward, one record at a time through the current found set of records.
2. *Record Slider* — displays the number of the record currently viewed or selected within the found set. The user can move the slider or type a record number and Enter to quickly jump to another record within the found set.
3. *Found Set Status* — displays the sort status of the found set, the total number of records contained in the layout's table and the number of records in the found set when that set is less than the total records.

■ **Caution** The record navigation arrows are sometimes confused with general navigation arrows like those in a web browser. Instead of navigating backwards and forwards to different layouts, these buttons navigate to records within the found set on the current layout.

Function Buttons

The function buttons region of the toolbar, shown in Figure 3-4, includes controls to manage records and share a database.

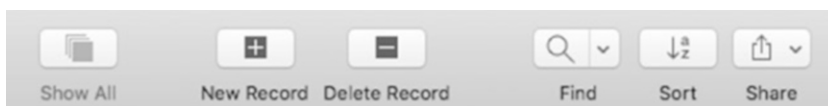


Figure 3-4. The default toolbar buttons in Browse mode

The button functions in this group are the following:

- *Show All* — expands the found set of records to include all records contained within the current table (Chapter 4).
- *New Record* — creates a new empty record within the current table (Chapter 4).
- *Delete Record* — deletes the currently viewed or selected record, after presenting a warning dialog (Chapter 4).
- *Find Menu* — starts a find process with a click or displays a menu of Find options (Chapter 4).
- *Sort*— displays a dialog allowing a new sort order for the records displayed in the current found set (Chapter 4).
- *Share Menu*— displays a menu with options to share a file from the current computer or to upload the file to a FileMaker Server (Chapter 39).

Quick Find Search Field

At the top right of the toolbar is the search text field, shown in Figure 3-5. After a user types text into the field and the Enter key, a search is performed in every field on the layout with Include field for Quick Find selected (Chapter 4, “Searching with Quick Find”).

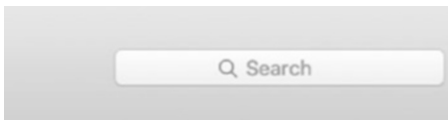


Figure 3-5. The default toolbar Quick Search field in Browse mode

Layout Menu

The lower, non-customizable level of the toolbar starts with the Layout menu on the left. In Browse mode, this pop-up menu displays a list of layouts that have the Include in Layouts Menu option selected checked (Chapter 28). When a user selects a layout from the list, the window view switches to display the selected layout.

Layout navigation is also available from the View > Go To Layout menu.

View Buttons

Next in the lower portion of the status toolbar are three buttons that will change the current window’s view to that selected. The view buttons that are enabled here are controlled by the current layout’s view settings (Chapter 28).

Preview Button

The *Preview* button will switch the window into print preview mode (Chapter 4).

Formatting Bar Button

On the right of the lower, non-customizable portion of the toolbar, the *formatting bar button* will toggle the visibility of a bar between the status toolbar and the content area of the window that contains text-formatting controls (described below).

Edit Layout Button

The *Edit Layout* button, located on the far right, will switch the window to Layout mode (Chapter 27).

Customizing the Status Toolbar (Browse Mode)

The status toolbar is customizable at the user-computer level. This means that a user can customize the contents and arrangement of objects in the toolbar, which will affect all databases they open on their computer.

Opening the Customization Panel

To begin customizing the toolbar, select the View > Customize Toolbar menu. This will open the customization panel attached to the window as shown in Figure 3-6.

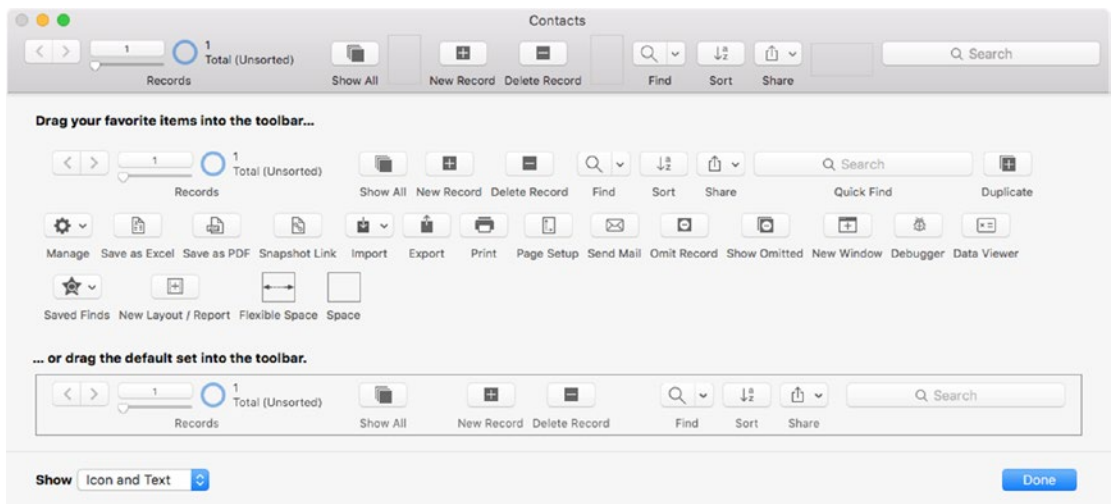


Figure 3-6. The customize toolbar panel attached to the Contacts database window in Browse mode

Choosing Toolbar Button Style

At the lower level of this dialog is a Show menu, which allows the user to choose which type of controls will be displayed in the toolbar. These are also available in a contextual menu, accessible by right-clicking on the background of the toolbar. The choices are:

- Icon and text (default)
- Icon only
- Text only

Removing Toolbar Items

To remove an item from the toolbar, follow these steps:

1. Click and hold the icon with the cursor.
2. Drag the object until it is clear of the toolbar area.
3. Release the mouse button to delete the object in a small graphical puff of smoke.

■ **Note** Don't be afraid to experiment. Later, we will review how to restore the entire toolbar to its default set of controls.

Adding Toolbar Items

To add an item to the toolbar, follow these steps:

1. Click and hold the cursor on the desired object from the customization panel.
2. Drag the object up into the toolbar at the desired location among the other objects.
3. Release the mouse button to finalize its placement in the toolbar.

To avoid duplication of items in the toolbar, FileMaker will automatically replace any object already present in the toolbar when that same control is added.

Rearranging Toolbar Items

Items can be rearranged within the toolbar by dragging them back and forth to the desired location when the customization dialog is open.

Restoring the Default Toolbar Set

To restore the toolbar to its default set of controls, drag the group in the rectangle at the bottom of the customization panel up to the toolbar. It will automatically replace everything and restore all controls to their default state.

Formatting Bar

The *formatting bar* is a thin bar that contains text-formatting controls for quick access. Not visible by default, this bar, shown in Figure 3-7, will appear between the status toolbar and the window's content area when the View > Formatting Bar menu is selected or the icon on the lower right of the toolbar is clicked.

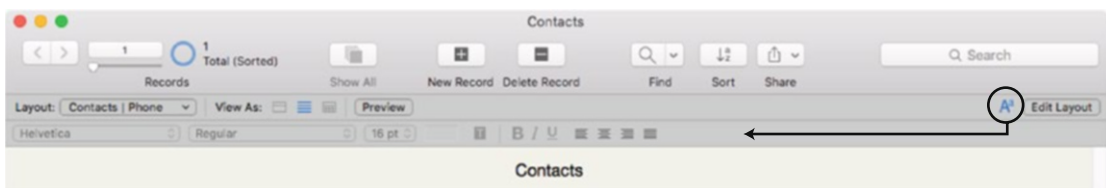


Figure 3-7. The formatting bar and toggle button

When a user is editing a text field in Browse mode, the following controls will become enabled in the formatting toolbar, providing access to the following formatting settings of a field's text:

- Font
- Font style
- Font size
- Font color
- Text alignment

When a window is in Layout mode, the formatting bar contains a few additional options for object fill color as well as line style, thickness, and color.

Window Ruler

A FileMaker window can display a horizontal ruler when in Browse mode, shown in Figure 3-8. To toggle the visibility of this ruler, select the View > Ruler menu.

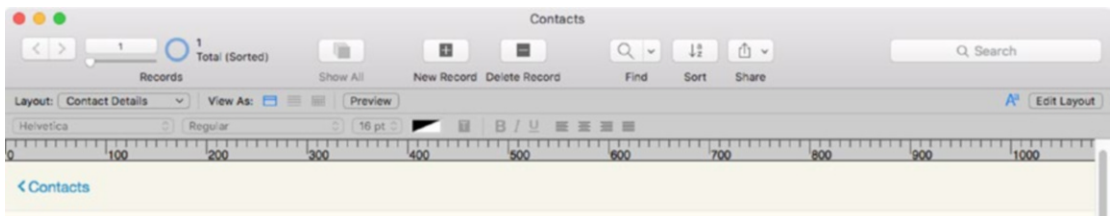


Figure 3-8. A window with the Browse mode ruler visible between the toolbar and the content area

In Browse mode, the ruler doesn't offer much value when displayed across the entire window. However, when the cursor focus is inside a text field, the ruler scales down to the size of that field, as shown in Figure 3-9, and will then display any tabs set for that field and allow margin adjustments (for that field's content for current record only).

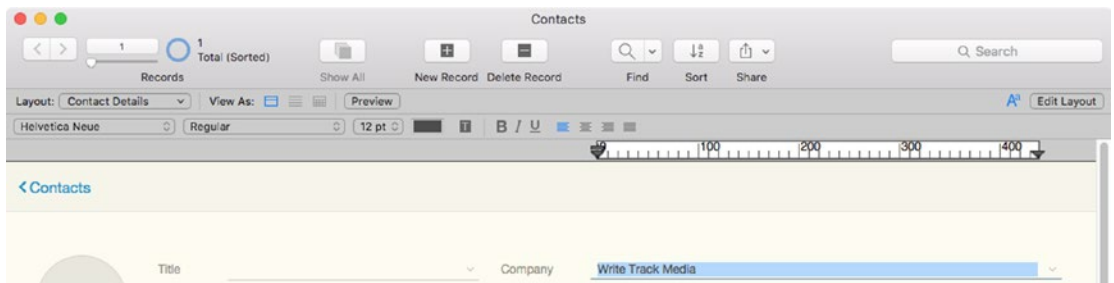


Figure 3-9. The window ruler when focus is inside of a field

When a window is in Layout mode, the ruler option displays a horizontal and vertical ruler that is useful when designing interfaces (Chapter 31).

The units can be changed within the ruler's contextual menu. The available units are:

- Inches
- Points
- Centimeters

Managing Multiple Windows

FileMaker has many window management functions that give a user some control over their interface experience when using a database.

When a database opens, a single window is automatically generated. It represents the open file and displays records for the default table. A user can then change layouts and perform other functions. They can also open new windows to gain access to multiple views into the database's content.

Using multiple windows makes it easy to do the following simultaneously:

- View different field sets or arrangements of fields for the same record on different layouts.
- View different records from the same table.
- View records from different tables.
- Preserve your position in a found set while pausing to perform work on another record in the same table or a record in a different table.
- View one record in different modes.

As a developer, you can take advantage of multiple windows. For example, you can edit different layouts simultaneously or use multiple windows to access separate lists of records with scripts.

Creating a New Window

To create a new window, select the Window > New Window menu. The new window will open slightly offset from the current window, as an exact duplicate with the same properties, including:

- Name (with a numeric suffix)
- Window mode
- Content view type
- Width and height
- Status toolbar visibility and locked status
- Formatting bar visibility state
- Ruler visibility state
- Layout
- Found set of records
- Current record

Once the window has appeared, the user can work within it as they could in the original window. The window can be moved and resized for side-by-side viewing. Each window has its own context so you can change layouts and work with different record sets.

Closing a Window

A window can be closed just like any document window, using any of the following methods:

- Select the File > Close Window menu item.
- Click the red dot in the upper-left corner of the window.

FileMaker requires one window to remain open as a representation of the database file. If the last remaining window for a database is closed, the file itself will be closed.

Selecting a Window from the Menu

As new windows are created, they are added to the list at the bottom of the Window menu, as shown in Figure 3-10. This list is blended, meaning that it shows all open windows, regardless of their source database in the layering order as they are currently stacked and displayed on the screen. To switch to a different window, select it from the list in the menu and it will be brought to the front.

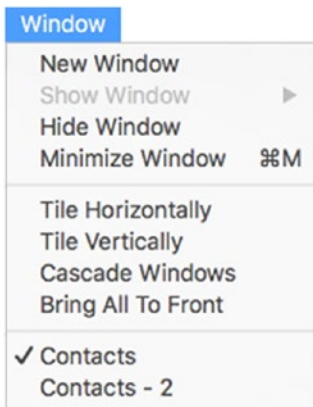


Figure 3-10. The list of open windows at the bottom of the Window menu

Hiding and Showing Windows

A window can be temporarily hidden from view by selecting the Window > Hide Window menu item or by running a script that uses the Hide Window command. When hidden, windows are listed under the Window > Show Window submenu, as shown in Figure 3-11. From here, they can be made visible and brought to the front with a click.

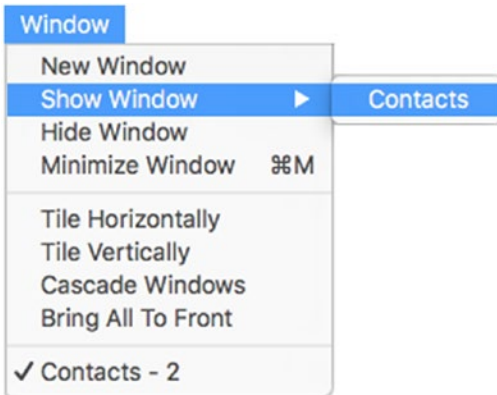


Figure 3-11. The submenu containing hidden windows

■ **Note** In some cases, a database may open with the default window immediately hidden. This can be intentionally done by a script or when a file is automatically opened in the background because its fields or scripts are used by another open file.

Minimizing a Window

A database window can be minimized into the toolbar by any of these methods:

- Select the Window > Minimize Window menu item.
- Click the orange dot in the upper-left corner of the window.

Minimizing will cause the current window to shrink down into an icon in the dock; however, FileMaker will continue to list it as an open window rather than a hidden window. Clicking the window icon in the dock or selecting it from the bottom of the Window menu will cause it to reappear at its former location and size on screen, in front of any other windows.

Opening an Existing Database

Once created and closed, a database file can be reopened in a way like any other document on your computer. The methods of opening an existing FileMaker database are these:

- Double-click on the file icon in a directory.
- Drag and drop the file on the FileMaker Pro application in the application folder or its icon in the dock.
- Select the File > Open menu item to open up the Launch Center for access to favorites, recent, and more.

- Select the File > Open Recent menu item to access a submenu of recently opened databases.
- Select the File > Open Favorite menu item to access a submenu of favorite databases.
- Select the File > Open Remote menu item to open the Launch Center Host's tab and access databases hosted on a FileMaker Server.

Opening a Database through the Launch Center

To open the Launch Center, shown in Figure 3-12, select the File > Open menu item. When the window opens, it displays a list of any recent databases opened and any files you have marked as favorites on the local computer. Click on the icon or name of the file to open any of these databases.

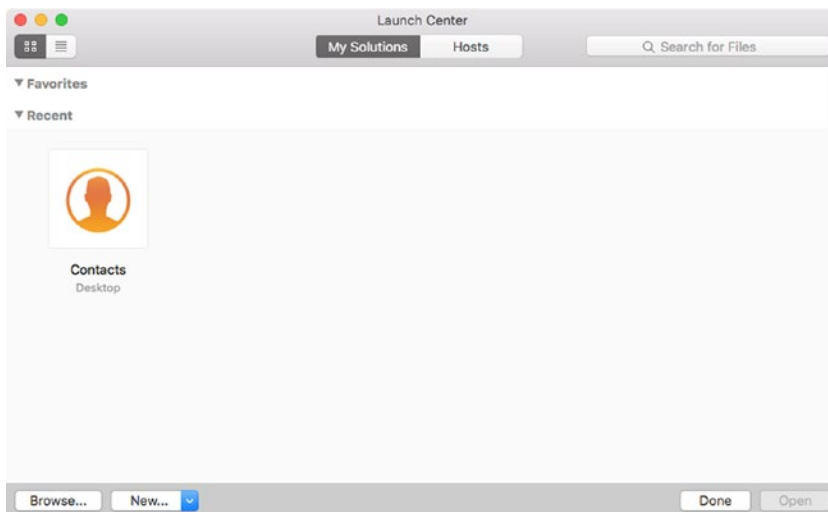


Figure 3-12. The FileMaker Pro Launch Center

To open a file not already displayed in the window, click the Browse button on the lower left to access an Open File dialog, shown in Figure 3-13. Navigate to the directory containing the desired file, select it and then click the Open button.

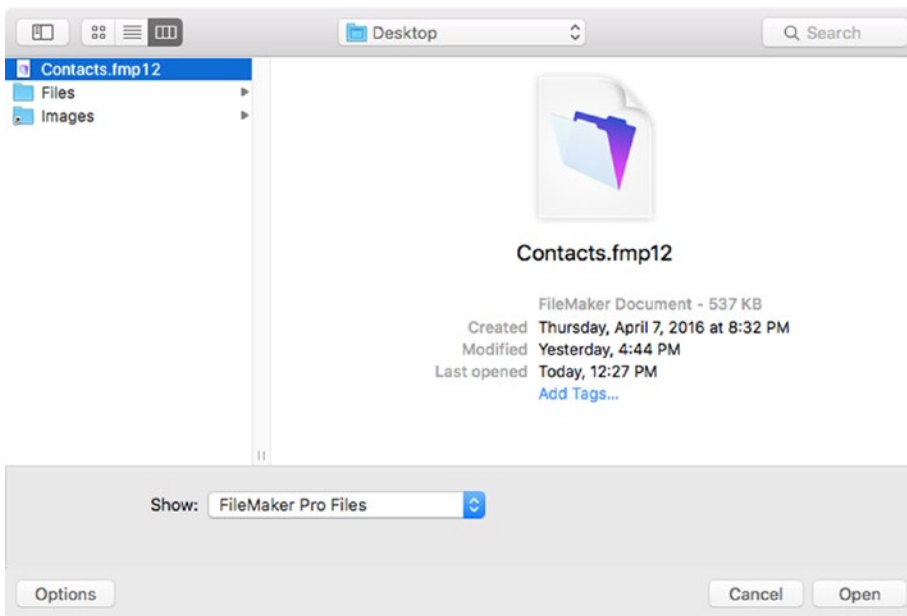


Figure 3-13. The macOS Open File dialog window

Managing Favorite Databases

Adding frequently used databases to your computer’s local favorites list is a great way to save time locating and opening a file. This can be done one of three ways:

- If the file is listed in the Launch Center’s recent list, click the star that is either to the left of the file icon (list view) or on the upper-left corner of the file icon (icon view), both appearing when the cursor is moved over the icon.
- When the file is open, click the File > Open Favorite > Add Current File to Favorites menu.
- When the file is open, click the File > Open Favorite > Manage Favorites menu to open a dialog where you can add or remove favorites from a list.

Summary

In this chapter, we created a new database from a starter solution template and discussed the basics of a FileMaker Pro window.

CHAPTER 4



Working with Records

A *record* is a collection of values that make up a single entry in a table. Similar to a row of a spreadsheet, each record contains field values that together represent the properties of one entity of the type after which a table is modeled. For example:

- In a *Contacts* table a record would represent one person.
- In a *Projects* table a record would represent one job.
- In an *Inventory* table a record would represent one product.

Most of a user's work in a database will be spent interacting directly with records, from within the content area of a window. Records can be created, edited, deleted, omitted, searched, exported, imported, printed, and viewed.

In this chapter, we will focus on how a user interacts with records, covering topics such as the following:

- Entering data
- Managing records
- Searching for records
- Working with found sets
- Printing

■ **Note** Throughout this chapter, we will continue working in the *Contacts* database we built in [Chapter 3](#) using FileMaker's starter template.

Entering Data

Data entry tasks are performed while a window is in Browse mode and a record is opened into an editable state. Data can be entered, edited, and formatted in whatever field has current focus on the current open record. Once finished, the record is closed and changes are either reverted or committed.

Opening a Record

Once a user has identified a record they want to edit, they must open it. If a user begins typing while viewing a record before opening it, an error dialog indicating this fact will appear.

Opening a record means that the record is activated into an edit state in which the fields are no longer simply displaying information, they are activated and ready for data input.

To open the current record for data entry, perform any one of these steps:

- Click into any editable field.
- Type the Tab key.
- Run a script or click a button that uses the Open Record/Request script step.

Once a record is open for data entry, it may appear like it does when viewed. However, differences in appearance between viewing and editing states will depend on the choices made by whoever developed the database. It's a good practice for developers to program some visual change to help a user clearly see that the record is opened and ready for input. For example, the field borders might become visible or change thickness or color. Unfortunately, the Contact starter solution from FileMaker doesn't provide such an indication and instead only makes clear that a single field has focus and is ready to accept input.

Understanding Field Focus

When a record is opened for editing, one field will be in focus. *Focus* means that the field is currently active and has the text cursor blinking within it, as shown in Figure 4-1. Any keyboard input will flow into the field with current focus.



Figure 4-1. A field in focus in the Contacts starter solution

Once again, other formatting changes may occur depending on the theme used or the formatting choices made by the developer of the layout.

Moving to Another Field with Tab Order

When finished editing the content of one field, the user can move on to other fields one of two ways:

- Clicking directly onto any other editable field on the layout will move focus to that field and allow the user to continue editing.
- Typing the Tab key will move the cursor to the next field based on the layout tab order. The *tab order* is a number assigned to fields by a developer while designing the user interface (Chapter 31). The number controls the cascade from one field to the next each time the user types the Tab key. This is sometimes called *tabbing* into a field, and it allows the user to maintain contact with the keyboard, avoiding the mouse for almost all data entry work. Once they reach the last field in the tab order, the sequence repeats back to the first field.

■ **Note:** A keys that allow navigating to the next object in the tab order can be customized with the Go to next object setting (Chapter 29).

Modifying Field Contents

Once a record is open and a field has focus, the value in the field can be changed in several different ways depending on the field's data type (Chapter 8) and layout behavior and formatting options (Chapters 29 and 30). These include:

- Typing
- Cut, copy, paste, undo, redo
- Drag and drop
- Insert functions
- Mouse clicks

Typing

The most basic method of entering data into any data entry field (except containers) is by typing on the keyboard once field focus has been established.

Cut, Copy, Paste, Undo, Redo

Like all standard applications that accept text input, FileMaker's Edit menu has commands for Cut, Copy, Paste, Undo, and Redo. Where applicable, these allow standard manipulations to content stored in fields and development resources such as formulas, layout objects and value lists.

■ **Caution** Undo does *not* include reversing actions such as record deletion.

Drag and Drop

If the application's Allow drag and drop text selection preference is checked, FileMaker allows the following:

- Rearranging text within a single field with drag and drop.
- Copying text from one field to another by dragging.
- Dragging content from sources in other applications and dropping it into fields.

Insert Functions

The Insert menu and Insert submenu of a field's contextual menu make available several functions for quickly inserting various content into fields such as pictures, files, and text. Each of these menu items are contextually enabled depending on the data type of the field being edited and based on the enabled status of a container field (Chapter 29).

- **Insert Picture** — select an image file that will be inserted into a current container field, where it will be displayed as an image.
- **Insert Audio/Video** — select an audio or video file that will be inserted into a current container field. This option is only available if the container field is configured to allow interactive content; otherwise use **Insert File**.
- **Insert PDF** — select a PDF file that will be inserted into a current container field. This option is only available if the container field is configured to allow interactive content; otherwise use **Insert File**.
- **Insert File** — select any file that will be inserted into a current container field, where it will be displayed as an enclosed file icon.
- **Current Date** — insert today's date into the current text or date field.
- **Current Time** — insert the current time into the current time field or the current timestamp into the current timestamp field.
- **Current User Name** — insert the current user name into the current field.
- **From Index** — displays a dialog showing a list of unique values for all records for the current field. The user can double-click on any of these values to automatically insert it into the field for the current record.
- **From Last Visited Record** — inserts the value of the current field from the last record viewed into the current record.

Mouse Clicks

When a field is configured with a control style other than Edit Box (Chapter 30) entering or selecting data may be required or optionally be performed with mouse clicks. For example, a check-box style formatting requires a click onto boxes next to the desired value(s) while a calendar style formatting gives the option of entering a date by selecting one from a graphical calendar.

Formatting Field Contents

Depending on a field's format settings on a layout, text entered within it will conform to that formatting unless additional formatting is allowed and applied by the user directly to the text within the field. Fields can contain rich text formatting using the **Format** menu, **Formatting Bar**, or the field's contextual menu.

Closing a Record

Once a user is finished editing, the record needs to be closed by either committing or reverting the changes.

Committing a record closes the record while saving all changes made during the editing session.

A record can be committed by:

- Clicking on the layout outside of any field and away from any other object.
- Typing the Enter key.
- Navigating forward or backward to another record.
- Running a script or clicking a button that uses the **Commit Records/Requests** script step.

FileMaker will perform any data entry validations prior to committing a record and will halt and report problems if any are detected.

Reverting a record closes the record while omitting all changes made during the editing session.

A record can be reverted by:

- Selecting the Records > Revert Record menu.
- Running a script or clicking a button that uses the Revert Records/Requests script step.

Managing Records

Users can create, delete, and duplicate records.

Creating a New Record

A new record can be created in the table displayed on the layout in the front window by using one of these methods:

- Select the Record > New Record menu item.
- Select New Record from the record contextual menu.
- Clicking the New Record icon in the toolbar.
- Running a script or clicking a button that uses the New Record/Request script step.

The new empty record will appear in the front window, open and ready for entry with focus on the first field in the tab order.

Deleting a Record

The current record in the front window can be deleted by using one of these methods:

- Select the Record > Delete Record menu item.
- Select Delete Record from the record contextual menu.
- Clicking the Delete Record icon in the toolbar.
- Running a script or clicking a button that uses the Delete Record/Request script step.

FileMaker will confirm that the user wants to continue with the deletion by presenting a warning dialog, shown in Figure 4-2. If the user confirms the deletion by clicking the Delete button, the record will be permanently deleted from the table. Clicking the Cancel button will dismiss the dialog without any further action taken.

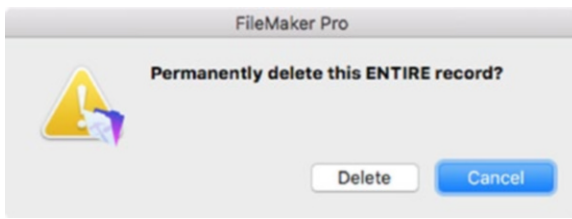


Figure 4-2. The delete record warning dialog presented to the user prior to deleting a record

■ **Note** A user can bypass this warning dialog and instantly delete a record by holding Option (macOS) or Shift (Windows) while selecting the `Delete Record` option! While this feature may be a useful tip for power users, it can be dangerous for new users and perhaps shouldn't be shared with everyone. Custom Menus (Chapter 33) can be used to suppress this feature.

Deleting All Records

The entire found set of records in the front window can be deleted all at once by using one of these methods:

- Select the `Record > Delete All Records` menu item.
- Running a script or clicking a button that uses the `Delete All Records` script step.

FileMaker will confirm that the user wants to continue with the deletion by presenting a warning dialog, shown in Figure 4-3. If the user confirms the deletion by clicking the `Delete All` button, the records in the found set will be permanently deleted from the table. Clicking the `Cancel` button will dismiss the dialog without any further action taken.

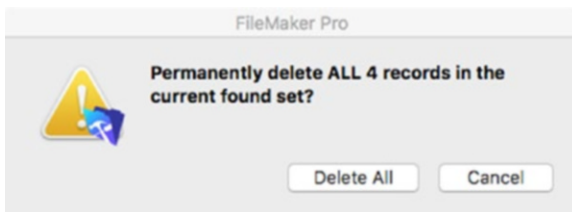


Figure 4-3. The delete all records warning dialog presented to the user prior to deleting all records in the found set

Duplicating a Record

The current record in the front window can be duplicated using one of these methods:

- Select the `Record > Duplicate Record` menu item.
- Select `Duplicate Record` from the record contextual menu.
- Running a script or clicking a button that uses the `Duplicate Record/Request` script step.

Once initiated, FileMaker will immediately create a duplicate of the current record, that is open and ready for entry with focus on the first field in the tab order.

Searching for Records

Searching allows a large list of records to be temporarily pared down to a subset based on the criteria specified by a user or script. Instead of scrolling through a table to visually locate one or more desired records, a search instantly displays only records with matching criteria.

FileMaker offers several methods for searching records:

- Performing fast searches with quick find
- Performing fast searches with find matching records
- Using find mode
- Searching with scripts (Chapter 35)
- SQL queries (Chapter 26)

Except for the SQL Query, each of these is *context sensitive*, meaning they are constructed and executed from within the user interface and are, therefore, based on the context of the table for the layout viewed in the current window.

Performing Fast Searches

FileMaker offers two options for fast searching, each based on a single criterion:

- Searching with Quick Find
- Searching with Find Matching

Searching with Quick Find

The *quick find* feature will search for records that contain a specified value in any field on the current layout that is configured for this purpose (Chapter 29). Access to quick find is through the search text field located in the status toolbar, shown in Figure 4-4.

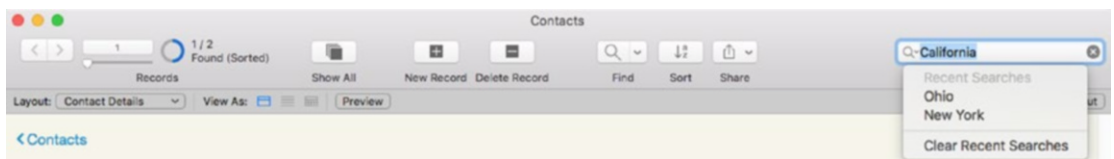


Figure 4-4. The quick find search field in the status toolbar

Performing a Quick Find

To begin a quick find, follow these steps:

1. Open a database window.
2. Navigate to a layout that displays records from the table you want to search.
3. Make the status toolbar visible if it is hidden.
4. Click into the search text area in the status toolbar.
5. Type the search criteria.
6. Type the Enter key.

The find is performed instantaneously and the window is updated with the result; a found set of records that contains the criteria entered. If no records match the criteria, a dialog, shown in Figure 4-5, will appear informing the user of this fact.



Figure 4-5. The dialog notification of no matching records for the quick find criteria entered

Changing the Quick Find Criteria

After a quick find is complete, the criteria entered will remain within the search field and will be highlighted. This allows the user to instantly type new criteria in its place and hit the Enter key again to initiate a new or updated find if necessary. Since the search field is a text field you can copy, paste, drag and drop just as you would with any other text on a computer.

Clearing the Quick Find Criteria

The quick find criteria can be cleared by:

- Clicking the “x” icon inside the search field.
- Selecting the entire text entry and typing Delete.

Clearing the quick find criteria does not affect the current found set, which will continue to display the records from the last find performed.

Using the Recent Search Menu

Clicking on the magnifying icon on the left within the quick find text field displays a menu of the last ten recent quick finds. Any recent find can be re-performed by clicking on the item from this list.

At the bottom of this menu is a Clear Recent Searches option. Clicking this will clear the menu of all recent searches. The recent search menu is automatically cleared when the database file is closed.

Searching with Find Matching

Another rapid find technique is the Find Matching functions, which are available from the field's contextual menu, shown in Figure 4-6.

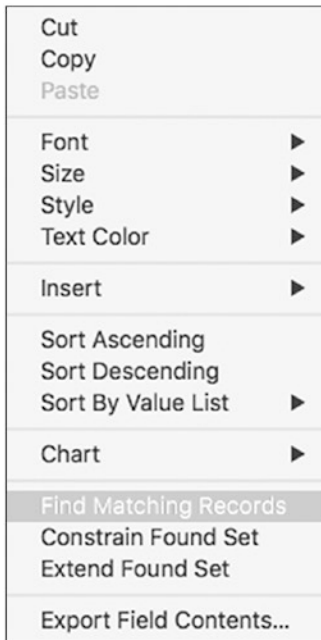


Figure 4-6. The find matching records function selected in the field contextual menu

Each of these three functions will instantly perform a predefined type of search for records that match the value in the current field:

- **Find Matching Records** — This function will perform a new find in the current field for the value in that field on the current record.
- **Constrain Found Set** — This function will narrow the current found set of records to only those within the found set that match that field value for the current record.
- **Extend Found Set** — This function will expand the current found set of records to include other records that match that field on the current record.

■ **Note** The behavior for constraining and extending a found set from the contextual menu varies slightly from that performed by the same options available in the menu bar and script steps.

Using Find Mode

Find mode is a transitional window state that provides an opportunity to enter search criteria directly into one or more fields on one or more find requests using the current layout before initiating a search.

Find mode can be initiated one of the following ways:

- Select the View > Find Mode menu item.
- Click the “Find” icon in the status toolbar.
- Select the Create New Find from the menu on the “Find” icon in the status toolbar.
- Running a script or clicking a button that uses the Enter Find Mode script step.

Identifying Find Mode

The appearance of a window in Find mode will be like Browse mode with a few important differences.

- The status toolbar contains a different set of options when in Find mode (described below).
- The content area of the window displays an empty temporary find request waiting for input that will be used as criteria for locating matching records. In addition to appearing as a blank version of the layout, the availability of some controls and their formatting might change depending on how the layout components are configured (Chapters 27-31).

Status Toolbar (Find Mode)

When a window is placed into Find mode the options available in the status toolbar will change to functions specific to searching. The options displayed will either be the default Find mode options or the user’s customized set.

Default Status Toolbar Items (Find Mode)

The default status toolbar for Find mode, shown in Figure 4-7, contains:

- Navigation controls
- Function buttons
- Saved finds menu
- Layout menu
- View buttons
- Matching records options
- Insert operators menu

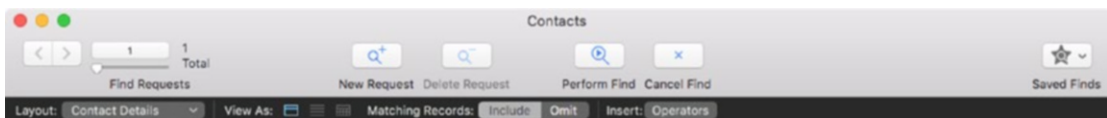


Figure 4-7. The default status toolbar for Find mode

Navigation Controls

At the top left of the toolbar are the navigation controls that appear and function exactly as they do in Browse mode with one important difference; they refer to and control find requests rather than records. A *find request* is a record-like version of the current layout that is used to enter search criteria.

Function Buttons

The following buttons are included in the default Find mode toolbar:

- **New Request** — creates a new find request for entering alternative criteria.
- **Delete Request** — deletes the current find request if it is not the only find request.
- **Perform Find** — performs the find, based on the criteria entered in the find request(s) and returns to browse mode with a found set of the results.
- **Cancel Find** — aborts the find process and returns the window to Browse mode displaying the previous found set.

Saved Finds Menu

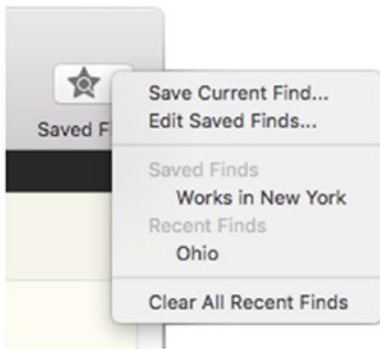


Figure 4-8. *The Saved Finds menu*

On the top right of the Find mode toolbar, is the Saved Finds menu button, which makes available the following functions:

- **Save Current Find** — saves the current find criteria so that the identical search process can be easily repeated in the future.
- **Edit Saved Finds** — opens the Edit Saved Finds dialog.
- **Saved Finds** — displays a list of saved finds, which can be performed with a click.
- **Recent Finds** — displays the last 10 recent finds, which can be performed with a click.
- **Clear All Recent Finds** — clears all the recent finds.

■ **Caution** Saved finds only store the *criteria* and not the resulting found set of records! Each time a find is performed, that criteria is applied in searching *every record* in the table, including those that may not have existed at the time the find was saved.

Layout Menu

The lower, non-customizable level of the Find mode toolbar starts with the Layout menu, which operates like it does in Browse mode. This pop-up menu displays a list of layouts that have the Include in Layouts Menu option selected (Chapter 28).

When a user selects a layout from the list the window view switches to display the selected layout while maintaining Find mode. This can be useful if the user wishes to create a find using multiple fields for the same table that are spread over several different layouts. However, if the user switches to a layout for a different table, criteria entered in layouts representing other tables will be ignored and the find will be performed based on the context and criteria entered for the last table.

View Buttons

Next on the lower portion of the Find mode toolbar are three buttons that will change the current window's view, like they do in Browse mode (Chapter 3). The views that are enabled are controlled by the current layout's view settings (Chapter 28).

Matching Records Options

The matching records option, unique to Find mode, is a toggle switch between two opposing options that control how records matching the criteria entered in the current find request will be handled in relation to the resulting found set:

- **Include** — This is the default behavior for any new find request; any record matching the criteria entered will be *included* in the resulting found set.
- **Omit** — Selecting this option will cause any record matching the criteria entered to be *omitted* from the resulting found set.

When building complex finds, these two settings can be used in various combinations on different find requests to create precisely refined results.

Insert Operators Menu

Clicking the Insert Operators menu in the Find mode toolbar will reveal a pop-up menu of available operators. An operator selected from this menu will be inserted into the field with focus and may be used with or without other criteria to enhance the search parameters. Operators are discussed later in this chapter.

Customizing the Status Toolbar (Find Mode)

The status toolbar in Find mode is customizable at the user-computer level exactly as it is in Browse mode (Chapter 3). The only difference is in the specific button functions available.

To begin customizing the status toolbar, enter Find mode and then select the View > Customize Toolbar menu to open the customization panel attached to the window as shown in Figure 4-9.

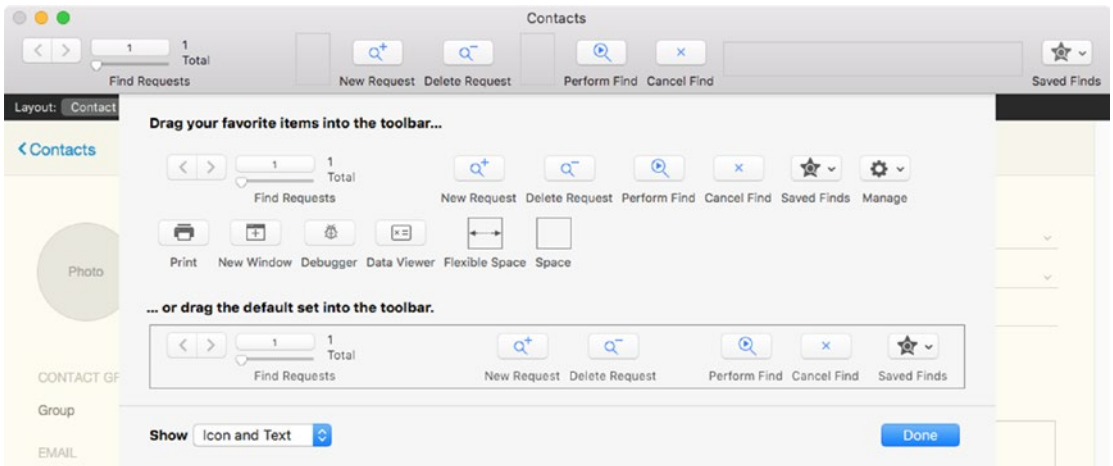


Figure 4-9. The customize toolbar panel attached to the Contacts database window in Find mode

Entering Find Criteria

Once a window is in Find mode, criteria can be typed into any field on the layout that is configured to allow it. Typed input does not result in a new record but a temporary find request that will be used to perform the search. When you type text into a field it indicates to FileMaker that you want to search for records containing *this value* in *that field*.

Unlike with the Quick Find and Find Matching Record functions, criteria can be typed into more than one field in Find mode. When doing so, each field with criteria acts like an “and” type search. In other words, when you enter criteria into two fields, a record will only be a match if it has both the value from the first field and the second field. It is like indicating that you want to search for records where *value A* is in *field A* and *value B* is in *field B*. *The same holds true regardless of the number of fields entered on a single find request. To enter an “or” type search, create a new find request for each set of criteria.*

Entry in Find mode works essentially the same as in Browse mode; use keystrokes, the insert menu or mouse clicks to enter data depending on the control type and use the Tab key to move the focus on to other fields.

Using Search Operators

A *search operator* is a character or set of characters that are used to narrow the focus of the search. Operators can be added to a field while in Find mode using these methods:

- Typing the operator directly into a field.
- Clicking the Insert Operators control in the second level of the toolbar to.
- Selecting the Operators submenu from the field’s contextual menu.

The Insert Operators menu, shown in Figure 4-10, allows quick access to a list of operators. An operator selected from this menu will be inserted into the current field.

=	match whole word (or match empty)
==	match entire field
!	find duplicate values
<	less than
≤	less than or equal
>	greater than
≥	greater than or equal
...	range
//	today's date
?	invalid date or time
@	any one character
#	any one digit
*	zero or more characters
\	escape next character
""	match phrase (from word start)
*""	match phrase (from anywhere)
~	relaxed search (Japanese only)

Figure 4-10. The Insert Operators menu

The available Find mode operators are described in Table 4-1:

Table 4-1. A description of the available Find mode operators

Operator	Description
=	Use in front of a value to match a whole word within the field. Entered by itself, it will match records where that field is empty.
==	Use in front of a value to match an entire phrase within the field.
!	Use to find any records where the field has one or more duplicate values on other record(s).
<	Use to find numeric or text values that are less than the value entered after the symbol.
≤	Use to find numeric or text values that are less than or equal to the value entered after the symbol.
>	Use to find numeric or text values that are greater than the value entered after the symbol.
≥	Use to find numeric or text values that are greater than or equal to the value entered after the symbol.
...	Use to find a range of values. For example, entering “1/15/2017...1/30/2017” will find all records where the field contains a date including or between the two dates entered.
//	Use to find records where the field contains today’s date.
?	Use to find records where the field contains an invalid date or time.
@	Use to find records where the field contains any one character or combined string of any character. For example, “@” will find “D” or “T” and “@@@” will find “Door” or “Test.”

(continued)

Table 4-1. (continued)

Operator	Description
#	Use to find records where the field contains any one number or combined length of any number. For example, “#” will find “3” or “8” and “##” will find “33” or “81.”
*	Use in place of a character to create a search pattern. For example, “1/15/*” will find any records where the field contains a date of January 15 for any year.
\	Use to escape the next character. This can be useful when searching for a literal operator by treating the operator as part of the search criteria and not as an operator. For example, to search for any records where the field contains a quote symbol, enter \”.
“”	Use to match the phrase typed between the quote marks.
*“”	Use to match the phrase typed between the quote marks. In a field containing a lot of text.
~	Use to perform a relaxed search in Japanese text.

Managing Find Requests

Find requests are like Browse mode records both in their location within the content area of a Window, similar appearance, and the options available for managing them. Like with records, it is possible to create, delete, and duplicate find requests to create complex searches.

Creating New Find Requests

Adding additional criteria to a single find request will *narrow* the results since a matching record must contain each value specified in each respective field. By contrast, adding an additional find request, each with their own criteria will *expand* the results since matching records must contain the criterion from one request *or* the other.

When in Find mode, a new request can be created as easily as creating new records in Browse mode. Any of these methods will create a new request:

- Select the Request > New Request menu item.
- Select New Request from the record contextual menu.
- Clicking the New Request icon in the toolbar.
- Running a script or clicking a button that uses the New Record/Request script step.

The new request will appear as a new, empty record awaiting find criteria. Each request is then accessible from the navigation area of the toolbar, shown in Figure 4-11.

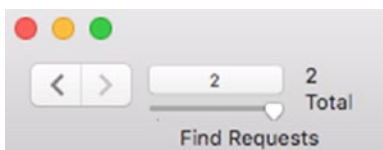


Figure 4-11. The navigation area showing two requests in Find mode

The navigation controls work just as they do in Browse mode, except you are clicking or scrolling back and forth between find requests instead of actual records.

In a Contacts file, to search for records that contain either “NY” or “OH” in a State field, follow these steps:

1. Enter Find Mode.
2. Type “NY” into the State field.
3. Create a new find request.
4. Type “OH” into the State field.
5. Click the Perform Find button or type the Enter key.

The results of this find should be all contacts that work in either state blended into a found set in the order in which they were created.

■ **Note** When creating many find requests, FileMaker will eventually present a dialog reminding you that you are in Find mode. This is designed as a warning to avoid a situation where a user might mistakenly assume they are creating *records* when they are really creating *find requests* since the views and process are so similar.

Deleting a Find Request

By its nature, a Find session must contain at least one request. Therefore, the default find request that is automatically created at the start of a new Find session can’t be deleted. However, once the session contains more than one request, the Delete Request function becomes active, and it is possible to delete any requests as needed provided that at least one is retained.

Any of these methods will delete the current find request:

- Select the Request > Delete Request menu item.
- Select Delete Request from the record contextual menu.
- Clicking the Delete Request icon in the toolbar.
- Running a script or clicking a button that uses the Delete Record/Request script step.

■ **Caution** Unlike when deleting records, deleting a find request happens instantly without a warning dialog.

Duplicating Find Requests

Like records, find requests can be duplicated by any of the following methods:

- Select the Request > Duplicate Request menu item.
- Select Duplicate Request from the record contextual menu.
- Clicking the Duplicate Request icon in the toolbar.
- Running a script or clicking a button that uses the Duplicate Record/Request step.

Performing a Find

Once all the criteria are entered into the appropriate fields, the search process can be performed by any of these methods:

- Type the Enter key.
- Click the Perform Find button in the toolbar.
- Select the Requests > Perform Find menu.

Once initiated, FileMaker will search the current table and display the results. If there are no results, a dialog will be presented notifying the user of this fact with an option to modify the find or cancel the process. If results are found, they will be listed in creation order as the current found set in the window.

Manipulating a Previously Executed Find

Once a find has been performed, FileMaker offers three methods for further refining the results without having to start over with a new request:

- Modifying the last find
- Extending the last find
- Constraining the last find

The first of these is triggered from Browse mode as an alternative method to using the Enter Find Mode command; it will automatically enter find mode and restore the last find criteria. The Extend and Constrain commands are available when criteria is specified: either in a new find session when they are used in place of the Perform Find command or when field contents are selected and the command is selected from the contextual menu. These control how the new criteria should be used in relation to an existing found set in the same window.

Modifying the Last Find

The Modify Last Find command will re-create the find request(s) and criteria of the last find performed in the current table as a new Find mode session. This quickly creates a new find request identical to the last performed without requiring it to have been formally saved the last find.

The last find criteria can be recreated by using any of these methods from any window in Browse mode:

- Select the Record > Modify Last Find menu item.
- Select Modify Last Find from the menu connected to the toolbar's Find icon.
- Run a script or click a button that includes the Modify Last Find script step.

Extending the Last Find

The Extend Found Set command will use the new criteria to add records to the existing found set. Any records that match the new criteria that are not yet included will be added to the found set.

This command can be used the following ways:

- Select Extend Found Set from the field contextual menu when highlighting some text within a field in Browse mode.
- Select the Request > Extend Found Set menu instead of Perform Find after setting up some criteria in a Find mode session.

For example, if viewing a found set of contacts generated by searching for a Work State of “NY”, this result can be extended to include contacts that have a Work State of “OH,” by following these steps:

4. Enter find mode.
5. Type “OH” into the Work State field.
6. Select the Request > Extend Found Set menu.

The resulting found set should consist of a list of records with a Work State of “NY” or “OH” in creation order.

Constraining the Last Find

The Constrain Found Set command will use the new criteria of the find request(s) to reduce to the existing found set. Any records within the current found set that match the new criteria will be included while any that do not match the new criteria will be excluded. Any previously excluded records will continue to be excluded. In other words, the new search is only performed against the records within the current found set.

This command can be used the following ways:

- Select Constrain Found Set from the field contextual menu when highlighting some text within a field in Browse mode.
- Select the Request > Constrain Found Set menu instead of Perform Find after entering some criteria in a Find mode session.

For example, if viewing a found set of contacts generated by searching for a Work State of “NY”, this result can be constrained to only include contacts that have a Work City of “New York,” by following these steps:

4. Enter find mode.
5. Type “New York” into the Work City field.
6. Select the Request > Constrain Found Set menu.

The resulting found set should be only those records with both a Work State of “NY” and a Work City of “New York.”

Building Complex Finds

Using multiple requests with a combination of the Include and Omit options, it is possible to build extremely sophisticated find requests. Although many users may never need to construct a find more complex than a single request with a few fields worth of criteria, professionals and power users will.

For example, to find every contact that works in the state of NY or OH but does not work in a city of New York or Columbus, follow these steps:

1. Enter find mode.
2. On the default find request, enter “NY” in the Work State field.
3. Create a new find request.
4. Enter “OH” in the Work State field.
5. Create a new find request and click the “Omit” option in the toolbar.
6. Enter “New York” in the Work City field.

7. Create a new find request and click the “Omit” option in the toolbar.
8. Enter “Columbus” in the Work City field.
9. Perform the find.

Working with Saved Finds

FileMaker includes a rather complex set of dialogs in which a user can save, create, and edit stored find request criteria for future use.

Saving a Find

The last executed find can be saved by one of these methods:

- Select the Records > Saved Finds > Save Current Find menu item while in Browse mode.
- Select Save Current Find from the menu on the Find toolbar icon while in Browse mode.
- Select Save Current Find from the menu on the Saved Finds toolbar icon after entering find criteria in Find mode.

Each of these will open the current or last find in a Specify Options for the Saved Find dialog, shown in Figure 4-12.

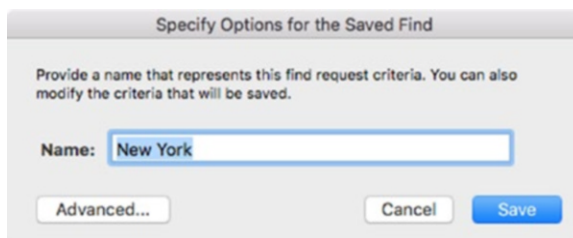


Figure 4-12. The Specify Options for the Saved Find dialog

The dialog allows the user to edit the name and criteria of the find being saved.

Naming a Saved Find

The name of the find being saved will default to the criteria of the find. It is a good idea to enter an enhanced name to provide clarity about what the find represents. For example, it is a good idea to include the name of the table and field(s) being searched rather than just the criteria.

In a multi-table database, the list of saved finds is not filtered based on the current layout’s table. Since a find will be executed from the current layout, including the table in the name will help the user to understand the proper context for successful use of the saved find.

Including the name of the field(s) being searched will help avoid confusion about where exactly the criterion is used. In the example above, “New York” was entered into the Work City field but it could have been entered into the Home City field.

In this example, a better name that includes the table and field name and the criteria could take the following form:

(Contacts) Work City = New York

Understanding the Criteria Selection

The find criteria, accessible by clicking the Advanced button in the dialog, will vary depending on the context when the save process was initiated. When in Browse mode or in Find mode when no find criteria have been entered, the criteria from the previous find will be stored. When in Find mode with new criteria entered, the new values will be stored.

Saving or Canceling the Process

Clicking the Save button will close the dialog and complete the save process. The find will be stored within the database file and accessible by other users with the same level of access. Clicking the Cancel button closes the dialog without saving.

Performing a Saved Find

A saved find can be quickly executed by one of the following methods:

- Selecting it by name from the Records > Saved Finds submenu.
- Selecting it by name from the pop-up menu of the Find icon in the toolbar in Browse mode.
- Selecting it by name from the pop-up menu of the Saved Finds icon in the toolbar in Find mode.

When this command is selected in Browse mode, the find is immediately executed and the results become the found set in the current window. In Find mode, the selection of a saved find sets up the request(s) and then waits for the user to modify them or perform the find.

Managing Saved Finds

Saved finds can be managed from within the Edit Saved Finds dialog, shown in Figure 4-13. This dialog is accessible from the following menus:

- Select the Records > Saved Finds > Edit Saved Finds menu item when in Browse mode.
- Select Edit Saved Finds from the Find menu in the toolbar when in Browse mode.
- Select Edit Saved Finds from the Saved Finds menu in the toolbar when in Find mode.

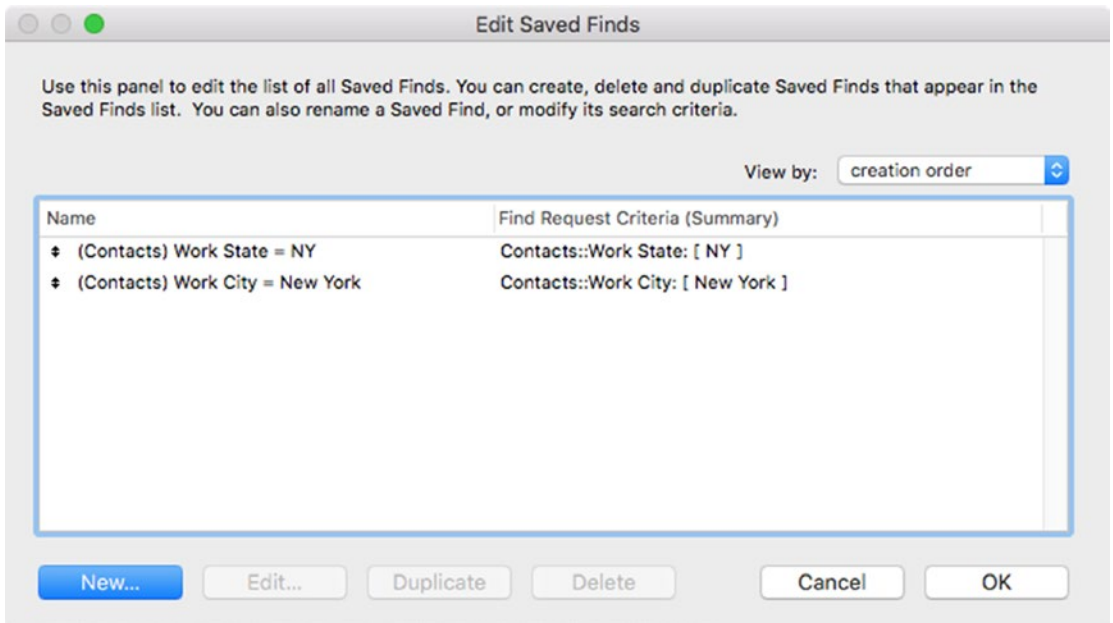


Figure 4-13. The *Edit Saved Finds* dialog

The dialog displays a list of all finds stored within the current database file and the following function buttons:

- **New** — creates a new, empty find and opens it in the *Specify Options for the Saved Find* dialog.
- **Edit** — opens the selected find in a *Specify Options for the Saved Find* dialog.
- **Duplicate** — creates a duplicate of the selected find.
- **Delete** — deletes the selected find.
- **Cancel** — closes the dialog without saving changes.
- **OK** — saves changes and close the dialog.

Managing a Find Request

Click the **Advanced** button from a *Specify Options for the Saved Find* dialog to display the stored criteria in a *Specify Find Requests* dialog shown in [Figure 4-14](#).

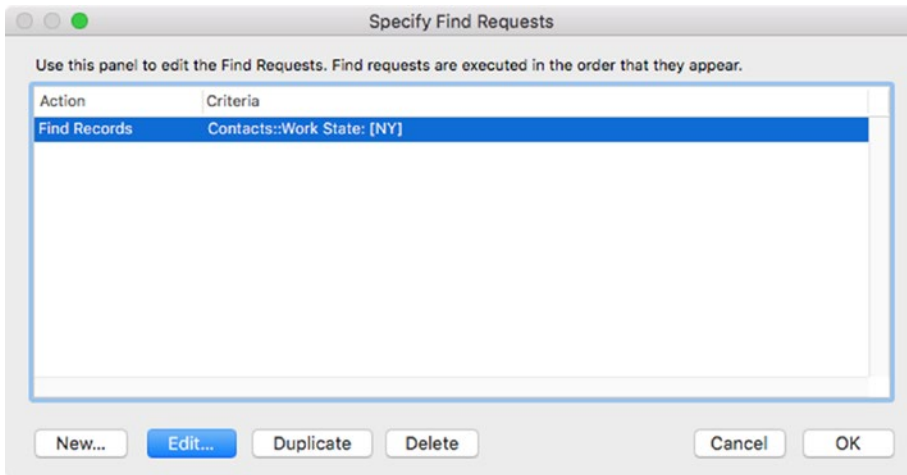


Figure 4-14. The Specify Find Requests dialog

The dialog displays a list of the requests that make up one stored find. Each line represents a single find request containing one or more criterion. These can be managed using the following function buttons:

- New — creates a new, empty find request for the current find in an Edit Find Request dialog, shown in Figure 4-15.

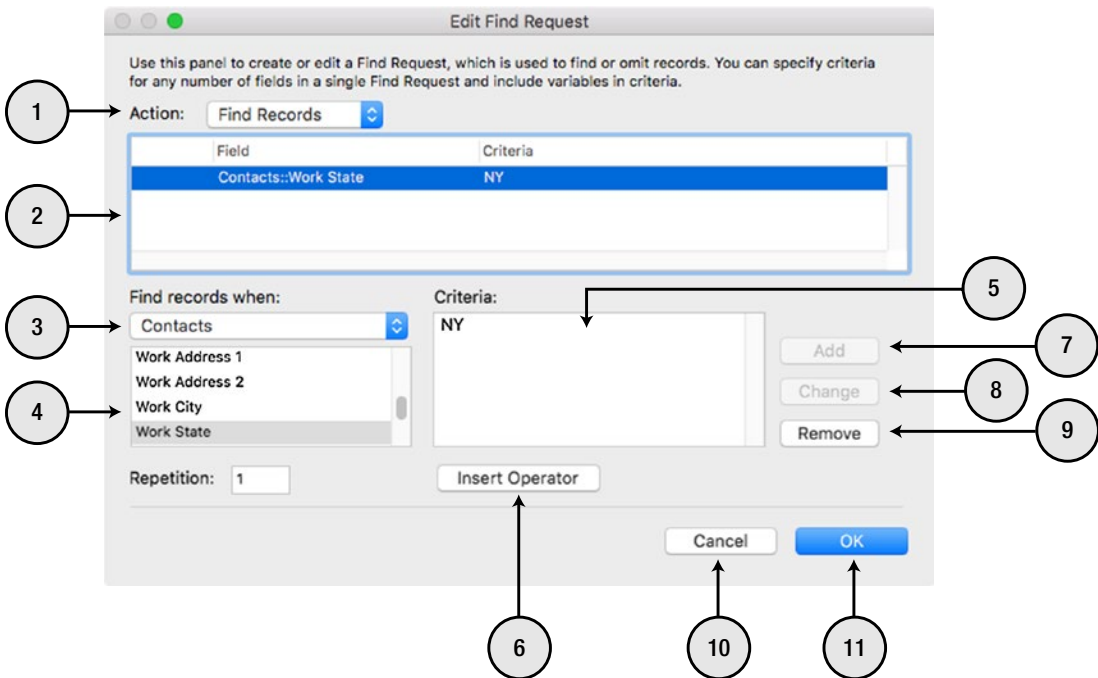


Figure 4-15. The Edit Find Request dialog

- **Edit** — allows editing of a selected request by opening it in an **Edit Find Request** dialog.
- **Duplicate** — creates a duplicate of the selected request.
- **Delete** — deletes the selected request.
- **Cancel** — closes the dialog without saving changes.
- **OK** — saves changes and closes the dialog.

Editing a Find Request Criteria

A find request in a **Specify Find Request** dialog can be edited by opening an **Edit Find Request** dialog, shown in Figure 4-15. This dialog can be opened by one of two methods:

- Selecting the request in the list and clicking the **Edit** button.
- Double-clicking on the request in the list.

This dialog is a rather complex set of interface elements that allows a batch of find requests to be edited one by one in a single interface. The dialog controls include the following:

1. **Action** — choose **Find Records** or **Omit Records**, which are the equivalent of the **Include** and **Omit** matching records option in the toolbar's second row.
2. **Criteria List** — lists each field-criteria pair that makes up the request. Whichever row is selected will be displayed in the bottom of the dialog for editing.
3. **Table** — contains a list of all related tables with the table containing the current field-criteria pair selected.
4. **Fields** — displays all the fields for the selected table with the search field.
5. **Criteria** — This text area contains the value that will be search for within the selected field.
6. **Insert Operator** — contains a list of available operators; select one to insert it into the current criteria.
7. **Add** — creates a new field-criteria pair in the request list based on the values selected.
8. **Change** — modifies the currently selected field-criteria pair based on the edited values.
9. **Remove** — deletes the currently selected field-criteria pair.
10. **Cancel** — closes the dialog without saving changes.
11. **OK** — saves changes and closes the dialog.

■ **Tip** To avoid confusion, especially for complex finds, create the find in the interface and then save it and look at the request in this dialog. That will help you understand how to construct a find in the dialog.

Working with Found Sets

Unless a find matches every record within the table, the result is likely a subset of the total record count. Once established, the records in this set can be manipulated in various ways.

Defining a Found Set

A *found set* is a group of records that are visible and navigable within a given window for a particular table. While the found set may contain all records within a table, the term generally refers to a *subset* of records generated by a search. Records that are not part of the found set, and therefore not actively visible or navigable, are called the *omitted set*.

With the toolbar hidden, it is impossible to know if there is an active found set of less than the total number of records unless a developer has created layout elements to inform the user. Even with the toolbar visible, it's easy to miss. An uninitiated user might panic at first wondering where all their records have gone.

The difference can always be seen in the record count located in the navigation area of the toolbar, as shown in Figure 4-16. When all records are accessible, one number is displayed: the total count of records. When a found set is active, the first number displayed indicates how many records make up the visible found set, while the second number indicates the total number of records in the table including those that are omitted from the found set.

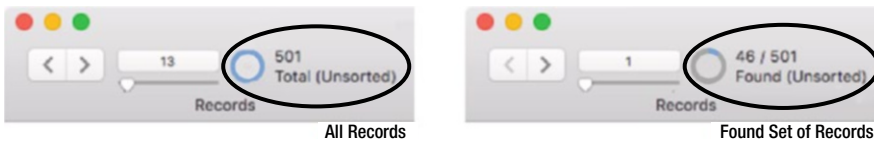


Figure 4-16. The different information displayed in the navigation area when viewing all records or a found set

When navigating to other layouts that display records for the same table within the same window, the found set remains constant until the user or a script changes it. Every table within a window has its own found set. When using multiple windows, each window has its own context and, therefore, a unique found set for each table contained in the file.

Changing the Records in the Found Set

The records that are included within a found set can be the result of a find. However, once established, membership in the set can be adjusted or transformed in various ways with the following commands:

- Show All Records
- Omit Record
- Omit Multiple
- Show Omitted Only

Each of these commands can be accessed from the Records menu while in Browse mode or by running a script or clicking a button that includes them. None will affect the actual records contained within the table, only their inclusion within the found set.

Show All Records

The `Show All Records` command will replace the found set in the current window with all records in the table. This command is available in both Browse and Find modes. When used in Find mode, it acts as the find criteria and perform find command, returning to Browse mode and displaying all records. If the initial found set is sorted, using this command will unsort the records.

Omit Record

The `Omit Record` command will move the current record from the found set to the omitted set. This helps to fine-tune the results of a find without the need to perform a new, more complex search.

Omit Multiple

The `Omit Multiple` command displays a dialog for the user to enter a number indicating how many records they wish to move from the found set to the omitted set, starting from the current record. This helps to fine-tune the results of a find like the single-record `Omit` command but in larger record chunks.

For example, if the user is viewing the first record of a found set of 100 records and they choose this command and enter the number 10, the first 10 records will be omitted leaving the remaining 90 records as the new found set. However, if they are viewing record 50 of 100 and do the same thing, then record 50-59 will be omitted leaving records 1-49 and 60-100 as the new reduced found set.

Show Omitted Only

The `Show Omitted Only` command will replace the current found set with the omitted set, effectively causing the two sets to trade places. The found set will become the omitted set, and the previously hidden omitted set will become the visible found set.

Sorting Records in the Found Set

After searching, the found set of records will be displayed in creation order, which is considered an unsorted state. The records can be re-sorted as needed by any local or related field(s) and unsorted again using the `Sort` command configurable through a dialog, shown in Figure 4-17. The dialog can be opened using any of the following methods from Browse mode:

- Select the `Records > Sort Records` menu item.
- Select `Sort Records` from the Record contextual menu.
- Clicking the `Sort` button in the toolbar.

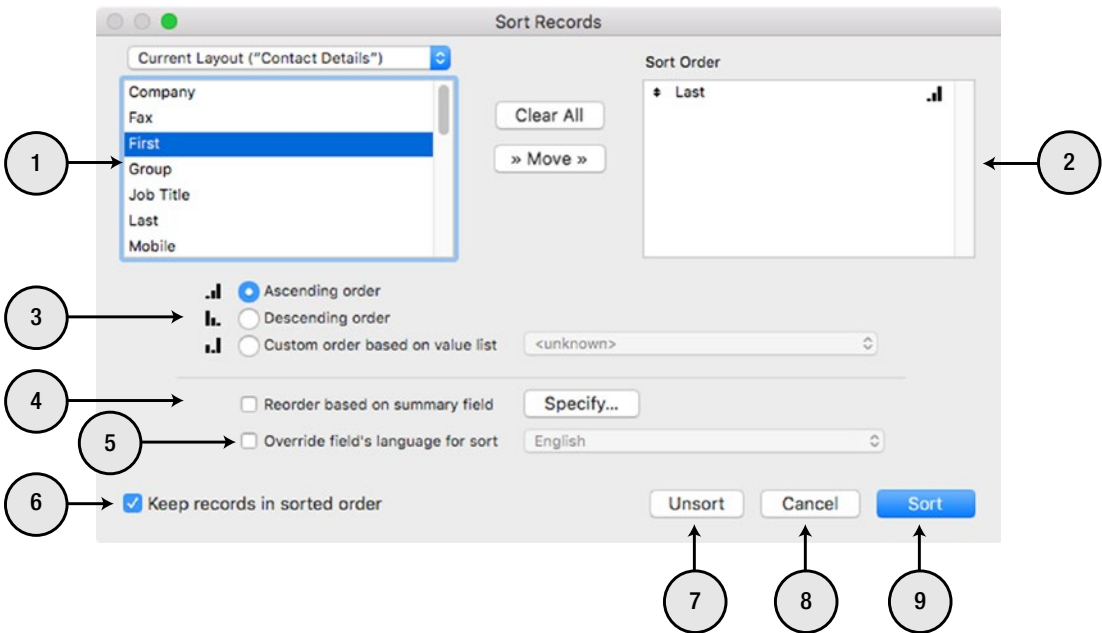


Figure 4-17. The Sort Records dialog

The following controls are available on the Sort Records dialog:

1. **Field selector** — The field list displays every field for the table or layout selected from the pop-up menu above, which contains every table occurrence in the current file but only allows selection of the current table or those related to it.
2. **Sort field order** — This list specifies the order of field(s) by which the found set will be sorted. A field can be added to the list by either double-clicking it in the field selector or selecting it and clicking the Move button. When a field in the sort order is selected the Move button will become a Clear button. To remove a field from sort order list, double-click it or select it and click the Clear button. All fields listed can be removed by clicking the Clear All button. Fields can be dragged and dropped to rearrange the order within the sort order list.
3. **Sort field direction** — This choice of three radio buttons control the direction or order of the sort for a selected field in the sort order. The options are ascending (default); descending; or a custom, nonstandard order based on a value list. The adjacent icons are displayed next to each field in the sort field order indicating which option will be used for which field. Fields retain the sort order based on the radio button selected at the time a field is added to the sort order. To change the option for a field, select it in the sort order and click another directional option.

4. **Reorder based on summary field**— This check box and associated Specify button allows the selection of a summary field that will be used as a secondary sort to reorder records based on a field value that is first sub-summarized by another sort field. First add the field that is used by a summary field. Then, with that field selected, check this box and select the summary field to re-sort the summarized records. When used, a small document icon will appear in place of the sort direction indicator next to the field in the sort order.
5. **Override field's language for sort** — This check box and associated pop-up menu allows the selection of a different language to control indexing for the sort for text fields.
6. **Keep records in sorted order** — When this box is checked, changes to the values of sort fields on records will not automatically update the sort order. Uncheck this box to have the sort dynamically update after any change is committed.
7. **Unsort** — This button will unsort the found set of records and close the dialog.
8. **Cancel** — This button will close the dialog without modifying the sort order of the found set.
9. **Sort** — This button will re-sort the records in the found set and close the dialog.

The sort status of records will be indicated in the record navigation area of the toolbar as shown in Figure 4-18. If the Sort dialog is reopened while records are already sorted, it will display the fields controlling the current sort order.

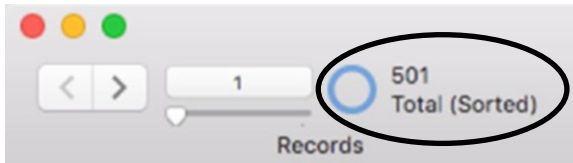


Figure 4-18. The record count with an indication that the records are currently sorted

Modifying Field Values for Records in the Found Set

In addition to the obvious advantage of isolating a set of records for manual work, creating a found set allows changes to field values to be applied to a batch of records all at once. There are several commands that always or optionally apply to all the records in a found set, including the following:

- Replace Field Contents
- Relookup Field Contents
- Find and Replace
- Spell Checking

Replace Field Contents

The `Replace Field Contents` command allows a field value to be updated for the field in focus for every record in the current found set at once. The new replacement value can be a choice of:

- The value in the field for the current record.
- Serial numbers, starting from a specified number on the first record and incremented a specified amount for each subsequent record.
- The result of a calculation formula.

These options are available from the `Replace` dialog, shown in Figure 4-19.

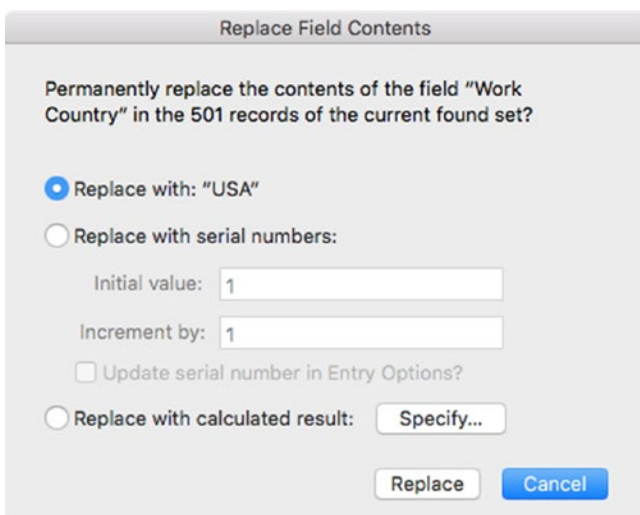


Figure 4-19. The `Replace Field Contents` dialog

Replacing with Current Contents

To apply the contents of a field on the current record to that same field for all records in the found set, click into the field and use one of the following methods to open the dialog:

- Select the `Records > Replace Field Contents` menu item.
- Run a script or clicking a button that uses the `Replace Field Contents` command.

The default option in the dialog is to replace the contents of the current field for every record in the found set with the value of that field for the current record. Clicking the `Replace` button will close the dialog and complete that task, instantly inserting “USA” into the `Work Country` field for every record.

Replacing with Serial Numbers

When a field containing serial numbers needs to be updated for a group of records, select the **Replace with serial numbers** radio button and follow these steps:

1. Enter the **initial** value that should be entered into the field for the first record in the found set. The value entered can have a non-numeric prefix if desired. For example, entering a value of “Test-1” is valid and only the numeric portion will be incremented for subsequent records.
2. Enter a numeric value in the **increment by** field. This indicates the amount by which the serial number will be incremented for each subsequent record in the found set.
3. If the field being updated is defined to have a serial number auto-assigned during data entry, checking the **Update serial number in Entry Options** check box will update the settings so that the next value entered on a new record will be the incremented value of the last record in the found set after the **Replace** function is concluded.

Once these settings are configured, click the **Repl**ace button to close the dialog and complete that task.

Replacing with Calculated Result

To specify a calculated result as the value to be entered into the field for the found set, click the **Specify** button. This will open a **Specify Calculation** dialog, into which you can choose a function or type a custom calculation (Chapter 12).

The calculation can evaluate to a standard result that will be applied to the field of every record in the found set or it may use field values and other criteria to dynamically calculate a unique value for each record.

Relookup Field Contents

Fields that pull data from related records using the **Lookup** feature (Chapter 8) are not automatically updated when the match field of the related table or the lookup source field is modified. The **Relookup Field Contents** command will force all fields looked up by the selected match field for every record in the found set.

To perform a **Relookup**, follow these steps:

1. Create a found set containing only the records you wish to update.
2. In **Browse** mode, click into the match field that establishes the relationship through which you want to update.
3. Select the **Records > Relookup Field Contents** menu item or run a script containing a **Relookup Field Contents** script step.
4. Click the **OK** button in the alert dialog.

Find and Replace

FileMaker features a traditional, word-processing style, Find and Replace command.

This function will locate text within one or all fields for one or all records in the current found set and optionally replace it with alternate text. The following methods can be used to access this function:

- Select an option from the Edit > Find/Replace > Find/Replace menu item.
- Run a script or clicking a button that uses a Find/Replace script step.

This will open the Find/Replace dialog, shown in Figure 4-20. This dialog is modal and will not allow any other input on records until it is dismissed.

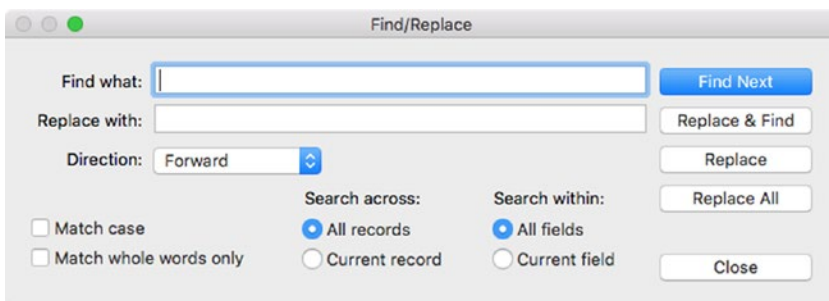


Figure 4-20. The Find/Replace dialog

The features available should be familiar to anyone with minimal computing knowledge and word-processor experience so an exhaustive tutorial should be unnecessary. The following summarization of the available features accessible through the dialog should suffice:

- The text to search for is typed into the Find what field.
- Optional replacement text is entered into the Replace with field.
- The Direction pop-up menu offers the choice of moving through the fields on a record and the records in a found set forwards or backwards.
- Selecting the Match case check-box will force the find process to only consider a match when the case of text in the field matches what is typed in the Find what field exactly.
- Selecting the Match whole words only check box will force the find process to only consider a match when the text in a field contains the entire, separate word(s) typed into the Find what field.
- The Search across radio buttons instructs the function whether it should search all records in the found set or just the current record.
- The Search within radio buttons instructs the function whether it should search all fields on the layout or just the current field.
- The Find Next button will locate and highlight the next instance of the text from the Find what field in the current field, next field, or next record, depending on the other settings in the dialog.

- The **Replace & Find** button will either locate the first instance of a match, if one has not yet been made, or replace the currently highlighted matched text with the replacement text and then locate the next instance of a match.
- The **Replace** button will perform the replace function if a find has already selected a matching instance of the search text. Afterwards, the cursor remains immediately after the replaced text.
- The **Replace All** button will replace all matching instances of the search text based on the settings in the dialog.
- The **Close** button will close the Find/Replace dialog.

The other options only available in the **Edit > Find/Replace** submenu are the following:

- **Find Again** — will repeat the last Find command, stepping to the next instance of a match if the command is already in process.
- **Replace & Find Again** — will either perform the first find match or, if one is already selected, perform the replace and move to the next instance of a match.
- **Find Selected** — will find the next instance of the selected text using the last settings from the dialog but without opening it.

Spell Checking

FileMaker has a fully featured spell checker that can check and correct a selected piece of text, the contents of a field, every field on a record or field(s) for all records within the found set.

Check Spelling Menu

The **Edit > Check Spelling** submenu has several commands that provide standard spell checking functionality:

- **Check Selection** — spell check the selected text in the current field, stepping through each questionable word and offering alternatives through the Spelling dialog.
- **Check Record** — spell check every field on the layout for the current record, stepping through each questionable word and offering alternatives through the Spelling dialog.
- **Check All** — spell check every field on the current layout for every record in the found set, stepping through each questionable word and offering alternatives through the Spelling dialog.
- **Correct Word** — This function will only be enabled when the **Check spelling as you type file** option is set to **Beep on questionable spellings**. Once a word is typed and is detected as a potentially misspelled word, selecting this function in the menu will open the last word typed in the Spelling dialog.
- **Select Dictionaries** — opens a dialog that allows the choice of a language and dictionary.
- **Edit User Dictionary** — opens a dialog that allows custom terms to be added to, imported to, or exported from the user dictionary on the local computer.

Contextual Spelling Features

There are spelling-related features available from the File Options dialog (Chapter 6). One of these will cause questionable words to be highlighted with a red underline like that found in word-processing applications.

When this feature highlights a word, suggested spellings and alternate words are available at the top of the text contextual menu, shown in Figure 4-21.

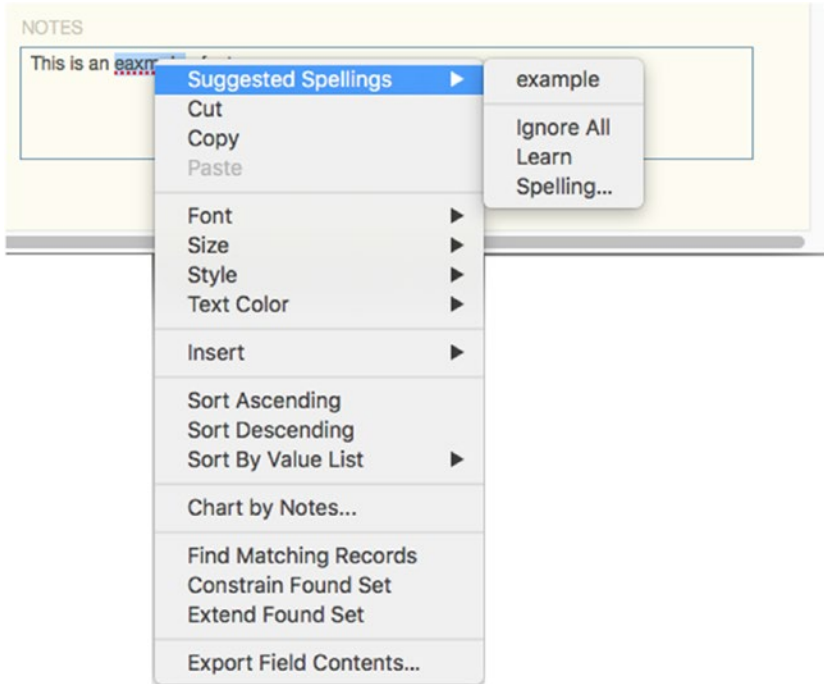


Figure 4-21. The spelling features at the top of the text contextual menu when selecting a questionable word

The following spelling-related commands appear at the bottom of this menu:

- Ignore All — will ignore all instances of the questionable spelling of the word.
- Learn — will add the selected word to the user dictionary.
- Spelling — will open the Spelling dialog to begin a more formal spell check of the selected word.

This feature can be turned off for individual fields on a layout (Chapter 29, “Include field for Quick Find” setting).

Printing

The contents of a database window can be previewed and/or printed using one of several commands available through the standard **Print** dialog window. The print settings and page setup configuration for the window can be configured manually through standard **Page Setup** and **Print** dialogs by a user, directly with a script, or a combination of the two.

While layouts can be designed specifically for printing (Chapter 28), any layout can be printed.

Page Setup

A database file's page setup options can be configured while in **Browse**, **Preview**, or **Layout** mode, using one of the following two methods:

- Selecting **File > Page Setup** to open the **Page Setup** dialog, shown in Figure 4-22.

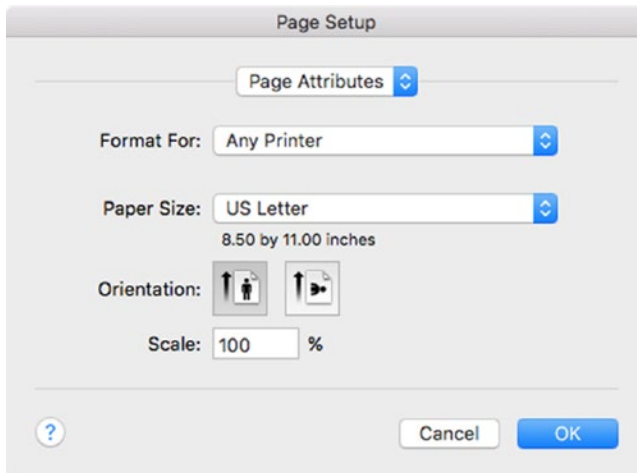


Figure 4-22. The **Page Setup** dialog window

- Running a script or clicking a button that uses a **Page Setup** script step.

The **Page Setup** dialog is a standard operating system dialog that allows control of how layouts in the file will behave when printing.

Using Preview Mode

Preview mode is a transitional window state that gives a user the opportunity to view a layout exactly as it will appear when printed.

Preview mode can be initiated one of the following ways:

- Select the **View > Preview Mode** menu item.
- Click the “Preview” button in the status toolbar.

- Select “Preview” from the mode menu in the window footer.
- Running a script or clicking a button that uses the `Enter Preview Mode` script step.

Identifying Preview Mode

The appearance of a window in Preview mode will be like Browse mode with the following differences:

- The status toolbar contains a different set of button options when in Preview mode.
- The content area displays the layout formatted as one or more pages as they will appear when printed.
- Interactive layout objects such as buttons, tabs, and slide controls may still be displayed but no longer be functional.
- Some layout objects and data may be invisible, reformatted, or positioned depending on their settings (Chapter 29).

Status Toolbar (Preview Mode)

When a window enters Preview mode, the options available in the status toolbar will change to functions specific to printing. The options displayed will either be the default Preview mode options or a user-customized set that applies to all databases on a user’s computer.

Default Status Toolbar Items (Preview Mode)

The default status toolbar for Find mode, shown in Figure 4-23, contains the following:

- Navigation controls
- Function buttons
- Layout menu
- Exit preview button

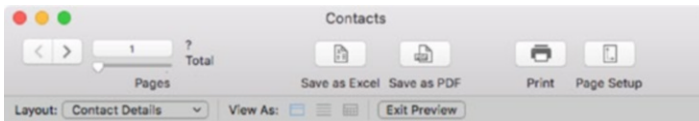


Figure 4-23. The status toolbar for Preview mode

Navigation Controls

At the top left of the toolbar are the navigation controls that appear and function like how they do in Browse mode with one important difference: they refer to and control *pages* rather than records. At first, the page count will be displayed as a question mark because the entire document has not yet been rendered. Clicking through pages or scrolling to the end will force the page count to be updated.

Function Buttons

The following function buttons are included in the default Preview mode toolbar:

- **Save as Excel** — opens a **Save Records as Excel** dialog that allows the user to configure how the records displayed will be exported into an excel format.
- **Save as PDF** — opens a **Save Records as PDF** dialog that allows the user to configure how the pages displayed will be saved as a PDF file.
- **Print** — opens the **Print** dialog.
- **Page Setup** — opens the **Page Setup** dialog.

Layout Menu

The lower, noncustomizable level of the Preview mode toolbar starts with the **Layout** menu, which operates the same as it does in Browse and Layout modes.

Exit Preview Button

The only other control in the lower level of the Preview mode toolbar is an **Exit Preview** button that will return the window to Browse mode.

Customizing the Status Toolbar (Preview Mode)

The status toolbar in Preview mode is customizable at the user-computer level exactly as it is in other modes. The only difference is the types of button functions that can be made available.

To begin customizing the status toolbar, enter Preview mode and then select the **View > Customize Toolbar** menu item to open the customization panel attached to the window as shown in Figure 4-24.

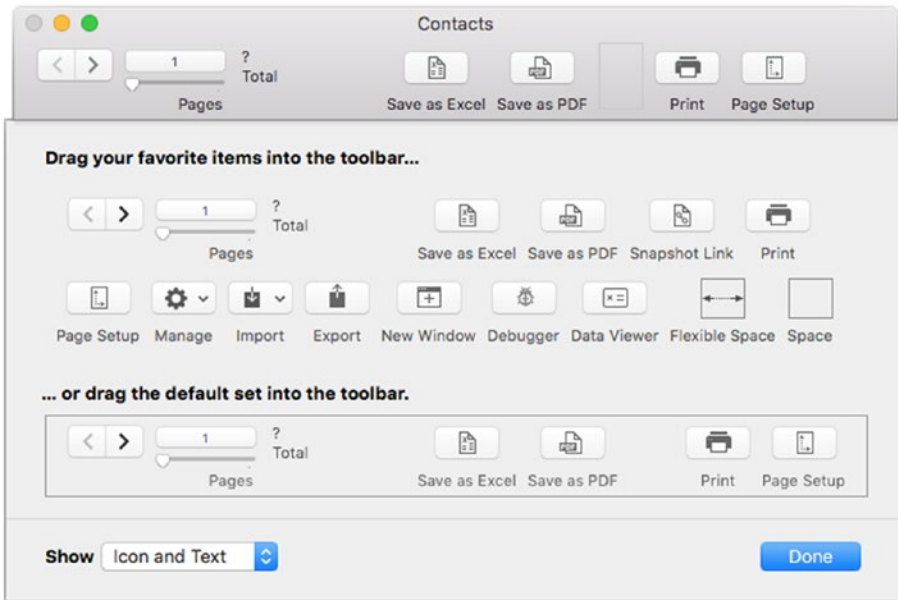


Figure 4-24. The customize toolbar panel attached to the Contacts database window in Preview mode

Print Dialog Options

When ready to print the current window, the Print dialog, shown in Figure 4-25, can be opened by any of these methods:

- Select the File > Print menu item.
- Click the Print button in the toolbar (Preview mode only).
- Run a script or click a button that uses the Print command.

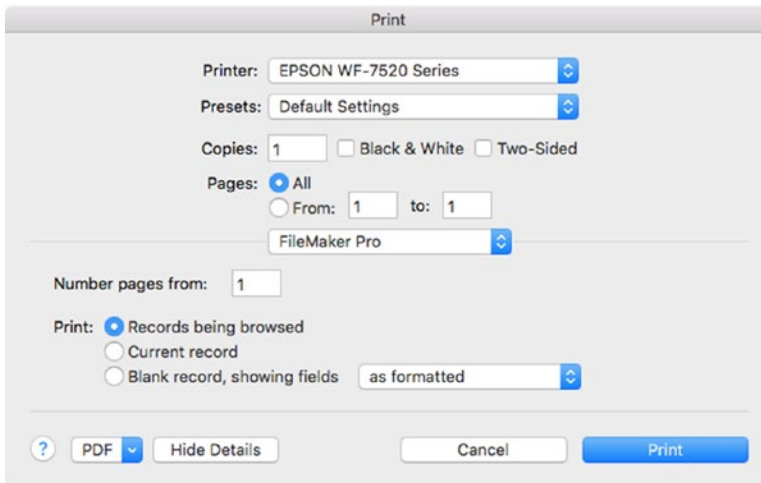


Figure 4-25. The Print dialog showing the FileMaker details

The dialog is a standard operating system dialog and shouldn't need much elaboration. The FileMaker options available are the following:

- **Number pages from** — Enter a number in this field to indicate the page number of the first page.
- **Print: Records being browsed** — Selecting this radio button will indicate that every record in the found set should be included in the printout.
- **Print: Current record** — Selecting this radio button will indicate that only the current record should be included in the printout.
- **Print: Blank record, showing fields** — Selecting this radio button will indicate that the *layout* should be printed with record data. The adjacent pop-up menu allows the selection of formatting options for fields: *as formatted*, *with boxes*, *with underlines*, or *with placeholder text*.

Clicking the **Print** button will perform the print operation as configured in the dialog and close the dialog.

Summary

In this chapter, we discussed numerous subjects pertaining to how a user interacts with records.

CHAPTER 5



Transferring Data

FileMaker supports many file types and offers various options when transferring data in or out of a database. These functions can all be performed manually or automatically with scripts (Chapter 35).

In this chapter, we are concerned only with importing and exporting manually.

Importing and Exporting Records

Data needs to be fluid, with the ability to be transferred, shared, and repurposed. There are many reasons one might need to import or export records into or out of a database, including the following:

- Importing records that already exist in another system to avoid retyping.
- Importing records into a newer version of a database system when upgrading.
- Importing records from a template table for use in a content table.
- Importing records from one copy of a database to another to synchronize them.
- Exporting records to share with coworkers or clients.
- Exporting records for use in other programs without retyping.

Supported File Types

FileMaker can exchange data into the ten file types listed in Table 5-1.

Table 5-1. *The file types FileMaker supports for exporting and importing data*

File Type	Extension(s)	Export	Import
Tab-Separated Text	.tab, .txt	Yes	Yes
Comma-Separated Text	.csv .txt	Yes	Yes
dBase	.dbf	No	Yes
DBF	.dbf	Yes	No
Merge	.mer	Yes	Yes
HTML Table	.htm	Yes	No
FileMaker Pro	.fmp12	Yes	Yes
XML	—	Yes	No
Excel Workbooks	.xlsx	Yes	Yes
Excel 95-2004 Workbooks	.xls	No	Yes

In addition to the file types above, FileMaker supports *importing* from several different sources including:

- A folder of pictures, movies, or text.
- A XML file contain data with an accompanying XSL style sheet.
- An ODBC data source.

Downloading Sample Data

One of the many great resources on the Brian Dunning website is free sample contact data that can be imported into a database for testing purposes. Before moving forward, take a moment to download a sample contacts file that we will use to import and export with the Contacts starter solution that we have been working with.

1. Launch your favorite web browser.
2. Open — <https://www.briandunning.com/sample-data/>.
3. Locate the link to download 500 U.S. records, shown in Figure 5-1.
4. Find the us-500.csv file on your computer. In macOS, it should be in the ~/Downloads folder.
5. Drag the file to your desktop.

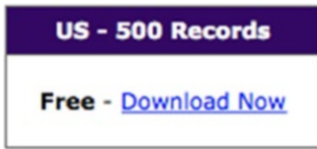


Figure 5-1. Download link for free sample data for testing from *briandunning.com*

The file should open in Excel and display 500 rows of sample data in the following twelve columns:

- First_name
- Last_name
- Company_name
- Address
- City
- County
- State
- Zip
- Phone1
- Phone2
- Email
- Web

Importing Records

The **Import Records** process will create or update records in the current layout's table from a file of any of the supported type listed at the start of this chapter.

To begin an import:

1. Open the database into which records will be imported. For the examples in this chapter, open the **Contacts** database we have been working with.
2. Navigate to the desired layout into which the records will be imported. Since the **Contacts** database has only one table, any layout is fine.
3. Select the **File > Import Records > File** menu item to open a **Choose File** dialog, as shown in [Figure 5-2](#).

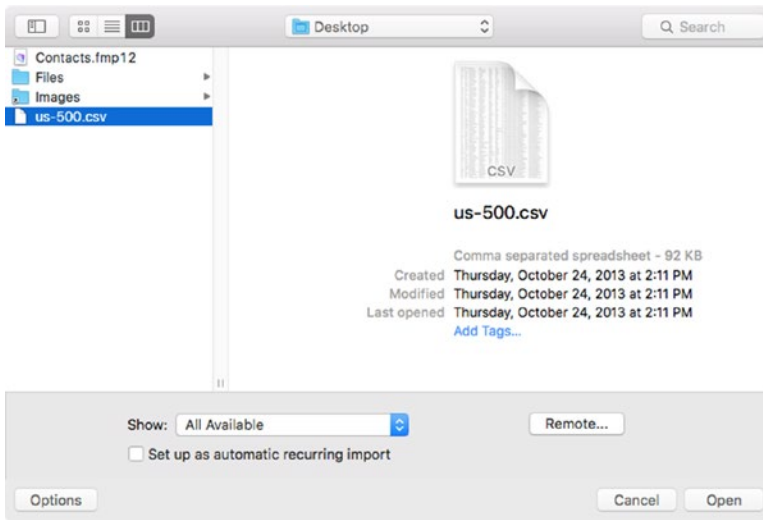


Figure 5-2. The import records Choose File dialog

There are two options available in the footer of this dialog:

- Show — select a specific file type to make it easier to locate a file from within a crowded folder.
- Set up as automatic recurring import — check this box to route the import process to an alternate dialog described next.

Setting Up an Automatic Recurring Import

Selecting the optional Set up as automatic recurring import check box and clicking the Open button will open the Recurring Import Setup dialog shown in Figure 5-3 that completely bypasses the rest of the import process. This feature provides a convenient way to quickly create a new table, layout and script that can be used to periodically update the data from the external file.

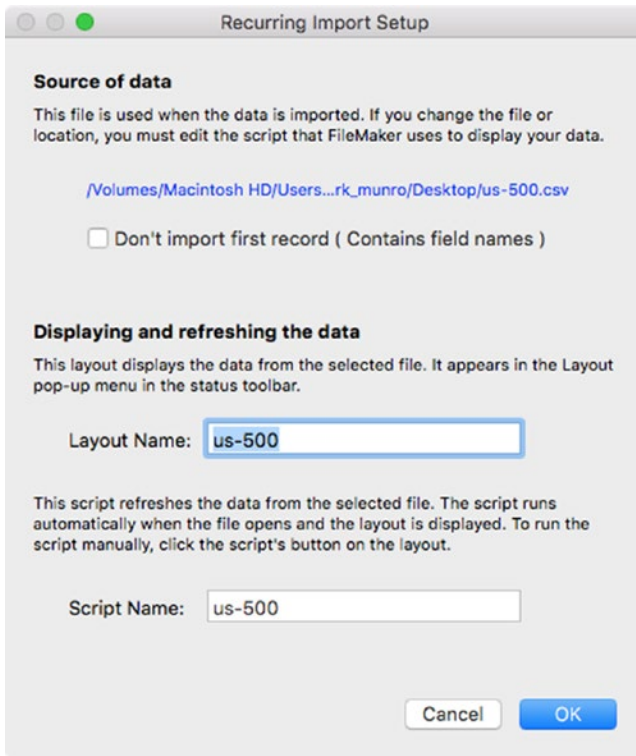


Figure 5-3. *The Recurring Import Setup dialog*

Once the dialog is configured and the OK button is clicked, FileMaker will perform these functions automatically:

- Create a new table named the same as the selected incoming file.
- Create a field for every column of data in that file.
- Create a layout with the specified name, assigned to that table, set to table view and with every field displayed.
- Create a script with the specified name that will go to that layout, *delete every record*, and import the records from the file thereby refreshing the data.

This dialog remembers the path of the source data and has three configurable options:

- **Don't import first record** — If the first row of data in the source file contains field names instead of data content, check this box.
- **Layout Name** — specify the name for the new layout that will be created. The default will be the name of the source file.
- **Script Name** — specify a name for the new script that will be created. The default will be the name of the source file.

When this option is used and the import finishes, you will need to rename fields, reconfigure the layout, and customize the script to suit your needs and preferences.

Performing an Import

To continue with a manual import, point the Choose File dialog at the `us-500.csv` file on the Desktop and click the Open button. This will open the Import Field Mapping dialog, shown in Figure 5-4.

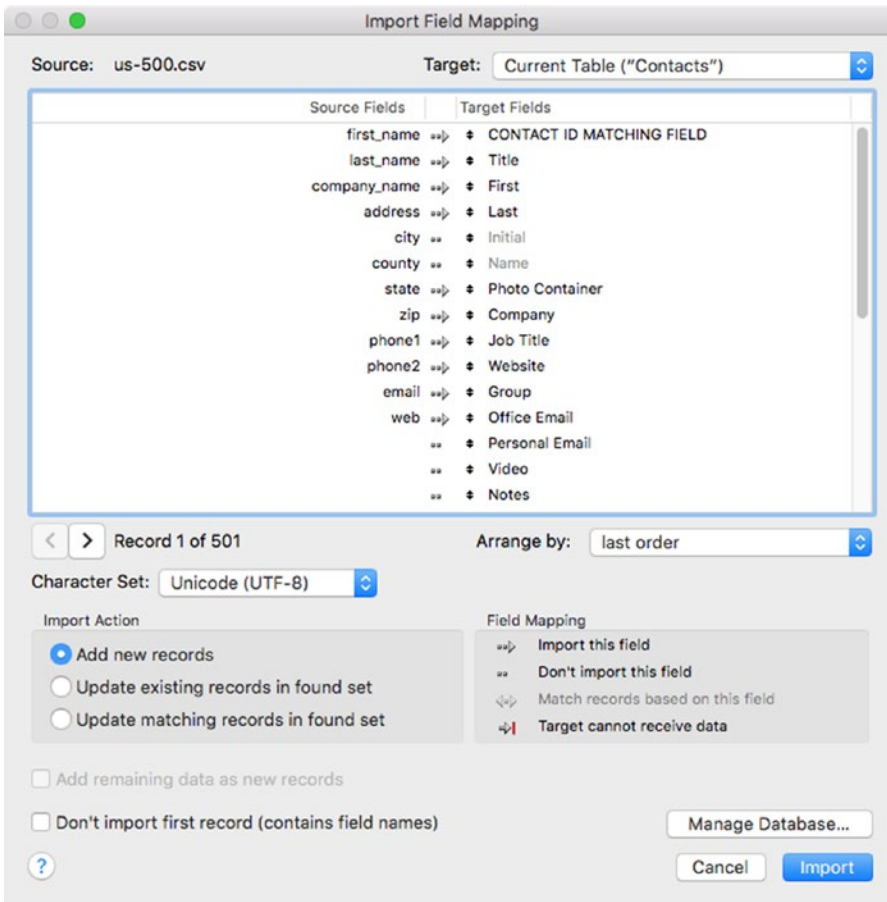


Figure 5-4. The import field-mapping dialog

The *field-mapping dialog* is used to instruct FileMaker where the incoming data should flow. The dialog allows the selection of a table and control over various import behaviors.

Browsing the Source Records

By default, the dialog displays the first row of data as a vertical list on the left under the “source fields” heading. Since the first row of a data file often contains the field names, it helps when mapping fields for import. However, the dialog has actually loaded the entire source file, and additional records can be viewed in order by clicking the left and right arrows on the left just below the field lists. This can be quite useful if the source field names aren’t completely clear about what content is contained within or if the first record begins with data and has some empty fields. Clicking through the list of incoming records provides a quick way to figure out what content is contained in which position directly it is from within the dialog.

Selecting a Target Table

The *target table* is the table into which the incoming data will be imported. The table whose records are displayed in the window will be the default selection in the Target pop-up menu at the upper right corner of the import dialog, shown in Figure 5-5.

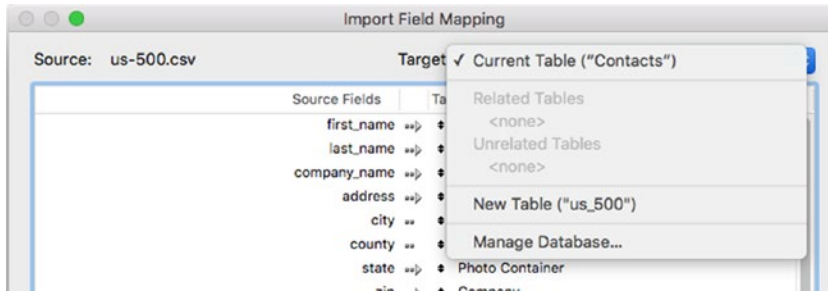


Figure 5-5. The import target table pop-up menu

While this universal menu will list other related and unrelated tables that exist in the database, these options will remain disabled since you can only import into the current layout's table (unless you use the option to create a new table).

The target menu contains two additional options:

- **New Table** — selecting this option will configure the dialog to automatically create a new table during the import and flow the data into it. The field names will default to a sequential generic list, f1, f2, f3, etc. Unless you check the Don't import first record check box on the lower left indicating that the values of the first incoming record should be used as the field names.
- **Manage Database** — selecting this option or clicking the button of the same name at the bottom of the dialog will open the developer dialog for defining tables, fields and relationships (Chapters 7-9). This allows you to create a new table with custom field names or to add fields to an existing table. Unfortunately, any new table you create will not be available from the menu unless you cancel the import process, create a layout for the table, and restart the process while that table is current. But this can be a great way to quickly add a new field to the current table.

Mapping Fields

The central part of the import dialog is used to align fields from the source file on the left with fields from the database on the right.

The Arrange By pop-up menu located below the field list, shown in Figure 5-6, is used to automatically reorder the list of target fields to any of the predefined list of options. These options can sometimes help to quickly match fields by name when the lists are more similarly matched than our example. In the case of our sample data, they do not.

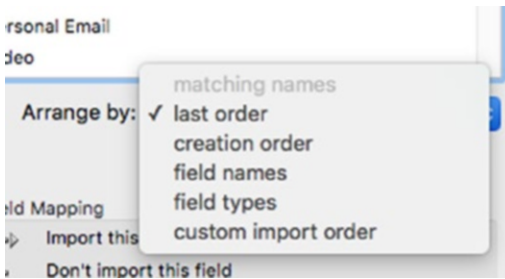


Figure 5-6. The options available for arranging the list of target fields

When the two lists of field names are not closely matched, target fields must be dragged to align them correctly with their corresponding source field. The target fields can be dragged up or down by grabbing the small black two-sided arrow icon just to the left of the field name, shown in Figure 5-7.



Figure 5-7. The two-sided arrow icon is used to drag a target field up or down in the list

When a field is dropped somewhere in the list, the two fields — the one dragged and the one it is dragged onto — exchange positions. Continue dragging the fields until they align to the correct fields, shown in Figure 5-8.

Source Fields	Target Fields
first_name ↔▶	⚡ First
last_name ↔▶	⚡ Last
company_name ↔▶	⚡ Company
address ↔▶	⚡ Work Address 1
city ↔▶	⚡ Work City
county ↔▶	⚡ Name
state ↔▶	⚡ Work State
zip ↔▶	⚡ Work Postal Code
phone1 ↔▶	⚡ Office Phone
phone2 ↔▶	⚡ Mobile
email ↔▶	⚡ Office Email
web ↔▶	⚡ Website

Figure 5-8. A final arrangement of properly aligned source and target fields

The icon in the middle column indicates how the fields are mapped to each other and, therefore, how they will behave during the import. There are four possible icons that will appear here depending on the import action selected. Each is labeled in the **Field Mapping** key on the right of the dialog layout. For the default **Add new records import** action, the icon should be the **Import this field** arrow shown in [Figure 5-9](#).



Figure 5-9. *The Import this field icon*

If you see two square dots as shown in [Figure 5-10](#), this is the **Don't import this field** icon and indicates that the adjacent field will not be imported. To include it, click the icon to toggle it until it displays the arrow.



Figure 5-10. *The Don't import this field icon*

A target field displayed with gray coloring indicates a field that can't receive data, such as a calculation or summary field. Clicking the icon for these will toggle to the **Target cannot receive data** icon shown in [Figure 5-11](#).



Figure 5-11. *The Target cannot receive data icon*

Once the fields are lined up and the icons set, optionally select the **Don't import first record** check box to indicate that records should be imported starting from the second row/record of the source data file. This avoids importing a record containing only field names with no real data, which would need to be deleted.

When ready to proceed, click the **Import** button.

Setting Import Options

The Import Options dialog, shown in Figure 5-12, is the final step in a manual import process.

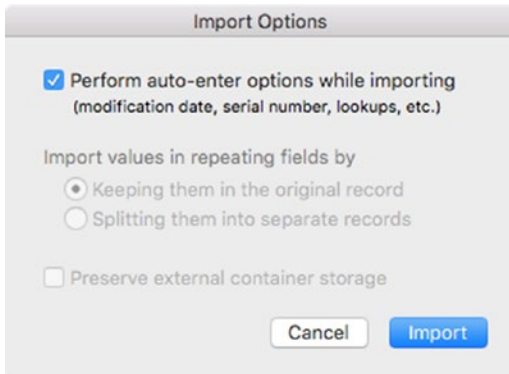


Figure 5-12. The import options dialog

There are three options presented in the dialog:

- Perform auto-enter options while importing — checking this box will cause any fields with auto-enter options defined to be updated as part of the import. For new records this box should be checked. If updating records or importing data with serial numbers or creation and modification dates that should be preserved, this box should be unchecked. For more information on field options, see Chapter 8.
- Import values in repeating fields by — this radio button allows you to choose between keeping repeating fields together in a single record or splitting them into one imported record per repetition during the import. When this latter is selected, all the non-repeating fields will be duplicated for each record generated for one of the repetitions.
- Preserve external container storage — checking this box will suppress container field content validation, allowing the target table to use existing external container contents. This is useful when reimporting data back into an existing file or a copy of a file, avoiding decrypting and re-encrypting external files when the base directory of the source and target fields are the same.

Notification of Results

Once the import process is complete, the Import Summary dialog, shown in Figure 5-13, will appear with a notification of the results including the following:

- Total records added or updated
- Total records skipped because of errors
- Total fields skipped because of errors
- Name of a table created when the New Table option was selected

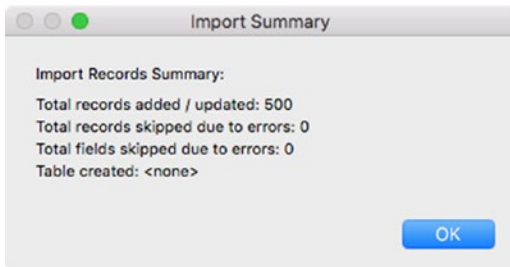


Figure 5-13. The *Import Summary of results dialog* after a successful import of the sample contact data

A text file named `Import.log` will be saved to the desktop containing details about the import process and noting any errors that may have occurred.

Resulting Found Set

After the import process, the found set in the window will be that of the imported data only.

Performing an Update Import

Besides the `Add new records` import option used above, the following two import options perform a slightly different process:

- Update existing records in found set
- Update matching records in found set

Each of these is available only when there are existing records in the target table before the import process begins because they are used to overwrite existing records rather than create new ones.

Update Existing Records in Found Set

The `Update existing records in found set` option will import the source records by overwriting matched fields on existing records starting from the first record in the found set.

This can be useful in a situation where you need to export a set of data, manipulate it outside of FileMaker, and then replace some or all field values of those same records by means of the import process. FileMaker expects the source data and the target table's current found set to contain the same number of records. If this is not the case, one of two warnings will be presented to the user:

- If the source data record count is *more* than the target table's found set, the import will stop after the number of records in that found set are finished, thereby excluding the remaining source records.
- If the source data record count is *less* than the target table's found set, records in the found set beyond that number will not be modified.

When this action is selected, the `Add remaining data as new records` check box will become enabled. If checked, any records in the source data beyond the number in the target table's found set will be imported as new records. After the import, the found set will remain unchanged unless this box was checked, in which case, the found set will contain the additional new records.

When finished, a dialog, like that in Figure 5-12, will notify of the success or failure of the process.

Update Matching Records in Found Set

The **Update matching records in found set** option will import each record of the source data into a record based on one or more matching fields.

A *matching field* is denoted in the field list by a double arrow icon, shown in Figure 5-14, which instructs FileMaker to look for records in the target table where the field matches the corresponding value in the source table and to update the remaining fields in that record. So, instead of simply updating records in linear fashion like the previous action, this action dynamically searches for records with the same value in match field(s), updating them in a nonlinear fashion.



Figure 5-14. The double arrow icon indicates a pair of matching fields that will be used when updating records on import

To denote match fields, click the icon between two fields to toggle between three choices; import (arrow), don't import (two dots), and match (double arrow).

When importing based on matching fields, selecting the **Add remaining data as new records** check box will cause any records in the source data that can't establish a match to be imported as new records. After the import, the found set will remain unchanged unless this box was checked, in which case, the found set may contain the additional new records.

When finished, a dialog, like that in Figure 5-12, will notify of the success or failure of the process.

Exporting Records

The **Export Records** process will save one or more selected fields from every record in the current found set of the front window into a file of any of the supported types listed at the start of this chapter.

To begin an export:

1. Open the database containing the records to be imported. For the examples in this chapter, open the Contacts database we have been working with.
2. Navigate to the desired layout containing the records to be exported. To follow these examples, switch to the Desktop > Contacts layout.
3. Perform a **Find** or **Find All** to establish a found set containing the records to be exported.
4. Select the **File > Export Records** menu item to open a **Save File** dialog, as shown in Figure 5-15.

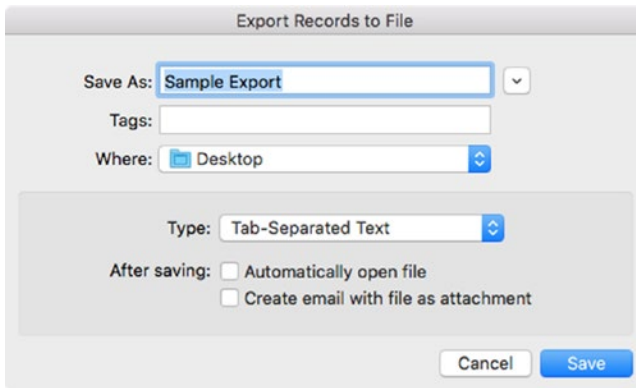


Figure 5-15. A save file dialog, configured to save a tab-separated text sample export file

In addition to the standard save options such as name and location, the following options can be specified:

- **Type** — choose from any of the available file types.
- **Automatically open file** — check this box to have the exported file automatically open after saving.
- **Create email with file as attachment** — check this box to have the exported file automatically attached to a new email on your default mail application.

Once the dialog options are configured, click the Save button.

Specifying Export Fields

Next, the Specify Field Order for Export dialog will appear, as shown in Figure 5-16. This dialog is used to select the fields that will be exported, the order they will be saved in the output file, and some formatting options.

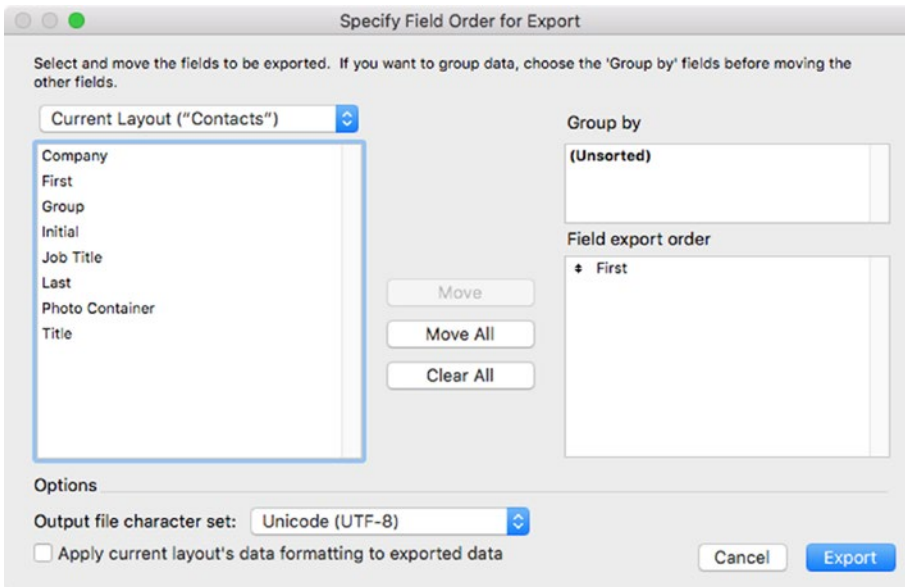


Figure 5-16. The specify field order for Export dialog

The controls at the top of the dialog are used to specify which fields should be exported. On the left side, a field picker contains a popup menu from which a layout or table can be selected. The list below it displays the fields available for export from that selection. On the right is a list of fields that will be exported. In the middle are buttons to add or remove fields from the export list.

Selecting a Table

The pop-up menu above the field list has the current layout selected by default. This is apparently done as a convenience to narrow the list of available fields to just those visible on the layout you are viewing. While this can sometimes be a benefit, it can also be error-prone if you hastily click the `Move All` button thinking you are selecting all fields for the *table* when you are not. Selecting “Current Table” from the menu will refresh the list of fields to show all fields for the Contacts table.

Multi-table database files will include a list of related and non-related tables in this menu. While it is possible to include a mixture of fields from the local table and fields from other related tables, this can have unexpected results so experimentation is advisable.

Adding Fields to the Export Order

Fields can be added to the export order by any of the following methods:

- Double-clicking a field from the available field list will add it to the export list.
- Clicking the Move button will add the currently selected field(s) to the export list.
- Clicking the Move All button will add all the fields to the export list.

Removing Fields to the Export Order

Fields can be removed from the export order by any of the following methods:

- Double-clicking a field in the export list will remove it.
- Typing the Delete key on the keyboard will remove any selected field(s) from the export list.
- Clicking the Clear button will remove any selected field(s) from the export list.
- Clicking the Clear All button will remove all fields from the export list.

Reordering Fields in the Export List

The fields in the export list will be saved in the output file in the order they appear. Drag and drop the fields within the export list until they are in the desired order.

Setting Up the Sample Export

Since our sample data only populated a portion of the fields in the Contacts database, we will set up the export that includes only the following fields, in the order shown in Figure 5-17.

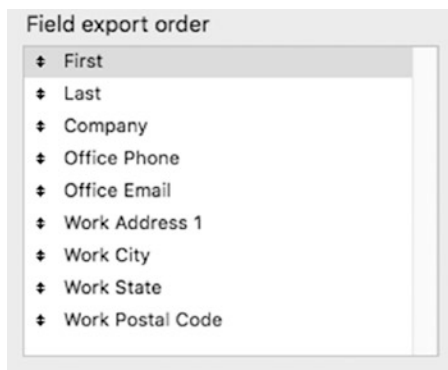


Figure 5-17. The field export order for the sample export

Output File Character Set

The `Output file character set` pop-up menu specifies which character encoding will be used during the export. The selected value will be the default for the file type of the export, but it can be changed as needed.

Apply Current Layout's Data Formatting to Exported Data

If the `Apply current layout's data formatting to exported data` check box is selected, any number, date, or time field will be exported with the data formatted as it is on the current layout rather than the format of the actual data entered.

For example, if a number field contains “10” but is formatted on the current layout as currency “\$10.00,” this box must be checked to export the number formatted as currency. See Chapter 29, “Data Formatting — Decimal” for information about imposing formatting to field contents on a layout.

Group by Option

Although not applicable to our example `Contacts` export, the `Group by` feature allows values to be sub-summarized during the export. If the found set of records is sorted, the `Group By` area of the dialog will contain a list of the sort which can be used to summarize data during the export.

For example, imagine a found set of records with three fields containing the values shown in Table 5-2. The `Quantity Summary` field is a summary that automatically totals the `Quantity` field for the six records (Chapter 8, “Summary Fields”).

Table 5-2. Hypothetical records showing how a non-grouped summary behaves

Industry	Quantity	Quantity Summary
Education	20	95
Education	15	95
Finance	5	95
Finance	10	95
Publishing	35	95
Publishing	10	95

If these three fields were exported, the resulting file would look the same as it does in the example. However, the data can be exported as a summary report of the total quantity for each industry following these steps:

1. Sort the records by the `Industry` field.
2. Begin an export, and when you reach the `Specify Field Order` dialog, the `Group by` area will have a check box for the `Industry` field.
3. Add the `Industry` field to the export list.
4. Add the `Quantity Summary` field to the export list. Two fields will be automatically added: `Quantity Summary` and `Quantity Summary by Industry`. The second one will be italicized, highlighting the fact that this is not an actual field in the database, but a sub-summary of the quantity by industry, as shown in Figure 5-18. For this example, delete the `Quantity Summary` field, leaving just `Industry` and `Quantity Summary by Industry`.

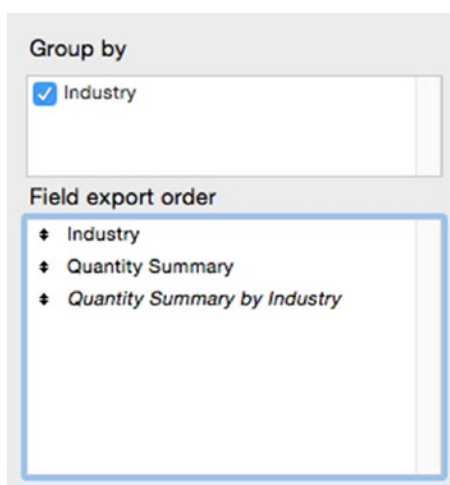


Figure 5-18. An example of a field export order using the group by feature

With this configuration, the result would be one row for each industry, the total quantity for all three industries (only on the first row), and the total of each industry in the third column, as shown in Table 5-3.

Table 5-3. Hypothetical data showing how data is summarized using the Group by option

Education	95	35
Finance		15
Publishing		45

Finishing the Export

Once the field order and other options are configured, click the Export button to finish the process. The exported file will be saved to the location specified.

There is no visual indication of the completion of the process for smaller found sets. If the found set is large enough, a progress dialog will indicate the records remaining. When it disappears, the process is complete.

Exporting Field Contents

The content of any individual field accessible on the current layout can be exported by selecting Export Field Contents from the Edit menu or the field's contextual menu. This will open a Save File dialog, as shown in Figure 5-19, into which a name and location can be specified. The content will then be saved into a file as specified, with a file type automatically determined based on the content.

If the field is a container, the saved file will be the same file type as the content. If the field is any other text-based content, the saved file will be a text file with an extension of .txt.

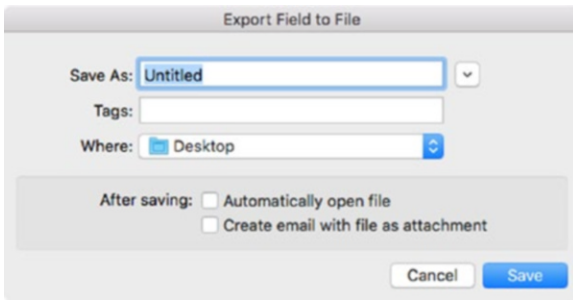


Figure 5-19. The save file dialog for the Export Field Contents process

Summary

In this chapter, we discussed the basics of manually importing and exporting records from a database.

PART II



Defining Data Structures

The basic structure of a database is made up of files, tables, fields, relationships, containers, and value lists. These provide the foundational data structure upon which interfaces and scripts are later built.

6. *Working with Database Files* — Provides an overview of creating, configuring, and maintaining a new database file.
7. *Working with Tables* — Covers the process for creating and managing data tables.
8. *Defining Fields* — Discusses the different types of fields that make up a table and how to define, manage, and configure them.
9. *Connecting Tables with Relationships* — Explores data sources, table occurrences, and interconnecting tables into relationships.
10. *Managing Containers* — Explores the options for file containers and storage options.
11. *Managing Value Lists* — Provides an overview of defining value lists.

CHAPTER 6



Working with Database Files

A *database file* is a type of document that contains all the structural elements you define and the data entry content that users enter.

This chapter covers basic details about creating, configuring, and maintaining a database file, including topics such as the following:

- Creating a new database file
- Configuring a sandbox table
- Database file options
- Maintaining healthy files
- Specifying file paths

Creating a New Database File

With the FileMaker application launched, close any other database windows you may have open to establish an uncluttered workspace and to avoid confusion. With only the application menu bar visible and your computer's desktop behind it, follow these steps to create the new database file:

1. Select the File > New Solution menu.
2. When the New File dialog appears, type the name “Learn FileMaker” into the Save As text area and select the Desktop folder as the save location, as shown in Figure 6-1.

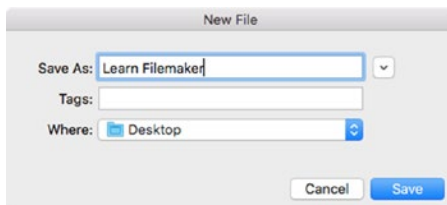


Figure 6-1. The New File dialog, configured to create a Learn FileMaker database on the Desktop

3. Click the Save button or type the Enter key to close the dialog and create the new database.
4. The Learn FileMaker.fmp12 file should appear on the Desktop and automatically open in FileMaker.

Configuring a Sandbox Table

To get a head start with a practical example of creating some basic structural elements, begin by configuring a sandbox table within the new Learn FileMaker database.

In software development, a *sandbox* is a testing environment that provides a realistic workplace for experimentation, safely isolated from a live system containing actual data. While it usually refers to a safe area or mode for working in a non-destructive way within some existing complex system, we will use it to denote a practice area that can be freely used to follow along with early basic examples presented in the forthcoming chapters.

Our sandbox file will perform several crucial functions in learning FileMaker:

- Creating the Sandbox table will provide a quick immersion into building a database from scratch, giving a brief overview of building a basic database including elements that won't be discussed until much later in the book.
- Once these elements are constructed, they will provide an experimental workspace for working through the remaining lessons.
- By providing a basic interface right from the outset, it provides a place in which you can *see* the practical results of more abstract topics of building a data structure.

To elaborate this last point, the material in this book is arranged in a logical hierarchy, with each lesson building on the last. We start with the foundational data structure of tables, fields, and relationships well before learning how to express that structure in a visible user interface. However, without an interface available now, it may become too abstract and difficult to fully grasp the practical implications of some of these early lessons. This is especially true when experimenting with calculation formulas (Chapter 12) with results that are not viewable without an interface. The sandbox will solve this problem by instantly providing a quick and simple data structure, with fields displayed on a basic layout, which can make the results of abstract calculation examples visually perceivable.

Changing the Default Table Name

When any new database file is created, FileMaker will automatically build a default table with the same name as the file. So, your file should have a table named “Learn FileMaker” already set up with no fields. Often this default name will suffice and not require any modification. However, for our purposes and to help gain a little more practical experience, let's rename the default “Learn FileMaker” table to “Sandbox” by following these steps:

1. Depending on your settings, the new file will either open in Layout mode or will open in Browse mode with the Manage Database dialog opened. If in Layout mode, you will need to take these extra steps to open the Manage Database dialog:
 - a. Switch to Browse mode by selecting the View > Browse Mode menu item.
 - b. Open the Manage Database dialog, by either selecting the File > Manage > Database menu item.

2. The default tab displayed in the dialog is usually Fields. Click on the Tables tab to switch the view.
3. Change the name of the default table from “Learn FileMaker” to “Sandbox” by following these steps, shown in Figure 6-2:
 - a. Select the Learn FileMaker table in the table list.
 - b. Type “Sandbox” in the Table Name field.
 - c. Click the Change button.

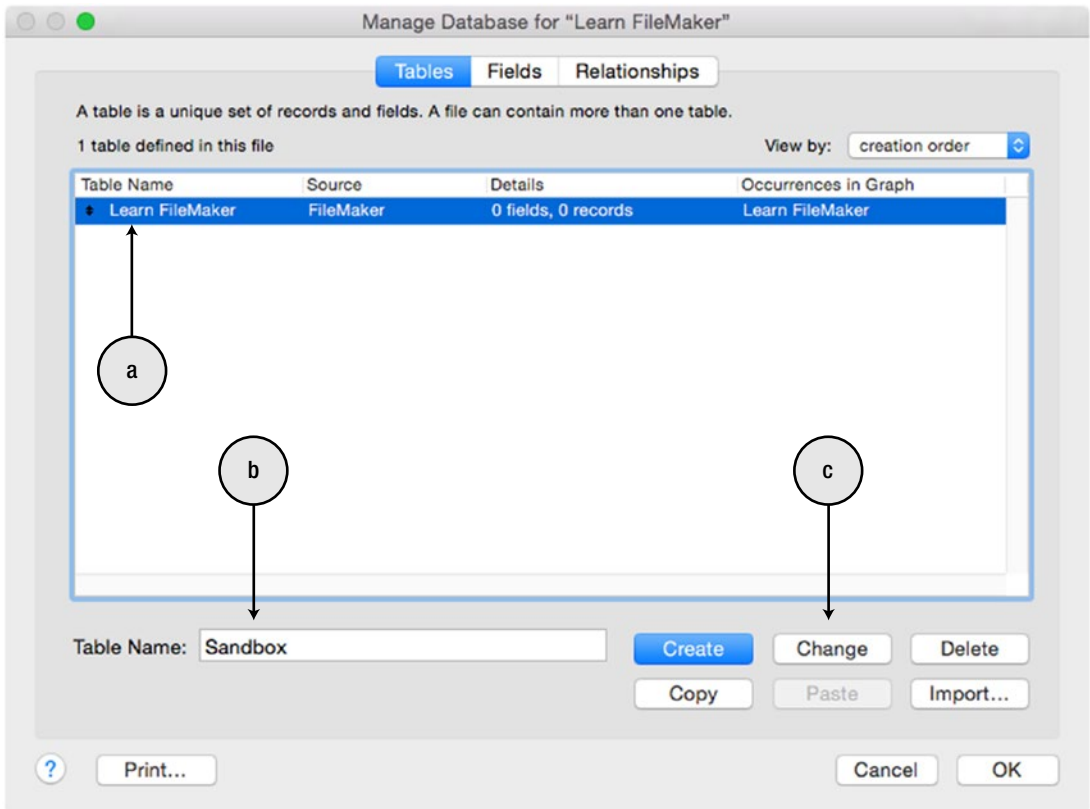


Figure 6-2. Steps highlighted showing the process of renaming a table

Creating Data Entry Fields

In the newly renamed Sandbox table, create one field of each data type to serve as an example for any experimentation you may perform on topics presented in forthcoming chapters. To get started, click on the Fields tab in the Manage Database dialog. Next, create a text field named “Example Text,” by following these steps as shown in Figure 6-3:

1. Type “Example Text” into the Field Name text area.
2. Select “Text” from the Type pop-up menu.
3. Click the Create button or hit either the Return or Enter key.

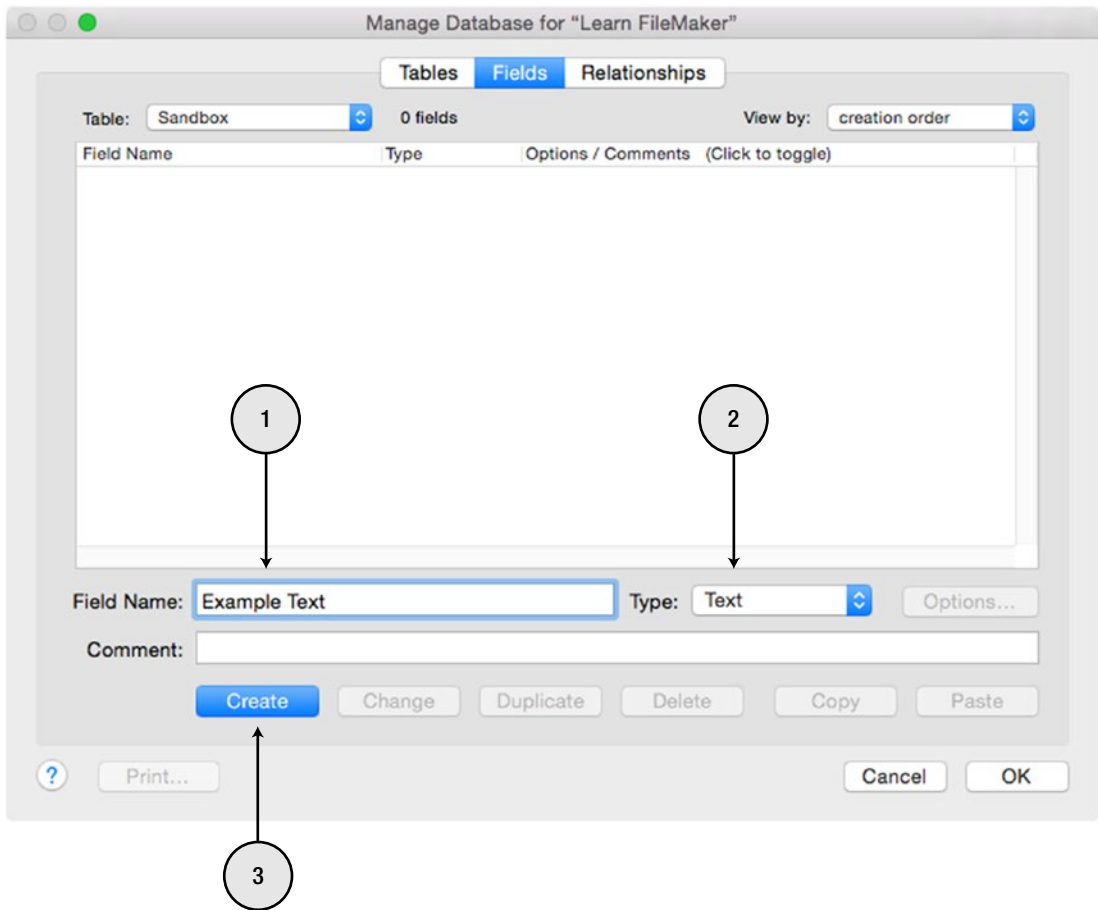


Figure 6-3. Steps highlighted showing the process of creating a text field in the Sandbox table

Follow these same instructions for each of these field types:

1. Create a number field named "Example Number"
2. Create a date field named "Example Date"
3. Create a time field named "Example Time"
4. Create a timestamp field named "Example Timestamp"
5. Create a container field named "Example Container"

For now, just ignore any other field options. These will be covered in depth in Chapter 8.

Creating a Calculation Field

Next, create a calculation field named “Example Calculation” by following these steps:

1. Type “Example Calculation” into the Field Name field.
2. Select “Calculation” from the Type pop-up menu.
3. Click the Create button.

When the Specify Calculation dialog appears, follow these steps shown in Figure 6-4 to enter a temporary calculation formula of 0:

1. Type a 0 in the text area; we will learn about writing calculation formulas in Chapter 12.
2. Select Number for a result type.
3. Click the OK button to close and save.

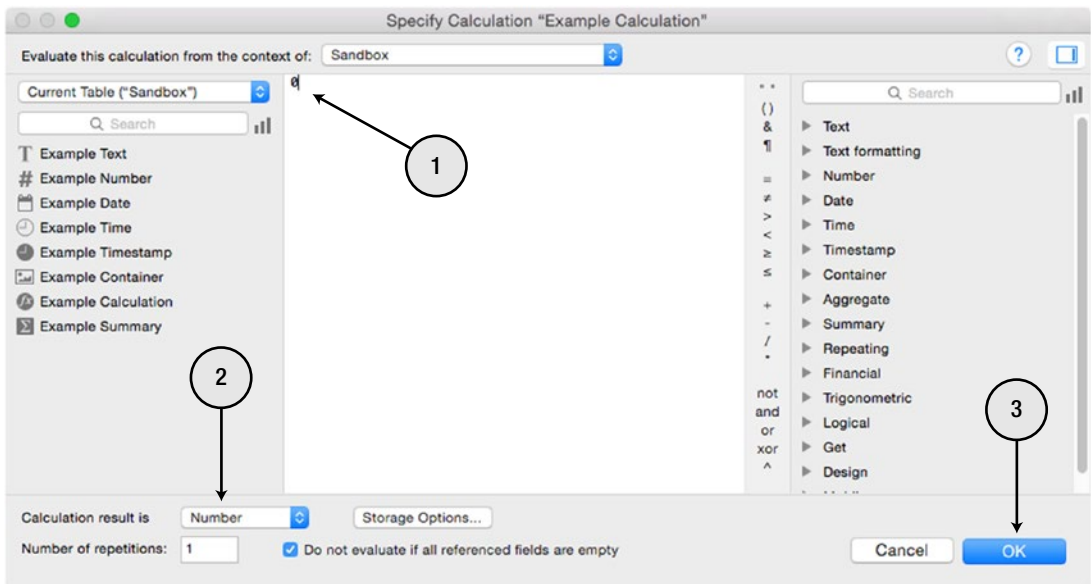


Figure 6-4. Steps highlighted showing the process of creating a calculation field in the Sandbox table

For now, just ignore the rest of the dialog controls and features. These will be covered in depth in Chapters 12-24.

Creating a Summary Field

Create a summary field named “Example Summary” by following these steps:

1. Type “Example Summary” into the Field Name field.
2. Select “Summary” from the Type pop-up menu.
3. Click the Create button.

When the Options for Summary Field dialog appears, follow these steps shown in Figure 6-5 to configure its settings:

1. Click the Total of radio button.
2. Select the Example Number field from the Available Fields list.
3. Click the OK button to save the Options for Summary Field dialog.

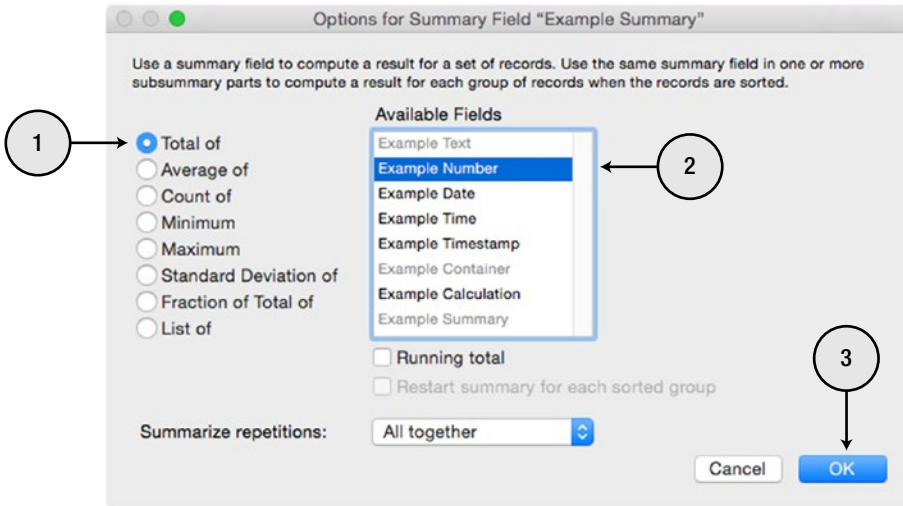


Figure 6-5. Steps highlighted showing the process of creating a summary field in the Sandbox table

For now, just ignore any other summary options. These will be covered in depth in Chapter 8. Now the Sandbox table should contain the eight fields shown in Figure 6-6.

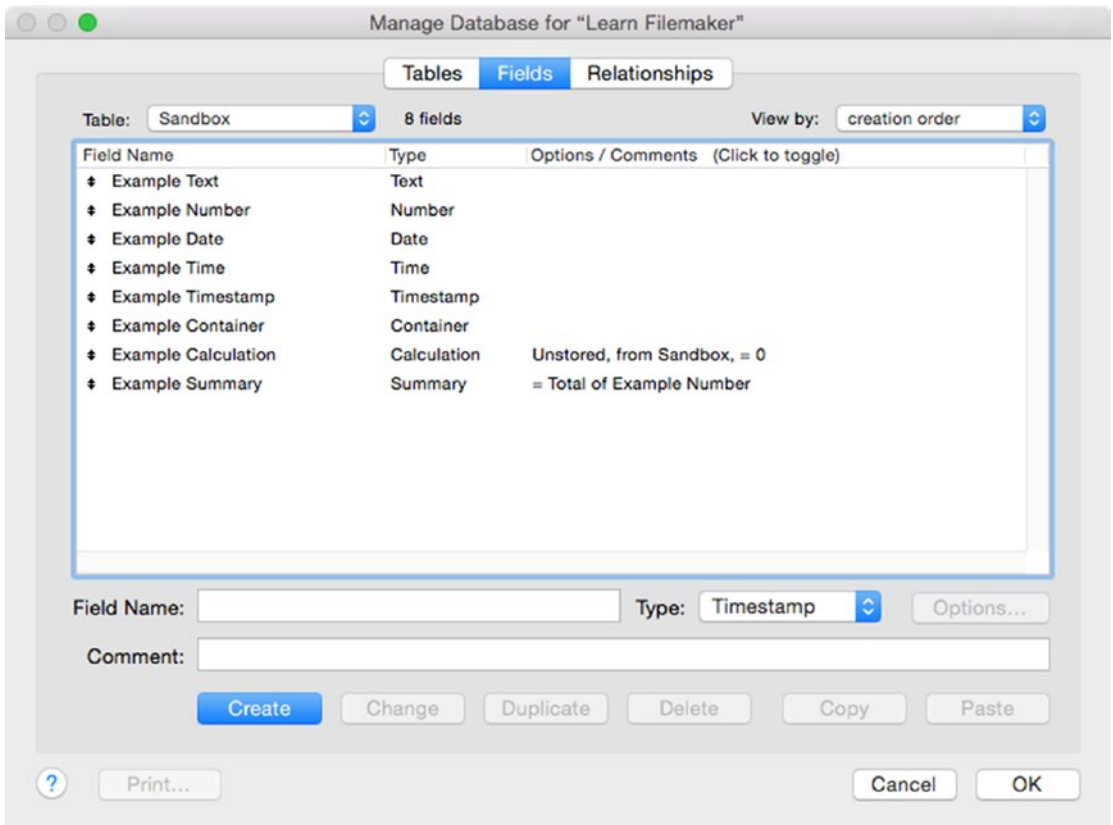


Figure 6-6. The complete list of example fields

Click the OK button to close the Manage Database dialog and save all changes made to the Sandbox table.

Setting Up Layouts

Although some tables will have more, most have at least two layouts. Let's quickly create these, each with a different content view specialized for a specific purpose:

- A *Form* layout is used for viewing and performing data entry tasks.
- A *List* layout is used for viewing and scrolling through many records, navigating to a Form layout, and for batch processing data entry tasks.

Renaming the Default Layout

Every new database file will automatically contain a single empty Form view layout with the same name as the default table. So, initially there would have been a single layout automatically created in your file named "Learn FileMaker." However, when we changed the name of the table to "Sandbox," the name of that default layout should have changed as well. So, if you have followed the instructions in this section so far, there should be a single empty Form view layout named "Sandbox." To help denote the function

of the layout and to distinguish it from the other layout we will construct, rename it “Sandbox Form” by following these steps:

1. Enter Layout mode by selecting the View > Layout menu item.
2. Open the Layout Setup dialog by selecting the Layout > Layout Setup menu item.
3. In the Layout Name text area on the General tab, shown in Figure 6-7, change the name to “Sandbox Form.”

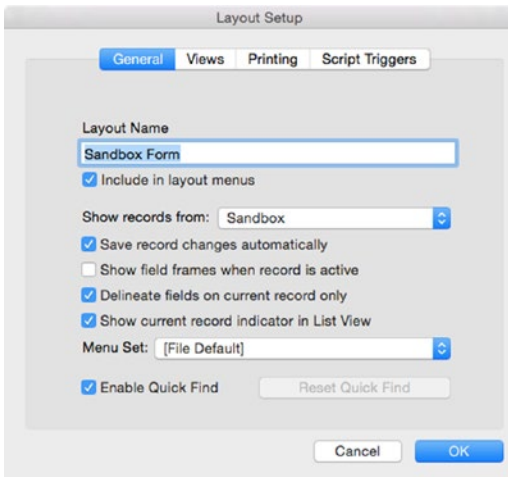


Figure 6-7. The Layout Setup dialog with the new layout name entered

4. Click the OK button to close the window and save the change.

Ignore the other settings in this dialog and other layout features. These will be covered in more detail in Chapter 28.

Adding Fields to a Layout

There are three ways to add fields to a layout (Chapter 30). We will use the Field Picker for now since it makes it quick and easy to add multiple fields to a layout at once in a neat and orderly fashion.

While still in Layout mode, add all eight fields to the layout by following these steps:

1. Select the Field Picker by either selecting the View > Field Picker menu item.
2. Select the entire list of all eight fields, shown in Figure 6-8, by clicking in the field list and then selecting the Edit ► Select All menu item.

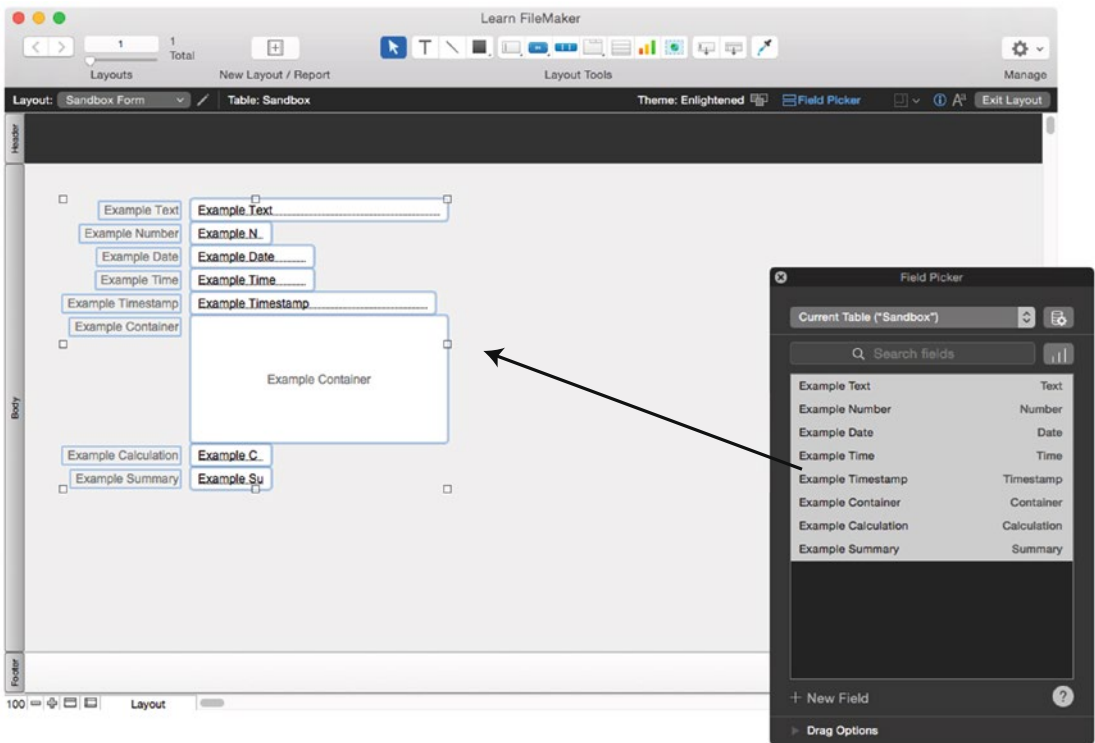


Figure 6-8. Dragging fields from the *Field Picker* dialog onto the layout

3. Click, hold, and drag the list of fields onto the layout and then release. The fields should appear on the layout, neatly aligned and sized according to their data type.

Creating a List Layout

With a Form layout finished, we next need to add a second layout for viewing lists of records. While still in Layout mode, select the Layout > New Layout/Report menu item to open the New Layout/Report dialog, shown in Figure 6-9.

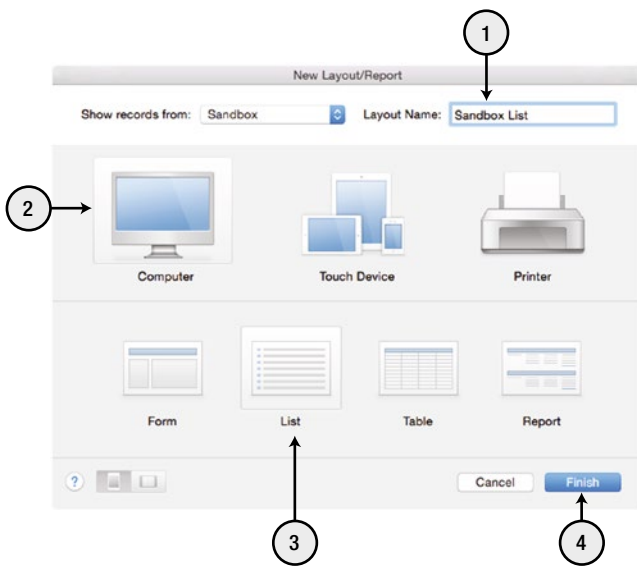


Figure 6-9. *The New Layout/Report dialog window*

Then follow these steps:

1. Enter “Sandbox List” into the Layout Name text area.
2. Click to select a layout type of Computer in the first row of option icons.
3. Click a computer type of List in the second row of option icons.
4. Click the Finish button to close the dialog and complete the process. An empty list layout should appear with three parts: Header, Body, and Footer.

Ignore the other settings in this dialog. These will be covered in more detail in Chapter 28.

Due to the vertical space constraints of a List view, we will change the process of adding fields slightly, following the steps shown in Figure 6-10:

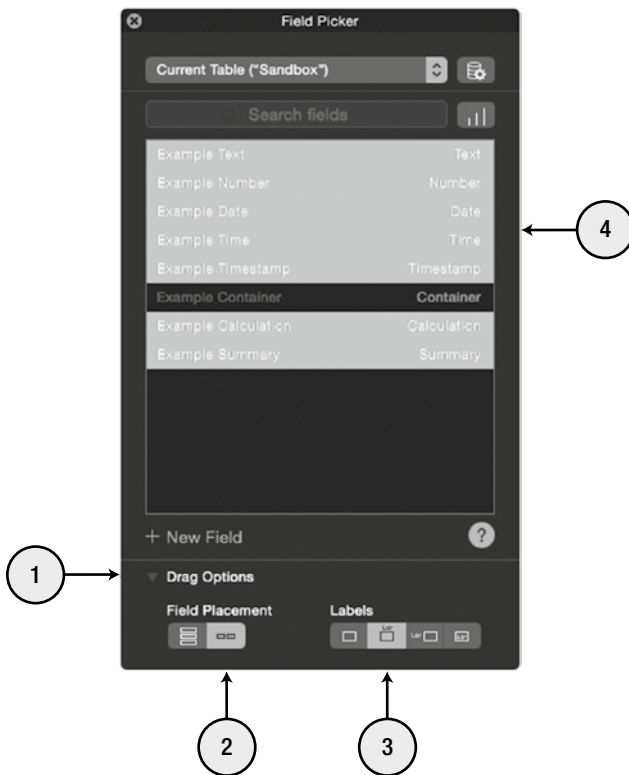


Figure 6-10. The expanded Field Picker, configured to place fields onto the list layout

1. In the Field Picker dialog, click the Drag Options disclosure triangle to reveal additional controls at the bottom.
2. Select the Horizontal option icon for Field Placement.
3. Select the Place labels above fields icon for the Labels option.
4. Select every field except the container field by clicking on each field while holding Command (macOS) or Windows (Windows). We'll skip the container field on the List view since it will default to a large vertical footprint and attempt to expand the layout.
5. Next drag the selected fields onto the layout. Be sure to place them high enough so you don't receive a dialog asking if you want to expand the layout space.
6. Once placed, drag them down so that the fields are roughly centered in the Body part, as shown in Figure 6-11.

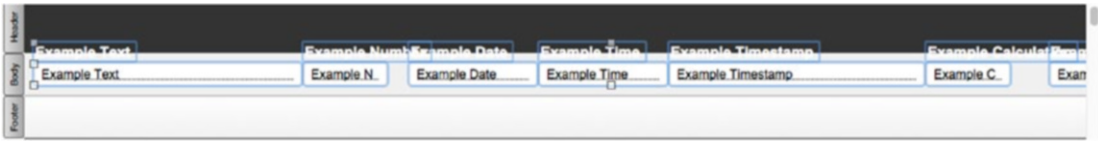


Figure 6-11. The fields placed in the Body part of the layout

7. Select only the field labels by clicking and dragging the cursor around them. Then move them up slightly into the Header area of the layout, as shown in Figure 6-12. You can move selected objects on a layout with the mouse or arrow keys.

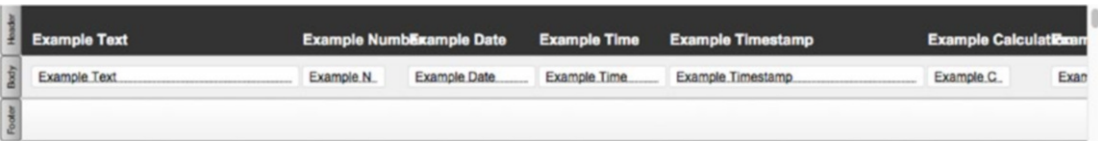


Figure 6-12. The field labels moved up into the Header area

8. To finish up, remove the repetitive “Example” portion of the field labels. To edit a text label, double-click the cursor to activate it, then select and delete the extraneous word. When finished, click on the background and repeat with the other labels.
9. Finally, scroll to the right and adjust the size and position of the last two fields so nothing extends into the gray area beyond what will be the visible layout, as shown in Figure 6-13.

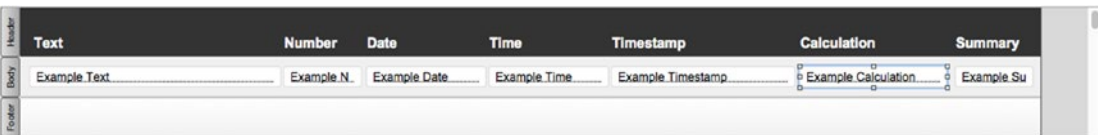


Figure 6-13. The fields with cleaned-up labels and adjusted so none extend too far to the right into the gray area

Database File Options

FileMaker has several preference options that control various behaviors of an open database. Unlike *application preferences*, these *file options* are accessible from the File menu only when a database is open. When creating a new database, default values are assigned that may or may not require modification to reflect your desired settings.

To open the file options dialog, select the File > File Options menu item. File options are separated into five tabs:

- Open
- Icon
- Spelling
- Text
- Script triggers

Open Settings

The *Open* settings tab, shown in Figure 6-14, contains settings that control behaviors and actions that take place when a database file is opened.

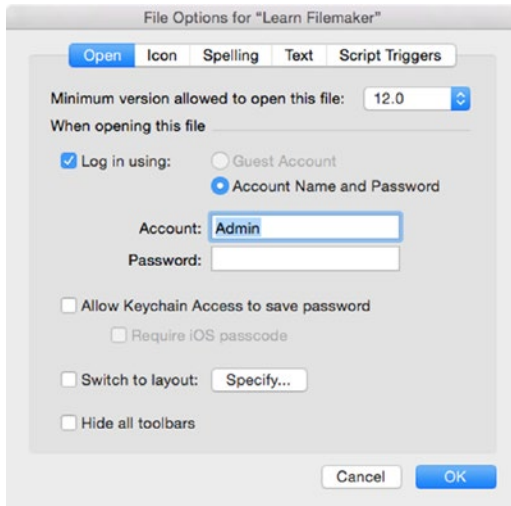


Figure 6-14. The *Open* settings tab of the *File Options* dialog

Minimum Version Allowed to Open This File

The *Minimum version allowed to open this file* setting allows selection of a FileMaker version between 12 and 16 that the database requires as a minimum. Since these versions share a file format, the setting allows you to stop a user from opening a database that uses newer features in an older version of the application. If you regularly use newer features and do not add conditional support for older versions of FileMaker, it is a good idea to keep this set to the latest version.

When Opening This Field

- **Log In Using** — This setting allows a database to automatically open with a specific default account and password. This eases user access to a file that doesn't need multilevel security while maintaining protection over the programming. Holding Option (macOS) or Shift (Windows) when opening a database will bypass this setting and open a standard password prompt instead (Chapter 40).
- **Allow Keychain Access to save passwords** — This setting allows a user the option to store their account and password in their computer's local repository, such as the macOS Keychain, for automatic entry into the password dialog during future log-in attempts. This should not be enabled for databases with multilevel security in an environment where users might share a computer or where tight security is an important consideration. The secondary setting allows the requirement of a device passcode on iOS devices even when the database password is automatically entered.
- **Switch to layout** — This setting allows a database to automatically switch to a specific layout when opened by any user. Script Triggers can also accomplish this task with greater variability based on the identity of the user or on other contextual circumstances (Chapter 37).

- **Hide all toolbars** — This setting will hide the toolbar and formatting bar automatically each time the file is opened, allowing the user to manually show them again as needed. A script that uses the `Toggle Toolbar` script step can also perform this function with greater variability at startup or in response to other events.

■ **Tip** The `Switch to layout` preference will be executed preceding all Script Triggers in the file's launch sequence. Use this setting to instantly open a splash screen or menu while avoiding screen refresh “flashes,” which can be a distraction to users.

Icon Settings

The *Icon* settings tab, shown in Figure 6-15, allows an icon to be assigned that will represent the database file in the Launch Center in FileMaker Pro or FileMaker Go. The default icon is the FileMaker document icon.

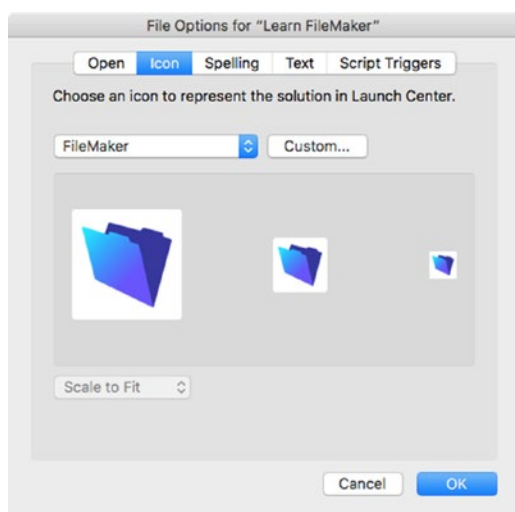


Figure 6-15. The *Icon* settings tab of the *File Options* dialog

The assigned icon can be either:

- Selected from the library of built-in icons provided by the FileMaker app, which are accessible in the pop-up menu.
- A custom 288 pixels square PNG or JPEG file, which can be selected through an Open File dialog accessible by clicking the `Custom` button.

■ **Note** The assigned icon does *not* affect the file icon on a macOS or Windows desktop, only how it appears in FileMaker Go.

Spelling Settings

The *Spelling* settings tab, shown in Figure 6-16, contains settings that control automatic spell check within a database file.

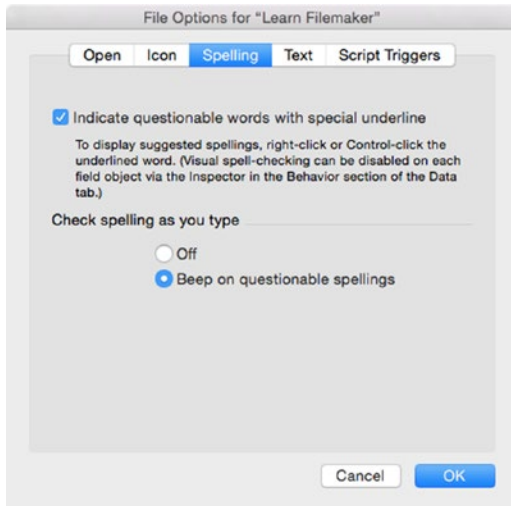


Figure 6-16. The *Spelling* settings tab of the *File Options* dialog

- Indicate questionable words with special underline — This setting will cause FileMaker to display a red dotted line under any potentially misspelled words in a field with focus unless that field or object has been configured in the Inspector to disable the feature (Chapter 29).
- Check spelling as you type — When Beep on questionable spellings is selected, a beep will sound anytime a user types a word that the dictionary indicates might be misspelled.

Text Settings

The *Text* settings tab, shown in Figure 6-17, contains settings that control how text is entered in fields.

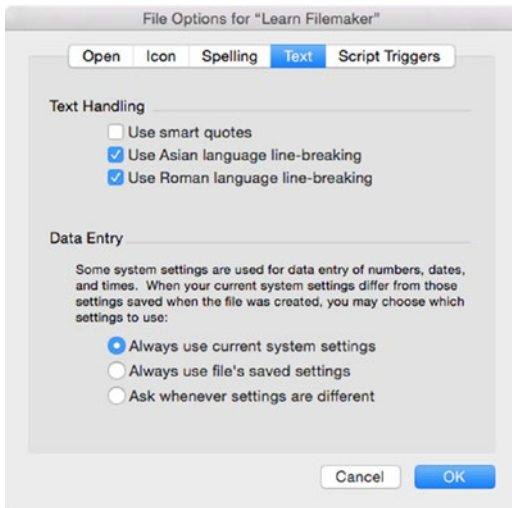


Figure 6-17. The Text settings tab of the File Options dialog

Text Handling

The following settings control various text handling functions:

- Use smart quotes — This setting will cause FileMaker to enter apostrophes and double-quotation marks into fields as curly quotes.
- Use Asian language line-breaking — This setting controls how line endings are handled when a word reaches the edge of a text field when using Asian languages.
- Use Roman language line-breaking — This setting controls how line endings are handled when a word reaches the edge of a text field when using Roman languages.

Data Entry Handling

When a new FileMaker database file is created, it is encoded with regional settings from the operating system of the computer used to create the file. These include date and time formats, currency symbols, and more. On macOS, these can be found in the System Preferences > Language & Region > Advanced panel. Once these settings are burned into a database file, they can't be modified. However, the group of radio buttons on the Data Entry group of the Text tab of the File Options dialog, allow control over which formatting is used for data entry:

- Always use current system settings — With this option selected, the file will automatically use the current system settings of the user's computer.
- Always use file's saved settings — With this option selected, the file will automatically use the database's settings.
- Ask whenever settings are different — With this option selected, if the file's settings differ from the user's computer, the database will ask the user which settings to use.

Script Triggers Settings

The *Script Triggers* tab, shown in Figure 6-18, allows database events to trigger custom scripts (Chapter 37).

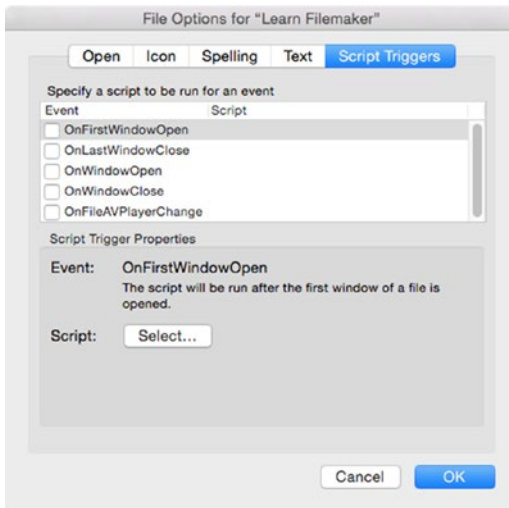


Figure 6-18. The *Script Triggers* tab of the *File Options* dialog

Maintaining Healthy Files

As you work in a database, committing content and structural changes, FileMaker continuously saves those changes to disk so there is no need or ability to explicitly perform a save function. Users can modify records and developers can design layouts, scripts, and other resources without much concern. However, things *can* occasionally go wrong, and it's important to know how to avoid, identify and remedy problems when they do occur and how to maintain healthy files.

Database Maintenance Functions

FileMaker has a several functions that make copies, help to maintain the integrity of database files and troubleshoot problems. These functions are:

- Save a copy as
- Check consistency
- Recover a file

Each of these functions must be performed on a local database file in a folder directory. Files hosted on a network with FileMaker Server must be taken offline and manipulated directly in a copy of the FileMaker application.

Saving a Copy As

FileMaker allows you to save a copy of an open database file for various purposes including:

- Making backup copies
- Creating a portable, self-contained copy
- Maintaining the health of a file
- Troubleshooting damage
- Repairing damage

To use the function, follow these steps:

1. Open the database you want to copy.
2. Select the File > Save a Copy As menu item. This will open the Create a copy named dialog, shown in Figure 6-19.

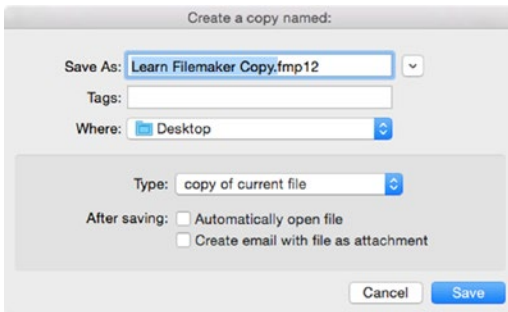


Figure 6-19. The Create a copy named dialog

3. The default entry in the Save As text area of the dialog will be the name of the original with the word “Copy” added to the end. However, you can enter a new name if desired.
4. Select the folder location into which the new file should be saved.
5. In the shaded options area at the bottom of the dialog, the Type pop-up menu, offers a choice of the following four options for the type of copy to be saved:
 - Copy of current file — This option will save a literal copy of the database.
 - Compacted copy (smaller) — This option will save an optimized copy of the database with empty space removed, resulting in a smaller file size. This can be performed periodically to maintain the health of a file, especially if a lot of resources or records are deleted from a file.
 - Clone (no records) — This option will save a copy of the database structure including tables, fields, page setup options, field definitions, custom functions, layouts, scripts and other resources but without any records. This is useful for some troubleshooting tasks as it quickly isolates file structure.

- Self-contained copy (single file) — This option will save a copy of the database with any external container data embedded in the file's container fields, making it fully self-contained.
6. Once the desired options are selected, click the Save button to complete the process. A copy of the file will be saved in the folder you specified.

Checking Consistency

Every time a database is opened, FileMaker checks the file and, if it detects the need, automatically performs a consistency check. This typically happens after a file has been improperly closed.

A *consistency check* will read every block that makes up a file and verify that the internal structure of the block is valid and the block is properly linked to other blocks. It does not read the data within blocks or check the file schema or higher-level structures; these functions are only performed by the full Recover command.

A manual consistency check can be performed on a file that is suspected of damage. To do so, follow these steps:

1. Launch the FileMaker application.
2. Select the File > Recover menu.
3. Navigate to the folder containing the file suspected of damage and select it.
4. Instead of clicking the Select button, which would begin the Recover process, click the Check Consistency button, shown highlighted in Figure 6-20.

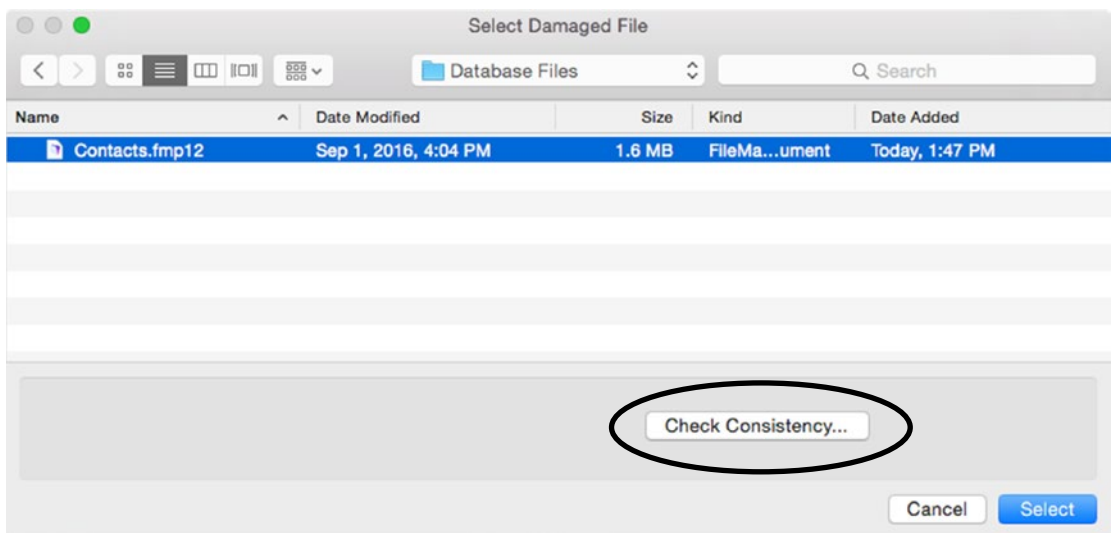


Figure 6-20. Click the Check Consistency button in the Select Damaged File dialog after selecting a database.

FileMaker will immediately check the file and report the results in a dialog and Recover.log file saved in the same folder that contains the file. Dismiss the results dialog by clicking the OK button.

If no problems have been reported, the file is probably safe to continue using. If damage is reported, it may recommend performing the full Recover process. If you suspect there may still be a problem with the file, continuing this process may be a good idea.

After the consistency check is complete, the Select Damaged File dialog remains open as a convenience. You can either choose to continue the full recovery process on the file, which is still selected, or opt to select a different file and quickly perform the consistency check on it as well. When finished checking files, click the Cancel button to dismiss this dialog.

Recovering a File

When a FileMaker database becomes damaged beyond other means of repair, the Recover function can be used to rebuild a file, regain access, and prevent the loss of data. This function is *aggressive* and will do whatever is necessary to restore access to a file. It will rebuild the file block by block and attempt to correct any corruption. However, if an object can't be repaired, it may be deleted. Because of its aggressive nature, the function should *not* be used as regular maintenance.

Before recovering a file, be sure to rule out other issues as the result of odd behavior you may be witnessing. While not an exhaustive list, a few examples of problems caused by development errors or other problems include:

- Incomplete or incorrect find results may be the result of either corruption (which would be repaired by saving the file as a compacted copy) or something like mismatched field types, such as a calculation field that is configured to result in a number or date when it returns a text value.
- Poorly designed scripts might accidentally delete records or overwrite data.
- Incorrect or broken relationship settings can result in odd behavior.
- Networking issues may cause weird behavior, which a simple quit and re-launch of the database may remedy.

Problems that may require recovery include:

- Inability to open a file.
- Odd indexing behavior.
- Records appearing blank or with question marks in every field.
- Random crashing of the application.

To perform a recovery follow these steps:

1. Launch the FileMaker Pro application.
2. Select the File > Recover menu item.
3. In the Select Damage File dialog, locate and select the damaged database, then click the Select button.
4. A Save dialog will appear, allowing you to name the new recovered file, choose a save location, and optionally specify advanced options.
5. Once finished, click the Save button.

6. A dialog message will inform you when the process is complete and summarize the activity performed. A Recover .log file, saved in the same directory as the new file, will contain additional information about the activity performed and the results.

Based on the results, you can determine if the file is safe for use or if you should transfer the records into a previously saved backup copy.

Advanced Recovery Options

The optional Advanced Recover Options dialog, shown in Figure 6-21, contains a variety of different settings that allow you to control exactly how a recovery is performed.

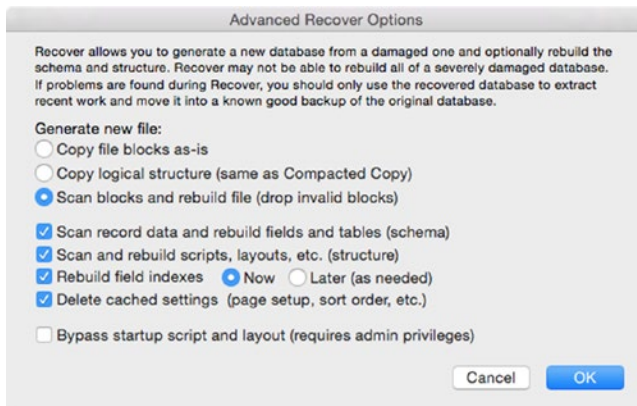


Figure 6-21. *The Advanced Recover Options dialog*

The Generate new file radio buttons provide the choice of one of three options for how the new file is generated:

- **Copy file blocks as-is** — This option will copy all file blocks exactly as they exist in the source file. The resulting file may still include damaged blocks. This is the equivalent of using the File > Save a Copy As menu item and choosing the Copy of current file option.
- **Copy logical structure (same as Compacted Copy)** — This option will copy all the data in the source file without checking the blocks but it will rebuild the tree structure. The resulting file may still include damaged blocks. This is the equivalent of using the File > Save a Copy As menu item and choosing the Compacted Copy (Smaller) option.
- **Scan blocks and rebuild file (drop invalid blocks)** — This option rebuilds the file, including only undamaged and non-duplicate blocks. The resulting file can be used but may be structurally unsound and should be used to extract data into a reliable backup copy of the solution.

The remaining check boxes can be selected as needed to achieve the desired result.

- Scan blocks and rebuild file (drop invalid blocks) — Check this box to rebuild the file's database *schema* (tables, fields, and relationships). This option will find and remove any fields or records found to be at invalid locations within the file and will re-create missing fields and table definitions, all of which may take a long time depending on the size of the file.
- Scan and rebuild scripts, layouts, etc. (structure) — Check this box to rebuild the file's *structure* (layouts, scripts, themes).
- Rebuild field index — Check this box to choose an option of whether to rebuild indexes Now or Later. Rebuilding indexes now makes the recovery process take longer but the work is done before users access the file. Rebuilding later forces indexing to occur as users begin searching or performing other functions, which can be time consuming. Uncheck the box to copy indexes without modification.
- Delete cached settings— Check this box to remove recent/last settings such as page format, print settings, layout formatting defaults, import/export order, sorts, finds, and send mail options.
- Bypass startup script and layout (requires admin privileges) — Check this box to bypass any OnFirstWindowOpen or OnWindowOpen script triggers (Chapter 37) for the file and open to a newly created blank layout.

Once finished, click the OK button to save settings and return to the Save dialog and continue with the recovery process.

Avoiding Design and Training Deficiencies

Users and developers must contend with the possibility of accidental data loss. Putting aside loss caused by file damage, there are several methods for losing data. They range from a misunderstanding to poor interface design.

There are many design techniques that can be incorporated into an interface and/or script design that can help users avoid data entry-related errors, including:

- Layout text can provide clear instructions.
- Style changes and other graphical prompts can make a more vivid indication of the current window mode and record state.
- Scripts attached to custom menus can override default vague notifications and instead include descriptive messages that make clear what action will be performed and confirm that an action has performed as expected.

However, even with the best interface and technical design, training is important. Any training sessions or manuals for users of your database, even for experienced users, should include at least a mention of risky topics and an emphasis on the proper methods of data entry and other tasks.

Some areas of risk to consider:

- Overwriting an existing record
- Forgetting to commit a record
- Deleting records carelessly
- Misidentifying a found set

These erroneous actions are not limited to a human user. A poorly designed or inadequately tested script can perform all the same mistakes as a user, and they can do so in much greater volume with the speed and efficiency unmatched by a human user.

When writing scripts Chapters 34-27 ensure that the script's actions are correct, properly limited, context sensitive, and well tested. Spend adequate time designing and testing every script, especially those that perform automatic record deletions or field modifications to ensure that they are designed with safety in mind and will only carry out functions exactly as intended.

Overwriting an Existing Record

Overwriting data on existing records is more common a problem than you might think. Be aware that there are several scenarios in which a record may be accidentally overwritten:

- *Ignorance* — People inexperienced with computers in general or databases in particular will often not realize that they need to create a new record prior to typing data and mistakenly type over an existing record with new information. This is generally the fault of inadequate training.
- *Failed Duplicate* — If a user unknowingly fails to create a duplicate of a record, they may begin typing over the original record thinking it is a duplicate. This is often an interface and script design failure that does not offer the user enough of a visual indication that they have successfully duplicated a record.
- *Find Mode* — Similarly, if a user fails to enter Find mode, they may begin typing over a live record thinking it is a temporary find request.

Forgetting to Commit a Record

For a new or edited record to be successfully saved, a user must commit or exit the record. While this happens automatically when a file is properly closed, it is still a good idea to manually close out a record after editing (Chapter 4, “Closing a Record”). Failure to do so can be an annoyance when the database is shared on a network and other users are waiting to gain access to the information; they won't see changes or be able to edit the record until the user who performed the data entry commits.

Not committing a record can even be *dangerous*. Various events can cause the changes to an uncommitted record to be *lost*, including:

- If the user's computer crashes.
- If the FileMaker application unexpectedly quits or is force quit by the user.
- If the network connection is dropped as it is when the user's computer goes to sleep.

Deleting Records Carelessly

The careless deletion of one or more records can be the result of inexperience. However, more often it is the most experienced and efficient user that commits this error. The problem here is two-fold:

- *Vague Dialogs* — FileMaker's default Delete Record and Delete All Records dialogs, which are intended to be generically applicable for any record in any table, present a rather minimalistic message that does not adequately notify the user about which record(s) in which window will be deleted.

- *User Efficiency* — When an experienced user mistakenly deletes the wrong record(s), it is often the result of their own *efficiency* within the system. While a conscientious new user will carefully read a dialog message and proceed with more caution, an experienced user, especially those who use key commands to more quickly perform various tasks, can get into a sort of “road hypnosis” autopilot mode that makes them more error prone. Combine a vague dialog with a user who thinks they know exactly what they are doing, anticipating dialogs without reading them, and you may find yourself with erroneous deletions.

The best way to address this problem is to override the default Delete Record and Delete All Records commands WITH custom menus (Chapter 33). This allows the implementation of one or more protective measures, including:

- *Restrict Access* — The security settings of a database (Chapter 40) can completely restrict access to either or both delete commands for certain users in some or all tables. This option won’t work for every situation, but is prudent when there is a class of users who never require the ability to delete records.
- *Explicit Dialogs* — With a custom script overriding the delete functions, the message presented in the dialog can be improved to include additional information such as the table name to make it clear exactly what is going to be deleted. For example, the message might read, “Really delete the 100 *Contact* records in the found set?”
- *Multiple Dialogs* — Adding a second confirmation dialog can help avoid accidental deletions, especially those involving large groups of records. This can be inconvenient for the user but helps avoid the greater inconvenience of deleting something mistakenly.
- *Require Explicit Confirmation* — When a confirmation dialog is generated using a layout rather than the Show Custom Dialog script step, you can include a field into which the user must type the word “Delete” or some other explicitly confirmation of the impending action.
- *Set Limits* — Setting limits on how many records can be deleted at once is a good can help avoid accidents. If a found set is greater than a chosen limit, an alternative dialog can inform the user that their request to delete the found set is denied.

Misidentifying a Found Set

A person unfamiliar with how to identify what is in a found set may accidentally perform functions that affect more records than they intend, including the following functions, all introduced in Chapter 4:

- *Delete All Records* — If the user thinks the found set contains ten records but it really contains ten thousand, they may inadvertently make a mistake.
- *Replace Field Contents* — Without a clear knowledge of what records comprise the found set, using this command to batch modify the field contents of every record in that set can be extremely dangerous, especially for novice users.
- *Find/Replace* — This command has an option to apply the find-and-replace functions to every record in the found set. If used with a misidentified found set or used accidentally, the results can be devastating.

Restraining File Size

Everything you add to a database will increase the size of the file. Defining tables and fields, designing layouts, building scripts, and entering hundreds, thousands or millions of records; it all slowly adds up to a larger file. This is a natural process and doesn't pose a problem, at least not at first.

FileMaker's database file size allowance is rather large, limited to the amount of disk space available with a technical maximum of 8 terabytes. Therefore, the likelihood of running out of space is rather slim in all but the most extreme cases. However, despite this incredible allowance, it is still a good idea to follow the adage "less is more" and design files to be as efficient as possible. Larger files can be slower, especially running across a network and may be prone to other problems.

There are two primary issues that can make a file larger than it needs to be:

- Inefficient design
- Fragmentation

Inefficient Design

An inefficient design can greatly increase the size of a file and may discourage the addition of time saving features a for fear of creating more clutter.

Some examples of things to watch for include:

- *Internal Container Storage* — If a database doesn't need to be portable, use external container storage instead of internal (Chapter 10). This will keep all attachments linked and accessible from within the database but store them in a folder outside the database, which greatly reduces the size of the file.
- *Redundant Technical Structures* — When planning tables, fields, relationships, and layouts, be sure to avoid anything that would cause a redundancy. Remember when tables are related, field information can be accessed and shared across a database's information hierarchy. If possible, avoid repeating anything anywhere.
- *Indexing* — Field indexing will impact file size you can't avoid indexing completely and wouldn't want to; however, be sure you learn how to use the options wisely to minimize excessive indexing (Chapter 8).
- *Graphics on Layouts* — Designing a graphically rich interface can improve the user experience and help workflow efficiency. However, it can increase the size of the file and the length of time it takes to load and draw layouts. Consider these strategies to avoid this problem:
 - Use FileMaker's built-in icons for buttons where possible.
 - Optimize graphics for screen use, especially if they are not on print layouts.
 - Keep commonly used graphics, such as logos, in a single global container field that stores externally if possible; then place the field on multiple layouts as needed.

Please don't feel limited by these suggestions. There are doubtless many other ways to be inefficient in a database and numerous creative techniques for maximizing efficiency. Further, it would be improper to consider any of these suggestions as appropriate in *every* context, and a violation of them where needed may be perfectly valid and even recommended in some situations.

Fragmentation

Even a file with an award winning and highly efficient technical design will eventually require some maintenance to avoid the bloat of empty “ghost” blocks. As records and resources are created and deleted through the course of normal data entry work and development changes, empty space will accumulate within the database file. Over time, this will artificially inflate the file size and may impact performance and stability.

To solve this problem, use the **Save a Copy as** command and choose the **Compacted Copy** option (described previously in this chapter). This will safely re-create the file, fitting as much data into each block as possible and reclaiming unused space.

Understanding File Damage

Like any digital information, a database file can become damaged. Unlike other file types, databases tend to get a lot of continuous use and are, therefore, more vulnerable. Even following all the best advice from countless experts, carefully crafting a perfect database and rolling it out with weeks of user training and detailed user manuals, at some point a power failure or some other tragedy will befall you. It is important that you understand how to avoid, detect, troubleshoot, and repair damaged files.

Generally, if you have a corrupt database, you’ll know it by one of the following symptoms:

- A database is automatically run through a consistency check when opened.
- The FileMaker application is randomly crashing.
- A dialog states that the file is damaged, can’t be opened or should be recovered.

Causing File Damage

Typically, the cause of file damage is that a database file isn’t closed properly. This can result in the loss of record data and structural flaws to the file. The damage may initially go unnoticed but can manifest as unexpected behavior or make a file unable to open.

Some of the situations that can result in database file being improperly closed are:

- Computer power is interrupted, resulting in abrupt shutdown while FileMaker hosts a database.
- A system-wide crash that also crashes FileMaker Pro.
- FileMaker unexpectedly quits due to a disk-read or file-damaged error.
- FileMaker is forced to quit by a user or some other process.

Avoiding File Damage

There are many things you can do to help keep your files free of damage.

Maintaining Hardware and Software

Always maintain stable and up-to-date hardware and software, paying close attention to these details:

- Perform extensive testing before running a live production database on upgraded hardware or software.

- Keep the operating system of host and user computers updated to the latest compatible and stable version.
- Use the latest updater for your version of FileMaker Pro to ensure the latest bug fixes.
- Keep an adequate uninterruptable power supply and surge protector on all computers, especially the database host, to avoid sudden shutdowns in the event of a power loss.
- Run regular hardware maintenance using a disk utility program and don't let computers get outdated to the point of risking failure.
- Use fast hardware and networking equipment with adequate cabling.

Avoiding Risky Behavior

Be sure to train users to avoid risky behavior such as the following:

- Don't force quit the FileMaker Pro application unless necessary.
- Don't leave databases running overnight or when your computer is idle.
- Don't allow your computer to put the hard drive or processor to sleep; use monitor dimming instead.

Using a FileMaker Server

Host files on a FileMaker Server to assist with regular backups and cushion databases from user computer crashes (Chapter 39).

Performing Maintenance

Perform regular database file maintenance, including:

- Save as compacted
- Run consistency checks
- Check for file damage

Improving Development Standards

Avoid mistakes that require a force quit or confuse users into thinking one is required.

- Avoid scripts with endless loops or modal dialogs without close buttons. Sometimes the only way to escape these is to force quit the database.
- Make sure users understand that when a complex script runs, it may take a long time. They may mistakenly assume their computer is crashing and force quit the application or perform a hard reboot.

Troubleshooting Process for Random Crashing

When a file explicitly reports damage or won't open, the course of action is clear; recover the file, and transfer the data into a known safe backup copy of the file structure. However, when corruption manifests itself in a file that seems to function properly but is plagued with random crashes, a more thorough troubleshooting and repair process is possible.

Copy the File

Start with a backup copy of the damaged database. If your files are hosted on a FileMaker Server, run a scheduled backup and then copy the backup to the desktop. If working locally on your computer, make a copy of the file onto the desktop.

Locate the Damage

First you want to determine if the damage is in the database structure or record data. Depending on what you find, your recovery option and choice of file for future use will vary slightly.

Checking the Database Structure

To determine if the *database structure* is damaged, follow these steps:

1. First, isolate the structure by making a copy without any record data. Open the backup copy of the file and perform the **Save a Copy as** command, selecting **Clone (no records)** option. Then close the file.
2. Then, compact the clone to preemptively fix any minor issues. Open the clone from step 1 and perform the **Save a Copy as** command, selecting **compacted copy (smaller)** option. Then close the file.
3. To diagnose the structure, Recover the compacted clone from step 2 using default recovery settings (uncheck **Advanced options**).

If damage is found, you can skip checking the record data and transfer record data from a recovered copy of the original backup file into a good structure.

Ideally use a clone of a backup from a time prior to the introduction of corruption. If the corruption has been lingering awhile, you may not have access to such a file and can use the recovered file generated above as the structure. However, keep in mind that the recovery process may result in renamed or deleted objects that could not be recovered so some work to remedy these may be required. For example, a field might be renamed "Recovered Field," indicating that there was some loss of data.

Checking the Record Data

Once the file structure has been ruled out as the source of corruption, follow these steps to determine if the *record data* is damaged:

1. First, compact the backup copy (the original copy with records intact) to preemptively fix any minor issues. To do this, open the backup copy and perform the **Save a Copy as** command, selecting **compacted copy (smaller)** option. Then close the file.
2. Recover the *compacted backup* from step 1 using default recovery settings (uncheck **Advanced options**).

Determine the Best Course of Action

If *no* damage is reported in both the file structure *and* record data, you can try to use the recovered file from the conclusion of the record data check. If random crashes continue to occur, consider transferring the record data into a safe structure.

If the structure *or* record data reported damage, immediately transfer the record data to safe structure

Transferring Record Data from Recovered File to Good Structure

To move record data from the recovered file into a known good structure, you can do so by following these steps:

1. Choose either the recovered file from the conclusion of the record data check or the original backup file as the source of records.
2. Open the file.
3. Perform a Find All to make sure the found set includes all records.
4. Export all records using the Excel Workbooks (.xlsx) type option. Be sure to check the Use field names as column names in first row option in the Excel Options dialog. In the Export Options dialog, take care to select all fields from the Current Table and not just the ones visible on the current layout.

■ **Caution** By exporting to a non-FileMaker format, we guarantee that the data is cleanly separated from any structural file corruption. However, container fields are not supported in these other formats and can't be exported. If the database has containers, those must be manually restored into the new database at the completion of this process when performing this process. In this case, you may attempt to export to the FileMaker format, but should double-check to confirm that no corruption accompanies the data.

5. Repeat steps 3 and 4 for every table in the database.
6. Identify a stable copy of the database *structure* (without records). Ideally use a clone that was created from a time prior to the corruption event. If that is not available, use the recovered copy generated after the database structure check.
7. Import the records from the spreadsheet into this structure, being sure to use the Arrange by Matching Names option.
8. Repeat steps 6 and 7 for every Excel export generated above.

Specifying File Paths

Throughout the development interface, there are many Specify File dialogs that request a File Path List like the one shown in Figure 6-22.

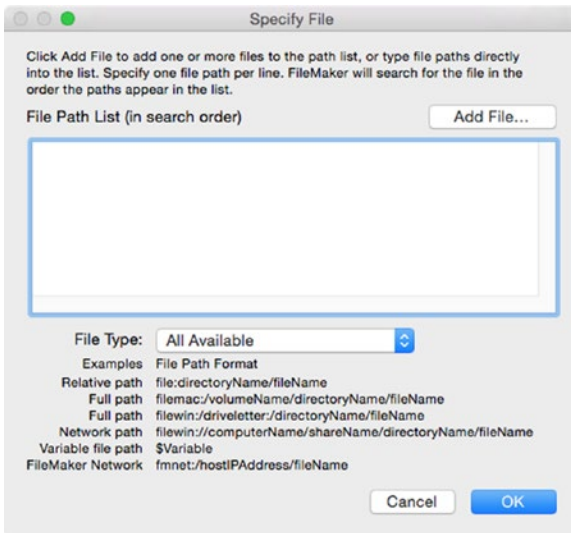


Figure 6-22. An example of a Specify File dialog

This option exists in the following developer dialogs:

- The Edit Data Source dialog when linking to external FileMaker Pro databases.
- Various Insert script steps for picture, audio/video, PDF, and file.
- The Export Field Contents script step.
- The Import Records and Export Records script steps.
- Various Save Records script steps for excel, PDF, and snapshot link.
- The Convert File script step.
- The Save a Copy As script step.
- The Recover File script step.
- The Send Mail script step when adding an attachment.
- The Send Event script step when sending an AppleEvent to a document.

Formatting Paths

As indicated by the text examples shown at the bottom of the dialog, FileMaker can handle a variety of different types of paths. The path(s) entered can be a local directory, server folder directory or a FileMaker Server network address. The File Path List text areas can contain one or more carriage return-delimited file paths. If more than one path is entered, FileMaker will check each one, in the order entered, until it finds a valid path. This comes in handy when working in a network with a mixture of macOS and Windows users of the database, as you may need to enter multiple paths to handle each user's operating environment.

Properly formatting a path or knowing when to use multiple paths can be tricky, especially when addressing files in a directory instead of addressing a file hosted on a FileMaker Server. There are various prefixes that can be somewhat mixed and matched with different path format types. If a path is incorrectly entered or the target file is moved after a path has been defined, an file missing error will occur.

To avoid errors, it is generally a good idea to use the **Add Files** button and subsequent **Open File** dialog to select a database file because FileMaker will then formulate the best path option automatically. This can be a good way to learn exactly how a path should be formatted. Once entered, you can always customize the path as needed.

Path Prefixes

There are four path prefixes that can be used depending on the type of path and the operating system:

- **file:** — used to create a generic, cross-platform path.
- **filemac:** — indicates a path to a folder for use on a macOS computer.
- **filewin:** — indicates a path to a folder for use on a Windows computer.
- **filenet:** — indicates a path to a FileMaker Server hosted database.

To create a file path for use with the **Insert PDF** script step or the **Insert Picture** script step, use a prefix of **image**, **imagemac**, or **imagewin**.

To create a file path for use with the **Insert Audio/Video** script step or the **Insert QuickTime** script step, use a prefix of **movie**, **moviemac**, or **moviewin**.

Path Types

FileMaker paths can be defined as the following different types:

- Relative path
- Full path
- Network path (Windows)
- FileMaker server network address path

Relative Path

A *relative path* is a path to a target database file that is specified from the context of the database's location. This type of path assumes the current file and the target database have a portion of their directory location in common and only specifies the difference between them. The formula for a relative path is:

```
file:[pathDifferential]fileName
```

The **file:** prefix and **fileName** are required values and the path differential is only included if there is a difference between the two folders containing them.

In the following examples, we assume two files: **Test Source** and **Test Target**, with the former defining a data source targeting the latter.

Within the Same Folder

If these two files are contained in the same directory folder (or hosted on the same FileMaker Server), the relative path will be:

```
file:Test Target
```

In this case, there is no need to include any information about the difference between their locations because they are located together.

Both Removed by One Subfolder

If we move the files, one to the Desktop folder and one to the user's Documents folder, the path will need to be modified to still locate the file. In this example, the two files are in the following locations:

```
Macintosh HD/Users/john_doe/Desktop/Test Source
Macintosh HD/Users/john_doe/Documents/Test Target
```

Given this, the data source path in Test Source that points to the target file would be:

```
file:../Documents/Test Target
```

The first part of the path differential, the two periods and forward slash, together indicate that, to locate the target, we must first move up the directory hierarchy one folder. The second part indicates the downward folder structure that is required to locate the file.

Adding Additional Subfolders

Leaving the source file in the Desktop folder, if the target was placed into a folder named "Target Subfolder" within Documents, that make the new path:

```
file:../Documents/Target Subfolder/Test Target
```

However, when the source is placed into a new folder on the Desktop, we must add an additional set of periods and forward slash to indicate the need to move up the hierarchy twice before navigating downward. In this example, the two files are in the following locations:

```
Macintosh HD/Users/john_doe/Desktop/Source Subfolder/Test Source
Macintosh HD/Users/john_doe/Documents/Target Subfolder/Test Target
```

Given this, the data source path in Test Source that points to the target file would now be:

```
file:../../Documents/Target Subfolder/Test Target
```

Now we are telling FileMaker to look for the target file two folders up from the source folder and then down into Documents and then into Target Subfolder.

Generally, you may want to avoid using relative paths unless the databases will always be within the same folder or inside of a mix of subdirectories within the same root project folder that will always remain intact and static — or if the files are hosted on the same FileMaker Server.

Full Path

A *full path* is a path to a target database file in a directory that is specified from the context of the disk or volume containing it. This establishes an absolute path that doesn't change, regardless of where the source database is located. These can be used for macOS directories locally or on a server and for Windows local directories.

The formula of an absolute path varies slightly depending on the operating system of the user computer:

```
filemac:/volume/directoryName/fileName
filewin:/driveLetter:/directoryName/fileName
```

For example, if the file is in the /User/Documents folder on a macOS computer, the path might be:

```
filemac:/Macintosh HD/Users/alex_smith/Documents/Test Target
```

Alternatively, the generic file: prefix will work for full paths that are cross-platform:

```
file:/Macintosh HD/Users/alex_smith/Documents/Test Target
```

Network Path (Windows Shared Directory Only)

A *network path* is a path to a target database file located in a server directory in a Windows environment. The formula of a network path is:

```
filewin:/computerName/shareName/directoryName/fileName
```

For example, a path might be:

```
filewin:/Company_Server/Databases/Sales/Test_Target
```

FileMaker Server Network Address Path

A *network path* is a path to a target database file hosted on a FileMaker Server computer. The formula of a network path is:

```
fmnet:/addressOrName/fileName
```

Some examples include:

```
fmnet:/192.100.50.10/Test Target
fmnet:/FileMaker-Server.local/Test Target
```

If the source database is located on the same FileMaker Server as the target, a relative path can be used:

```
file:/Test Target
```

Building Dynamic Paths

Keeping your database future-flexible is a prudent approach. One method is to keep all paths soundly formatted and able to dynamically change over time without extra development effort. There are a few techniques that can be used to construct dynamic paths and avoid future problems.

Using Variables in Paths

Paths can include global or local script variables (Chapter 12), either containing the entire path or mixed with the text of a path.

The `Set Variable` script step can place a value into a variable, which can be used in other steps that require a path. For example:

```
Set Variable [ $pathToExport ; Value: "filemac:" & Get ( DesktopPath ) & "Reports/"
Import Records [ With dialog: On ; "$pathToExport" ; Add; UTF-8 ]
```

The path text can include variables mixed in with literal text. This allows certain values to be set universally by storing them in a global variable and inserting the variable in their place in a path. For example, if a network address and file name for a commonly used target file is each stored in respective variables, the path becomes more dynamic, as shown here:

```
fmnet://$ServerAddress/$FileName
```

■ **Caution** Variables are only fully supported in script steps that use stored paths. Stored container fields may accept a variable path but it will not perform in a desirable manner.

Using Functions to Generate Contextual Paths

Some of FileMaker's built-in `Get` functions (Chapter 22) will automatically generate paths to standard folders on the user's computer, which can be used in calculation formulas (Chapter 12) and put into a variable for use in scripts (Chapter 34). These include:

- `Get (DesktopPath)` — returns the path to the user's Desktop folder.
- `Get (DocumentsPath)` — returns the path to the user's Documents folder.
- `Get (PreferencesPath)` — returns the path to the user's Preferences folder.
- `Get (TemporaryPath)` — returns a path to a temporary folder that will be automatically deleted when the FileMaker application is quit.
- `Get (SystemDrive)` — This function returns the name of the Startup Disk, which can easily be converted, into a path.

Using these when generating a path eliminates the problem of differences from one user's computer to the next or having to clean up files exported temporarily for emailing or other functions.

Excluding File Extensions

File extensions in paths of a target database are completely optional. Since FileMaker tends to update the database file extension every three or four versions whenever a new file format is introduced, it is a good idea to either drop extensions or use a variable so they can be universally updated in the future.

For example, any of these paths should locate and open the same file:

```
filemac:/Macintosh HD/Users/alex_smith/Documents/Test Target
filemac:/Macintosh HD/Users/alex_smith/Documents/Test Target.fmp12
filemac:/Macintosh HD/Users/alex_smith/Documents/Test Target.$$Extension
```

Avoiding Obsolete Wildcards

Previous versions of FileMaker allowed an asterisk to be used as a wildcard character in network file paths in place of the address. For example:

```
fmnet:/*/Test Target
```

This prompted FileMaker to search the network for the file on any eligible FileMaker Server. However, due to the impact these have on network traffic it is recommended that you not use these and, if converting an older file, search and replace these with a proper path.

Summary

In this chapter, we discussed the basics of creating, configuring, and maintaining database files as well as the how to refer to files by specifying paths.

CHAPTER 7



Working with Tables

A *table* is a collection of information pertaining to some specific type of entity, made up of a collection of *records*. They are the central focal point of the database structure, or schema. Tables define the field structure of the data that will be stored within them (Chapter 8). Instances of tables can be interconnected to form relationships (Chapter 9). A table's fields can be displayed on layouts (Part 4) and manipulated with scripts (Part 5).

In this chapter, we will begin defining and building tables. Topics will include:

- Modeling tables
- Introducing the manage database window (Tables)
- Naming tables
- Managing tables

Modeling Tables

Data modeling is a term used to describe the process of planning and creating an abstract, virtual representation of the various properties, relationships and actions that describe a set of related entity classes that will be managed within a database. This term is used because the resulting table, field, and relational structures of a database are sometimes called a *virtual model* of the types of entities that are represented by the data structure.

The first step in planning a database model is to determine the names of the tables required. A table's object category might be a broad category (people or products) or a narrower subcategory (employees or cars), depending on the specific needs of the database you are building. In some cases, a table may even model some aspect or property of an entity or action, such as a quality of an object or process of an event.

In the early planning stages, a data model can simply be a list of table names. Later, you will want to sketch out a list of the fields that will be defined within each and indicate how the tables will interconnect to form a relational hierarchy.

For now, our hypothetical data model can simply list the entity classes that will each be represented by a table. For example:

- Company
- Contact
- Project

As we expand into other areas of the data structure in later chapters, we will expand this model.

Introducing the Manage Database Dialog (Tables)

Tables are defined from the Tables tab of the Manage Database dialog. Once a database is open, this dialog can be opened by any of the following methods:

- In browse or layout mode, select the File > Manage > Database menu item.
- In layout mode, click the Manage icon in the toolbar and select Database from the pop-up menu.
- Running a script or clicking a button that uses the Open Manage Database script step.
- In the Import Field Mapping dialog (Chapter 5), click the Manage Database button.
- In most table selection pop-up menus throughout the programming interface, select the Manage Database option at the bottom.

Once open, make sure the Tables tab is selected, as shown in Figure 7-1.

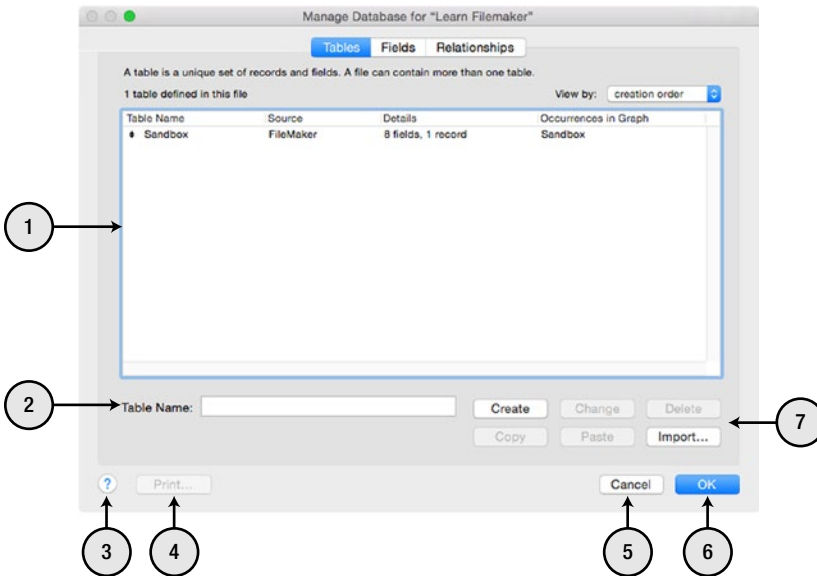


Figure 7-1. The Tables tab of the Manage Database dialog

The controls on the Tables tab are:

1. **Table List** — This area displays a list of all the tables that are defined within the current database file. The list can be sorted by either clicking on a column heading or choosing an option from the **View by** pop-up menu in the upper right corner. By default, a newly created file will automatically contain a single table with the same name as the file. However, in this case, it is different because we renamed that default table in the last chapter.
2. **Table Name** — This text area contains the name of the currently selected table. It is used to edit the table name or to enter a new table name.
3. **Help** — This button opens the FileMaker help guide to the “Defining Database Tables” page.
4. **Print** — This button will print the field definitions for the selected table(s).
5. **Cancel** — This button will close the dialog without saving changes in any tab after displaying a warning.
6. **OK** — This button will save changes and close the dialog.
7. **Buttons** — This group of six buttons is used to add tables and control table management functions described later in this chapter.

Table Properties

Tables are made up of five components, each listed as a column in the **Table List** on the Tables tab of the **Manage Databases** dialog.

- **Table Name** — This column contains the name of each table in the list.
- **Source** — This column contains the source type of each table, either “FileMaker” or the name of an ODBC data source.
- **Fields** — The **Details** column includes a count of the fields that are defined for each table (Chapter 8).
- **Records** — The **Details** column includes a count of the records that are stored in each table.
- **Occurrences in Graph** — This column contains a comma-separated list of every occurrence of the table in the relationship graph (Chapter 9).

Naming Tables

Table names can be flexible depending on a developer’s preferences. Depending on the database’s design, users may never even be aware of a table name so naming isn’t typically important from an interface perspective. However, from the perspective of an often-overlooked *technical design*, naming can greatly impact your performance as a developer when working in the programming interface.

Table names should at least follow these rules:

- They should be a unique word or phrase.
- They should not contain the name of functions, especially those that have no parameters such as `Random` or `Pi`.
- Although many reserved symbols and words *can* be used in a name, avoiding them altogether is a good idea. Some will cause conflicts in calculation formulas and FileMaker will warn you about these and confirm that you want to continue.

Including spaces in table names is optional. For some web integrations, other than FileMaker Web Direct, you may want to avoid them completely. Generally, any of these name formats are perfectly fine for tables:

Project Resources
 Project_Resources
 ProjectResources

Beyond these, consider what constitutes a *good* table name. Ideally, the name of any object in a database should be carefully planned to be clearly descriptive, concise but not cryptic, and integrated with similar objects into groups but still differentiated from each another. All this while considering how the name affects sorting with other objects in a list.

Here are a few guidelines to consider:

- A table name should clearly indicate the class of entity being modeled by the table while being concise without being cryptic.
- In almost every case, using full words is preferred over abbreviations. If a word seems too long, consider using a thesaurus to find a replacement.
- Remember, the choice of naming should be considered in the full context of other tables in the file. For example, `Stuff` may be unclear but acceptable in a database that manages only one kind of stuff. However, as a database expands to include multiple tables containing data for several different kinds of stuff, it becomes prudent to use more descriptive names like `Inventory`, `Resources`, `Supplies`, etc.
- Consider using several words to increase clarity where applicable. When there are other tables that store data for similarly named entities, make sure that their names *differentiate* each table's uniqueness. This will help avoid developer confusion.
- Including prefixes to group tables will help to *integrate* them into subgroups by causing them to sort together alphabetically. This is especially helpful when selecting a table in pop-up menus throughout the developer interface. A prefix can be one or more words but shouldn't expand the name to the point where it causes the opposite of cryptic vagueness, cluttered overload. Sometimes a grouping prefix may require a reversal of common language order. For example, instead of `Company Property` and `Personal Property`, reversing these as `Property Company` and `Property Personal` allows the common term to become a sorting prefix as well as entity class identifier.
- Be consistent in usage. For example, consider keeping all names either singular or plural; `Contact` and `Company` or `Contacts` and `Companies` are both fine, while `Contacts` and `Company` is inconsistent.

Using the above advice, consider a few examples of good table naming below:

Company
Company Offices

Contact People
Contact Company
Contact Log

Project
Project Budget
Project Schedule

Invoice
Invoice Line Item
Invoice Purchase Order

■ **Note** Remember, your preferences and needs for naming may be validly different than the isolated suggestions above. Never assume there is an absolute, context-less rule for naming that must be followed all the time.

Managing Tables

Tables can be selected, added, renamed, and deleted using the buttons on the Tables tab of the Manage Database dialog.

Selecting Tables

Many table management functions require a selection of at least one table. Selections can be made in the following ways:

- To select a *single table*, click on it in the list with the cursor. To change the selected table, click on a different table or use the Up and Down arrow keys to select the previous or next table.
- To select *multiple consecutive tables*, hold down the Shift key and click on two tables at the outside of the range desired.
- To select *multiple non-consecutive tables*, hold down the Command key and click on each desired table, one at a time.

Adding Tables

There are *six* different methods for adding tables to a database file:

- Creating a new table.
- Adding a table from an ODBC data source.
- Duplicating an existing table.
- Importing a table.
- Automatically converting a data file to a database.
- Importing records (Chapter 5, “Selecting a Target Table”).

Creating a New Table

Manually creating a new table from the Tables tab of the Manage Database dialog is a quick and easy two-step process:

1. Type the name of the new table into the Table Name text area.
2. Click the Create button.

The new table should appear in the list above with any other tables already defined.

Adding a Table from an ODBC Data Source

Open Database Connectivity (ODBC) is a standard application programming interface (API) that provides client applications a common language for interacting with other database systems written in the C language. *Java Database Connectivity* (JDBC), which FileMaker supports, is a similar API for accessing systems written in the Java language.

An ODBC or JDBC client application communicates with a *Driver Manager* application that uses a client driver to communicate with the external data source, as shown in Figure 7-2.

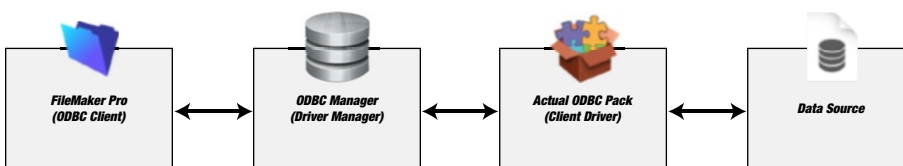


Figure 7-2. The connection between a FileMaker and an external data source through an ODBC connection

FileMaker can act as an ODBC client application or as an ODBC and JDBC data source. As a client application, FileMaker supports connections to external SQL data sources such as those from Oracle, Microsoft SQL, and MySQL. Once connected, tables from the external database can be added to a FileMaker database and, except for a few exceptions, can be used like a native FileMaker table.

The process of configuring a host computer and database for ODBC access is a relatively straightforward three-step process:

1. Prepare the host computer for an ODBC connection with a *Driver Manager* and *Client Driver*.
2. Connect the FileMaker database to the ODBC client driver.
3. Insert and use ODBC tables in the FileMaker database.

Preparing a Computer for an ODBC Connection

The first step to connecting a FileMaker database to an external ODBC source is to prepare the host computer for an ODBC Connection. This involves installing the ODBC Manager application (freeware for macOS), installing a client driver (such as Actual Technologies ODBC Pack), and then configuring the driver for a specific external data source.

The *host computer* is the computer upon which the database is installed (rather than a client computer that may be accessing it). This might be your computer with the database open in a client version of the FileMaker client application, used locally or shared across the network, or a dedicated FileMaker Server.

Installing the ODBC Manager Application

Download and install the freeware *ODBC Manager* application onto the host computer, by following these steps:

1. Download the *ODBC Manager* disk image from <http://www.odbcmanager.net>.
2. Locate the `ODBC_Manager_Installer.dmg` file in your Downloads folder and launch it.
3. The *ODBC Manager* disk image will appear on your desktop and open in a window.
4. Double-click on the `ODBC Manager.pkg` file to launch the installer.
5. Step through the installer panels to complete the installation.

Once the installer has run, the ODBC Manager application should be in your `/Applications/Utilities/` folder.

Installing the ODBC Driver

Next, download and install an ODBC driver, like the one described here from Actual Technologies, by following these steps:

1. Download the *Actual ODBC Pack* disk image, available at <http://www.actualtech.com/download.php>
2. Locate the `Actual_ODBC_Pack.dmg` file in your Downloads folder and launch it.
3. The *Actual ODBC Pack* disk image will appear on your desktop and open in a window.
4. Double-click on the `Actual ODBC Pack.pkg` file to launch the installer.
5. Step through the installer panels to complete the installation.

■ **Note** The driver is a fully functional demo, limited to display only the first three rows resulting from any query. Purchase a license key from Actual Technologies web store to remove this limit. They have two editions available, one for use on a single user database and another for shared databases.

Configuring Driver

Next, add and configure a driver for the specific database to which a connection will be made by following these steps:

1. Open the ODBC Manager application, shown in Figure 7-3.

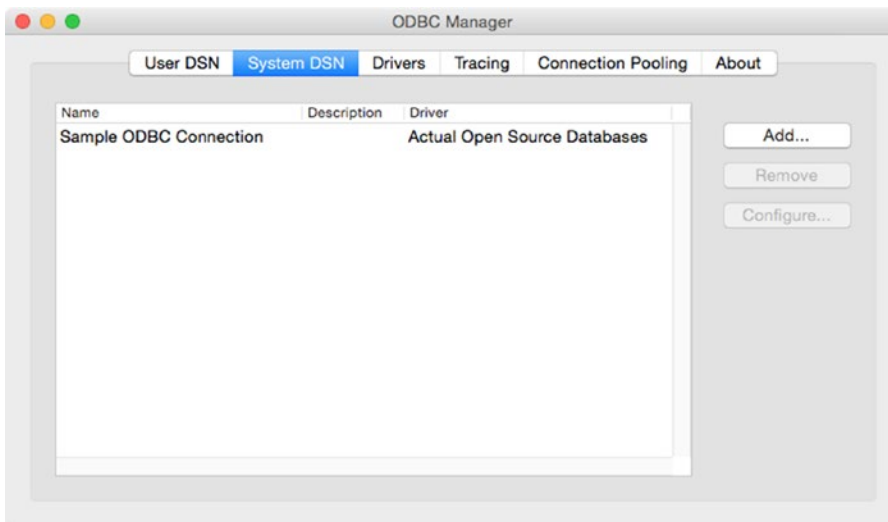


Figure 7-3. The ODBC Manager application window with a single ODBC connection configured to a sample database

2. Click on the System DSN tab.
3. Click the Add button.
4. A panel will appear attached to the dialog. Choose the appropriate driver from the list for the specific database you are connecting to. The list will include the following four driver options from Actual Technologies:
 - Actual access
 - Actual open source databases
 - Actual oracle
 - Actual SQL server

5. Enter settings into the multi-panel driver configuration dialog. The specific settings required will vary slightly depending on your choice of driver. However, generally, you must enter the server address, database name, username, and password.
6. When finished, click the Done button to close the configuration dialog.
7. The final panel of the configuration dialog allows you to test the connection to ensure that you have entered everything properly.
8. Quit the ODBC Manager application.

Connecting a FileMaker Database to the ODBC Client Driver

Next you must connect your FileMaker database to the ODBC driver you just configured in the ODBC Manager application. This is done by setting up an external data source in the database and then adding the table(s) from the external source into the FileMaker relationship graph.

Setting Up External Data Source

Select the File > Manage > External Data Sources menu item to open the Manage External Data Source dialog, shown in Figure 7-4, then click the New button.

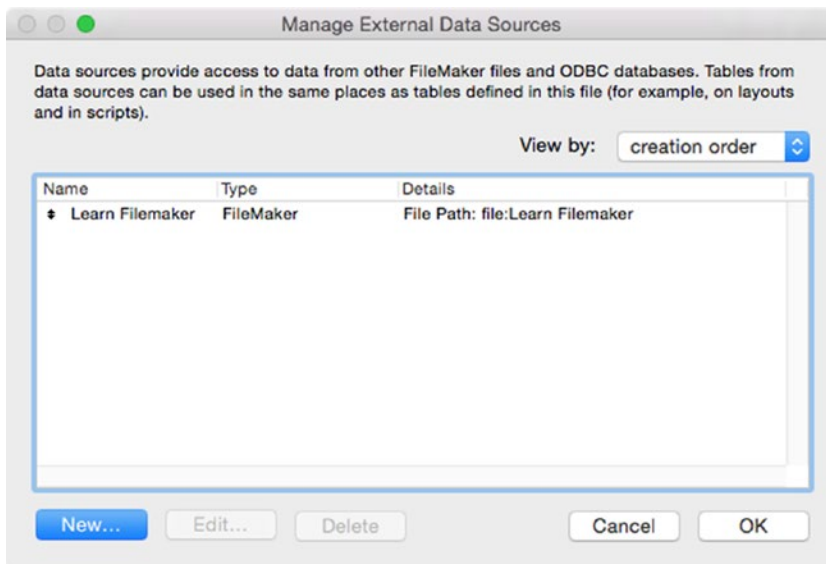


Figure 7-4. The Manage External Data Sources dialog

An empty Edit Data Source dialog will appear, like the one shown in Figure 7-5. Following the steps below to configure the connection to the Client Driver for the external data source.

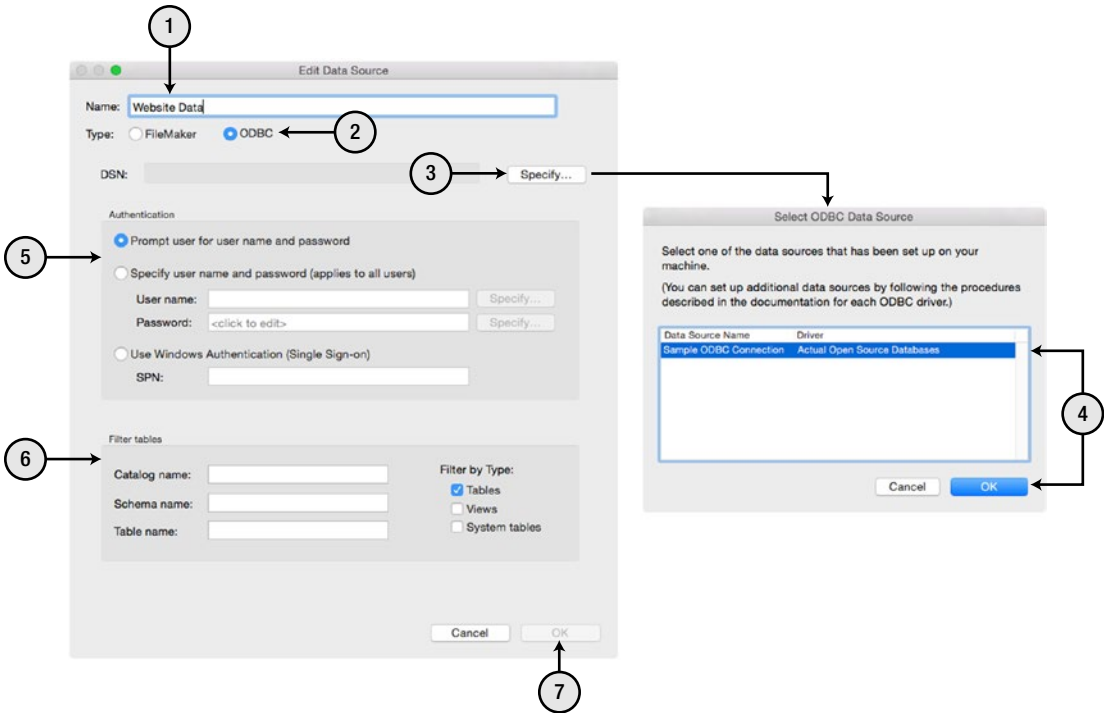


Figure 7-5. The process for setting up an external data source

1. Enter a name for the data source in the Name text area, which will be used to refer to the data source within the FileMaker database. It can be the same as the Data Source Name or a more descriptive name (as shown).
2. Select the ODBC radio button for the Type.
3. Click the Specify button on the right of the DSN text area.
4. In the Select ODBC Data Source dialog that opens, select from the available ODBC Data Sources. The list will include any you have previously defined in the ODBC Manager application. Once you select the desired driver, click the OK button to close the dialog.
5. Choose the Authentication option to indicate how a username and password should be entered at the user level when they attempt to access resources in the external database from within FileMaker. For the smoothest user experience, hard-code the user name and password directly in this area. For a more secure experience, have them prompted to enter these values at runtime.
6. Optionally, you can add filtering criteria to control which tables are displayed when a connection is made. This can be helpful when a database has many tables and you want to limit which are available from FileMaker.
7. Click the OK button to close the Edit Data Source dialog and save the current configuration.

The new external data source should appear in the list on the Manage External Data Sources dialog, shown in Figure 7-6. Click the OK button to close the dialog and save the configurations.

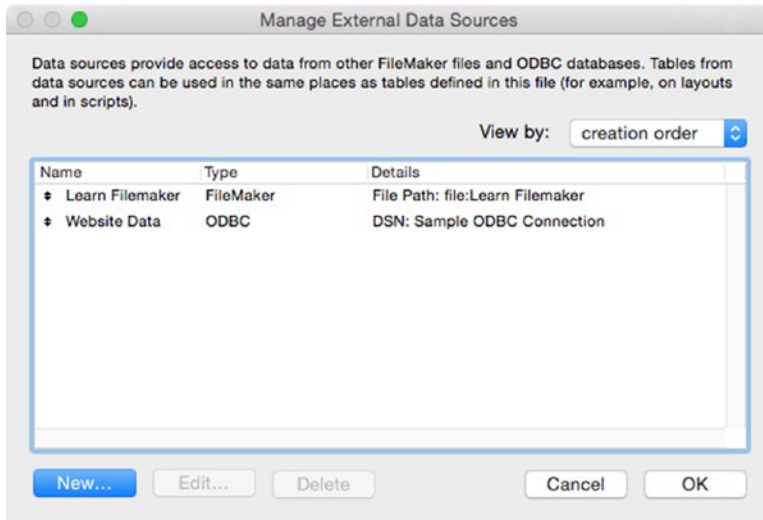


Figure 7-6. The dialog after setting up the connection to the external data source

Adding to Relationship Graph

Any table created directly within the Tables tab of the Manage Databases dialog will automatically be added to the Relationship graph, while an ODBC table must be manually added. To get started, open the Manage Database dialog and click on that tab. Then follow the steps shown in Figure 7-7.

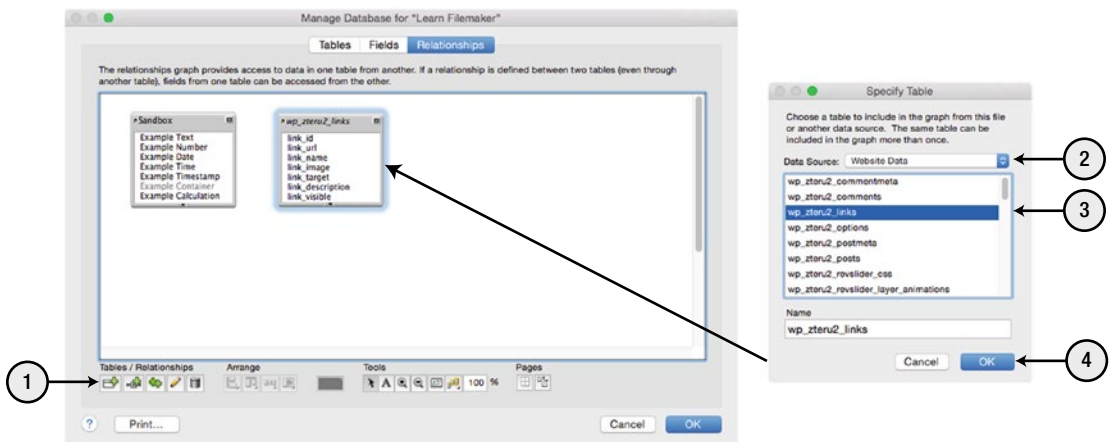


Figure 7-7. The process for adding an external data source to the relationship graph

1. Click the Add a table icon button, which is located on the far left of the row of icons.
2. In the Specify Table dialog that opens, click the Data Source pop-up and select the ODBC data source containing the table you want to add. This should refresh the list of available tables.
3. Click the table you want to add.
4. Click the OK button.

A new table occurrence box should appear in the relationship graph. If you switch to the Tables tab you will see the table in the list with italic formatting, which indicates an ODBC “shadow table.”

Using an ODBC Table in FileMaker

Once an ODBC table occurrence is added to the relationship graph, it is available throughout the database in almost the same way as any native FileMaker occurrence. However, it is important to remember that the table really exists on the external server and is represented within FileMaker as a “shadow table” so there are a few notable differences. These include:

- *Schema Lock* — The structure of the external data source is not available for modification from within FileMaker. The shadow table created within FileMaker allows some modifications but they do not affect the remote table.
- *Deleting Fields* — Fields can be deleted from the shadow table to help thin out the amount of data queried but the fields remain intact in the remote table.
- *Modifying Fields* — Auto-Enter settings for remote fields can be modified in the shadow table since they essentially operate in the same way as users entering information into a field.
- *Adding Supplemental Fields* — You can add additional fields to the shadow table but are limited to unstored calculations and summary fields and the new fields are FileMaker-only and will not be added to the remote table.
- *Data Types* — SQL has separate data types for certain information, such as integers and floating-point data, which FileMaker handles as a single data type. In some cases, you may need to create a calculation field to convert data from the external table into a FileMaker data type using one of the GetAs functions (Chapter 13).
- *Data Entry Limitations* — When there are differences in the amount of data some field types can hold in the remote table, FileMaker will do its best to validate and enforce these limits to avoid any data loss or confusion. For example, MySQL fields tend to be more limited in content size than are native FileMaker fields.
- *Data Updates* — Updates to refresh record changes in the external table are less frequent across the network than changes to native FileMaker tables. Use the Refresh script step to force a window to update and ensure that the data the user sees is current.
- *Record Locking* — Unlike with FileMaker native tables, when a user begins editing a record in an external table, other users are not blocked from making changes to the same record. So, two users can be editing the same record simultaneously. The user who commits the record second, thereby submitting their changes after the other user, will see a warning dialog informing them that the record had been modified since they began editing and give them the ability to stop to avoid overwriting the other user’s changes.
- *Indexing* — FileMaker can’t index SQL fields so searches in external tables should be limited to those fields already indexed by the remote table to avoid long delays.

Getting More Information

FileMaker has a technical brief document available on its website entitled “Introduction to External SQL Sources” that provides more information about ODBC, JDBC, and the various connection options to and from FileMaker databases.

Duplicating an Existing Table

When an existing table is similar enough to the requirements of a new table, consider duplicating it rather than starting from scratch. While there is no Duplicate button, the Copy and Paste buttons serve this purpose. To duplicate a table, follow these steps:

1. Select the table(s) that will be duplicated.
2. Copy the table(s) by clicking the Copy button.
3. Paste a duplicate of the copied table(s) by clicking the Paste button.

The duplicate table(s) will appear in the list, highlighted as shown in Figure 7-8. Each new table will have the same name as the original with a numeric suffix added to ensure it is unique. Every field from the original table will be present as an exact duplicate in the new table, with the same names and settings. Furthermore, a similarly named occurrence will be placed on the relationship graph (Chapter 9). The only difference is that the new table will not copy records from the original.

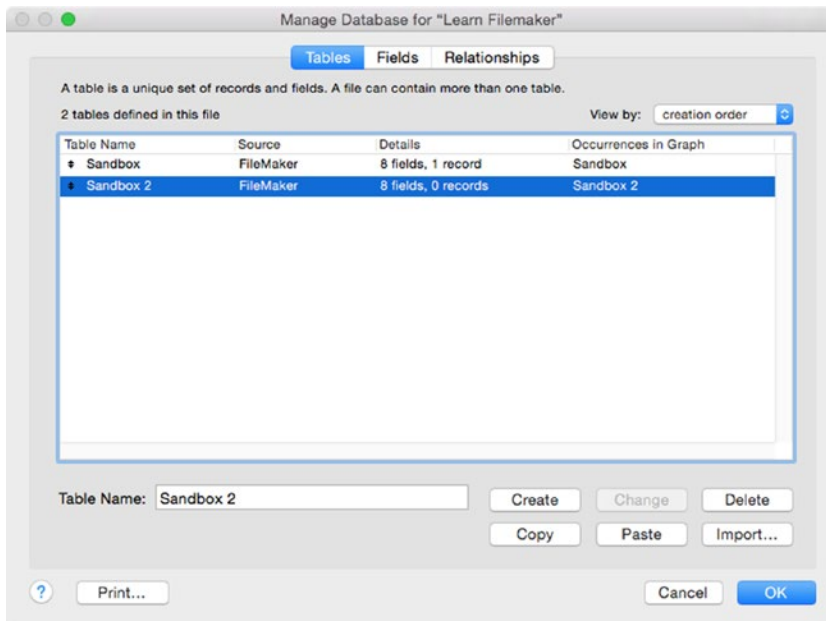


Figure 7-8. A duplicate of the Sandbox table after pasting

■ **Note** Tables can be copied and pasted *within* a file and *between* two files.

Importing a Table

The Import button on the Tables tab of the Manage Database dialog allows one or more tables to be imported from a different file directly. To import, follow these steps:

1. Click the Import button. An Open File dialog will appear.
2. Locate and open the database file containing the desired tables. This can be a file in a folder directory or hosted by a FileMaker Server. It can be the same file you are editing or a different file. Once opened, FileMaker will present an Import Tables dialog, shown in Figure 7-9, listing all the tables available for import from the selected database.

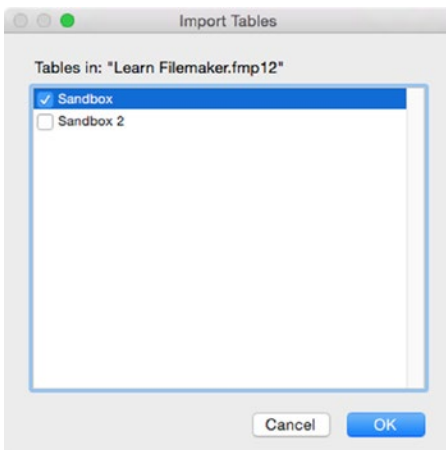


Figure 7-9. The Import Tables dialog

3. Check the check box next to the table(s) to import and click the OK button.

The selected table(s) will be copied into the current database and an Import Summary dialog will report the results, including:

- The number of items imported
- The number of items that were automatically renamed due to conflicts
- The number of errors that occurred

An Import.log file will be saved into the folder contains the destination file or onto the Desktop. If the destination file is remotely hosted.

Converting a Data File to Database

FileMaker can instantly convert a text or spreadsheet file into a new database when these are dropped on the application. The columns and rows of data from the source file will be automatically converted into a table, fields and records in the new file created by this process.

When the file is dropped onto the application, a dialog will offer two choices for field name handling, depending on what information the first row of incoming data:

- **Field names** — This option indicates that the first row of data in the file being imported contains field names rather than actual data. If selected, the first row of data will be used to provide the names of fields into which the remaining information will flow.
- **Data** — This option indicates that the first row of data in the file being imported contains actual data and not field names. If selected, field names will be given a default value (f1, f2, f3, etc.) and the first row will be imported as a record.

Once this choice is made and the OK button clicked, the process will build a new file with the following resources:

- The new database's name will be the same as the data file with a suffix of "Converted" and the FileMaker file extension.
- A default table will be created with the same name as the new database file.
- A field will be created for each column of data with naming based on the choice made in a dialog.
- A record will be created for each row of incoming data, with each column of information entered in a corresponding field.
- Two layouts will be created: a form layout named "Layout #1" and a list layout named "Layout #2."

Modifying a Table Name

Changing the name of an existing table is a simple three-step process.

1. Select the table to be renamed.
2. In the Table Name text area, type the new name.
3. Click the Change button.

Once complete the name of the table in the list should be updated. Click the OK button to commit this change.

Deleting Tables

Depending on which of the previous examples you performed, the Learn FileMaker database may now have a bunch of duplicate tables. Fortunately, deleting tables is a simple process:

1. Select the table(s) to be deleted.
2. Click the Delete button.

3. A Delete Tables warning dialog will appear to confirm you want to continue, as shown in Figure 7-10.

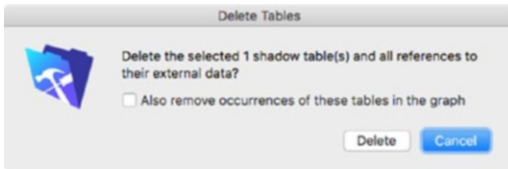


Figure 7-10. The warning dialog when deleting tables

4. The dialog includes a check box that, when selected, will delete both the tables *and* their corresponding occurrences in the relationship graph. This choice is optional and can vary depending on the reason for deleting the tables. For now, you should check the box. If you were deleting a table whose relationships will be repurposed with a new or existing table, you would want to uncheck the box to retain those connections. The table occurrences in the graph will become <missing table> indicating that they require reassignment.
5. Click the Delete button in the dialog to complete the process.
6. Manually delete and repurpose any layouts assigned to the deleted table(s).

Once complete the selected table(s) will disappear from the list. Click the OK button to commit this change.

■ **Caution** When deleting a table, all records that were contained within it will also be permanently deleted.

Summary

In this chapter, we discussed the basics of working with tables, including accessing external SQL data sources from within FileMaker.

CHAPTER 8



Defining Fields

Before a table can store records of information, fields must be defined. A *field* is a named container for one unit of information. Fields allocate a data entry space, define what data can be placed into that space, and control many aspects of how that data will be managed.

■ **Tip** Using a spreadsheet metaphor, a field is like a column; a record is like a row; and an instance of a field for a record is like a cell: the intersection of a column at a row.

In this chapter, we will focus on defining fields. Topics will include:

- Defining field data types
- Exploring field content types
- Introducing the manage database dialog (fields)
- Naming fields
- Defining standard fields
- Managing fields
- Setting field options
- Expanding the learn FileMaker object model

■ **Note** Technically, a single field can contain more than one unit of information when configured with *repetitions*. For example, a person might have multiple contact methods — phone, email, social media, messaging, etc. — and these can be placed in multiple separate fields, a single field configured with multiple repetitions or in a single field on multiple related records using relationships (Chapter 9) and displayed in portals (Chapter 30).

Defining Field Data Types

There are eight different types of fields supported in FileMaker that fall into one of two fundamental categories: *entry* and *display*.

Entry Fields

An *entry field* is a field capable of accepting data entered by a user or a script. FileMaker has six types of entry fields, each dedicated to a specific data type:

- Text
- Number
- Date
- Time
- Timestamp
- Container

■ **Note** The ability to enter data into entry-type fields is a *potentiality* based on their nature. The ability to *actually* enter information into these will depend on various factors such as the field's presence and configuration on a layout (Part 4), the configuration of a script (Part 5), the user's security access level (Chapter 40), or some combination of these.

Text Fields

A *text field* is used to store any combination of letters, symbols, or numbers as a string of any length up to ten million characters per repetition. The values can contain *any* text, including other data types entered as text. This includes any mixture of text, dates, times, and numbers. Numbers in text fields can contain leading zeros.

The structure of the text can vary depending on the field's settings. One field may be configured to impose various rules on an entry, requiring it to be an unformatted, single paragraph, limited to a few words, etc., and automatically force compliance with these rules. Another field may be configured to allow multiple paragraphs of robustly formatted text. Depending on the field, formatting can either be applied directly to the text through the interface by a user, applied dynamically to the text based on the field settings, or some combination of these two. Text formatting can also be superimposed on the text when a field is displayed on a layout.

When sorting records by a text field, the values in the field will be sorted in alphabetical order. This means that numbers contained within the text will be sorted *as text*. In other words, numbers will be evaluated one character at a time so the number 10 will sort before 2, since 1 comes before 2.

Number Fields

A *number field* is used to store any numeric value ranging from 10^{-400} to 10^{400} up to one billion characters per field.

A value in a number field can include non-numeric characters that are used to format numbers. For example, a number field can contain 5000 or \$5,000.00. While these additional characters don't adversely affect data entry or finding and sorting records, ideally only numeric characters should be physically entered into the field and the currency formatting should be applied dynamically based on the field display settings on a layout (Chapter 29, "Data Formatting") or through a separate calculation field that converts the numeric value to formatted text for display.

When sorting records by a number field, the values in the field will be sorted in numeric order.

Date Fields

A *date field* is used to store a formal date within a range from January 1, 0001, through December 31, 4000.

Dates must be typed into a date field using the *short date format* that matches the database's settings for dates and times, which are inherited from the computer upon which the file was created. Files can be configured to allow entry and display of dates based on the user's local system (Chapter 6, "Data Entry Handling") or with the Use System Formats script step. The format of a date entry will vary from one country to another so you may need to take that into consideration if your database will be used internationally. In the United States, the default format is typically the following:

<month>/<day>/<year>

For example, January 15, 2017, would be entered into a date field as follows:

1/15/2017

If the date entry is not properly formatted or contains extraneous text, FileMaker will present an error dialog. Although all dates must be entered as short dates, a field on a layout can be configured to display the value in one of a variety of formats (Chapter 29, "Data Formatting"). Some examples include:

01/15/17

2017.01.15

January 15, 2017

Sunday 15 Jan 2017

When sorting records by a date field, the values in the field will be sorted in chronological order.

Date Calculations

When entering a value into a date field, FileMaker stores the value as a number that expresses the number of days have passed since January 1, 0001. This is invisible to all users but allows a calculation to add or subtract a number of days to a previously entered date value. For example:

(1/15/2017 + 10) = 1/25/2017

(1/15/2017 - 10) = 1/5/2017

Two-Digit Date Conversion

FileMaker requires all dates to have four-digit years. Any entry of a date with a two-digit year into a date field will automatically be converted to a four-digit year using a formula based on the year the data entry takes place. This includes all entries, including those manually typed, imported, auto-entered, or entered by a script step.

The conversion process assumes that a date with a two-digit year is more likely to reach further into the *past* than into the *future*. Therefore, when automatically converting a date's year from two to four digits, the value will be adjusted to fall within either the *next 30 years* or the *preceding 70 years* based on the *current year*, when the data is entered. The borderline moment of this scale is illustrated by the conversion examples in Table 8-1, which assumes data entry in the year 2016.

Table 8-1. Showing two years on each side of the temporal divide where FileMaker switches four-digit date conversions from the past to the future

Date Entered	FileMaker's Conversion	Direction
1/1/48	1/1/1948	Back to Past
1/1/47	1/1/1947	Back to Past
1/1/46	1/1/2046	Forward to Future
1/1/45	1/1/2045	Forward to Future

A database managing dates that fall outside of this 100-year range or that don't fit this division pattern *must* be:

- Manually entered with four-digit years
- Entered as text
- Be entered in a field with an auto-enter formula or script trigger that converts the date to a four-digit equivalent *before* FileMaker imposes its conversion rule

There is one exception to the automatic conversion process. Back before the now-forgotten Y2K bug era, older versions of FileMaker allowed two-digit dates to exist. Therefore, when performing an upgrade conversion to a database built with one of these older copies of FileMaker, two-digit dates will *not* be converted using the same method but will instead automatically receive a century of 1900. Therefore, when upgrading an ancient database that used two-digit years for any century other than the twentieth, care should be taken to first convert all dates prior to upgrading to a more modern version of FileMaker or import the dates into a text field and convert them after import prior to reverting the field to a date field.

Time Fields

A *time field* is used to store a formal time string, which, like dates, must be entered with a time format matches the database's settings, which were inherited from the computer upon which the file was created. The format used will vary from one country to another so you may need to take that into consideration if your database will be used internationally. In the United States, the default formal is typically:

<hours>:<minutes>:<seconds> <optional_AM_PM>

For example:

11:30:59 AM

The values in a time field will sort chronologically.

When placed on a layout, a time field can be configured to display the time it contains in a variety of different formats (Chapter 29, “Data Formatting”).

Time Calculations

When entering a value into a time field, FileMaker stores the value as a number that expresses the number of seconds have passed since midnight. This is invisible to all users but allows a calculation to add or subtract a number of seconds to a previously entered time value. For example:

```
( 11:30:59 AM + 11 ) = 11:31:10
( 11:30:59 AM - 11 ) = 11:30:48
```

Timestamp Fields

A *timestamp field* is used to store a combination of a formal date and time as a single, unified string pinpointing a specific point in calendar time.

As with date and time values, a timestamp must be entered in a short format matching the database’s settings for each component, which are inherited from the computer upon which the file was created.

The two components are entered with a space between them, as shown here:

```
<date> <time>
```

A timestamp value can fall anywhere within the range from January 1, 0001 12:00 AM through December 31, 4000 11:59:59 PM.

Timestamp Calculations

When entering a value into a timestamp field, FileMaker stores the value as a number that expresses the number of seconds have passed since midnight on January 1, 0001. This is invisible to all users but allows a calculation to add or subtract a number of seconds to a previously entered timestamp value. For example:

```
( 1/15/2017 11:30:59 AM + 360000 ) = 1/19/2017 3:30:59 PM
```

Container Fields

A *container field* is used to store any kind of document within a database, that can be entered into a record as easily as text, dates, or numbers. Some document file types have an *Interactive content* layout setting that allow for interactivity like that provided by their native applications (Chapter 29). For example, a PDF file can be stored in a field that is placed and configured on a layout to allow a user to view and navigate through the pages of the document as if they were viewing it in Apple’s Preview application or with Adobe Acrobat. Other file types with interactive potential are photos, movies, and audio files.

Files can be placed into container fields by using a variety of functions including Insert, Copy & Paste or Drag & Drop. Depending on the configuration of the database and field, the files displayed in container fields can exist within the file structure or be linked from an external folder location (Chapter 10).

Display Fields

A *display field* is a field that automatically displays a value that is determined by its settings. FileMaker has two types of display fields:

- Calculation
- Summary

Calculation Fields

A *calculation field* is a field defined with a formula statement that is evaluated by FileMaker to produce a result automatically (Chapter 12). These are like a cell in a spreadsheet that has a formula applied except that it is *impossible* for a user to type into the cell, wipe out the formula, and enter data. Just as the spreadsheet displays the result of the formula in the cell for each row in the spreadsheet, a calculation field displays the result of the formula for each record in the database.

Summary Fields

A *summary field* is a field defined to automatically calculate another field's values across a set of records within the same table, taking into consideration the current user's context of found sets and sort orders. For example, a summary field can calculate and display the total value of a field for every record in a found set or for subsets of records based on key sort fields. Summaries are configured with a combination of field options (explained later in this chapter) and layout configurations (Part 4).

Exploring Field Content Types

While FileMaker defines field types based on the type of *data* they can contain, fields can store all kinds of *content* and serve a variety of different information management purposes.

The most obvious type of data that a field might store is an aspect or property of the entity class modeled by the table. For example, an entity's name, location, description, price, specifications, or any number of other values describing that type of entity. The information stored about an entity can range across an entire spectrum from properties relating to its appearance, use, status, possible actions, etc. However, describing entities is *not* the only use for fields.

Fields can be used to store information about the *record* rather than the entity the record represents. For example, in a Contacts table, instead of fields only storing information about the actual contact *person* that the record represents, some might contain information about the contact *record*. For example, some may store record's data entry status, notes, instructions, content warnings, hints, and tutorials. Fields can store data used to form more sophisticated dynamic connections with records in other tables.

Further, instead of storing information pertaining to an entity or the record representing that entity, some fields may even store data about the *table* itself. They may also store information about the *file*, *user*, *computer*, and other contextual data.

In still other cases, a field may be used to store global resources or temporary information that is used by other fields, formulas, or scripts.

Don't limit yourself in how you think about fields. They can serve a variety of different purposes to suit any number of different situations.

Introducing the Manage Database Dialog (Fields)

Fields are created from the Fields tab of the Manage Database dialog as shown in Figure 8-1.

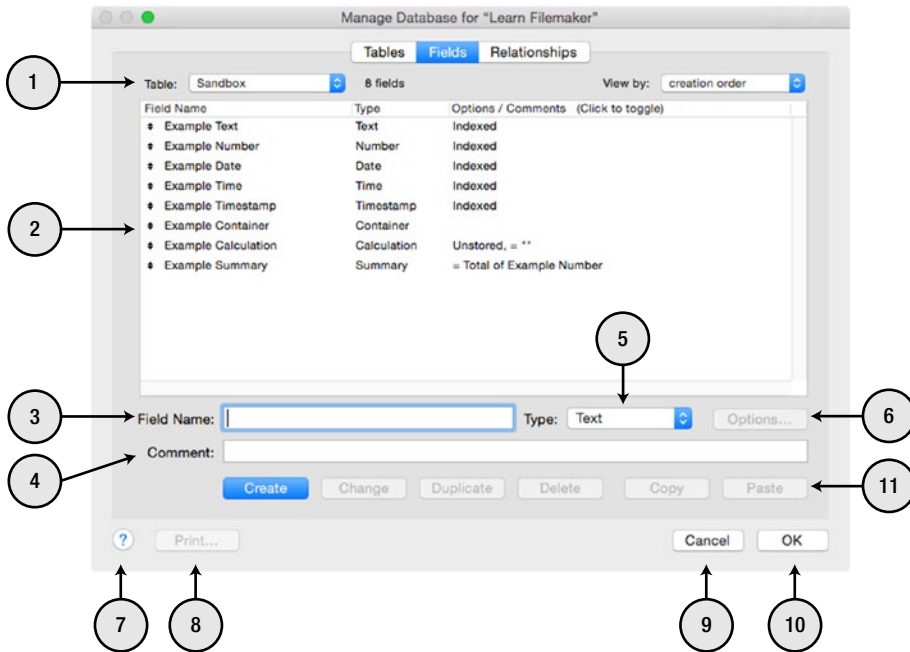


Figure 8-1. The Fields tab of the Manage Database window

The controls on the Fields tab are the following:

1. **Table Popup** — This pop-up menu contains a list of all the tables defined in the current file.
2. **Field List** — This area displays a list of all the fields that are defined for the selected table. The list can be sorted by either clicking on a column heading or choosing an option from the *View by* pop-up menu in the upper right corner.
3. **Field Name** — This text area contains the name of the selected field. It is used to edit the field name or to enter a new field.
4. **Field Comment** — This text area is used to add a short developer comment for the selected field.
5. **Field Type** — This pop-up menu displays the data type for the selected field and allows the selection of a different type when changing a field or creating a new one.
6. **Field Options** — This button will open the *Options for Field* dialog, discussed later in this chapter, which contains detailed settings for the selected field depending on its field type.

7. **Help** — This button opens the FileMaker help website directly to the “Defining Database Fields” page.
8. **Print** — This button will print the field definitions for the selected fields.
9. **Cancel** — This button will close the dialog without saving changes made in any tab after displaying a warning.
10. **OK** — This button will save changes and close the window.
11. **Field Buttons** — This group of six buttons are used to manage fields (see “Managing Fields,” this chapter).

Naming Fields

Field naming should follow the same rules and guidelines as tables (Chapter 7, “Naming Tables”). Like tables, field names can have spaces, should avoid reserved words, and should be consistently either singular or plural. They should be descriptive of the content they contain or the function they serve, without being too lengthy or cryptic. They can be named with prefixes to become visually organized when alphabetically sorted in lists and pop-up menus, even if this requires a reversal of language; for example, using `Name First` and `Name Last` instead of `First Name` and `Last Name`.

Any of these name formats are perfectly fine for fields:

```
Company Name
Company_Name
CompanyName
```

The examples in this book will use follow the pattern of the first, with a space between each word of the field name.

Highlighting the Visibility of Field Names

There are a few things to consider when naming fields. Unlike tables, a field name *may* need to be considered from the perspective of both technical *and* interface design.

When a field is placed on a layout, the *field label*, a piece of text that identifies the field for users, will default to the *actual* field name. To avoid confusing users with technical oriented naming, the label can be modified to a user-friendly variant. So, a field defined with a name of `Contact Name First` can have a visible label of `First Name`. This gives you the best of both worlds: a clear and organizing technical name and a shorter, human-readable, visible label name.

However, depending on a user’s access settings and the use of custom menus, there are some circumstances where a user can be exposed to the actual field names. If those names are too developer-centric and different from the layout labels they are used to seeing, users may become confused. They may be unable to recognize fields in certain dialogs. Some of the dialogs where this occurs are:

- Import Records
- Export Records
- Replace
- Sort
- Saved Find > Advanced

Before you consider abandoning field name prefixes to benefit development tasks, consider that, even if fields are named from a user's perspective, there will *still* be instances of field confusion in these dialogs. For example, many fields that exist for strictly technical purposes and are not displayed on any user layout will be visible in these dialogs. In this case, it may be better to include prefixes to group more technical fields away from the fields for which a user might be searching.

The benefits of naming from a technical perspective are huge and you can always educate users who may access fields through these dialogs. You can override the dialogs with your own custom menus (Chapter 33) and scripts (Part 5), although that limits users to only functionality predefined and controlled by you. You could even design your own dialogs with field name translations to ease this burden, but that can become a lot of work some will find unnecessary. In the end, you will have to find the right balance for you, your projects, and users.

Defining Standard Fields

Some developers prefer to define *standard fields* that are universal to all or most tables in any database file, regardless of the variety of information those tables contain. Often standard fields contain information about the record itself, instead of the entity represented by the record, so they can apply to almost any circumstance. This practice is completely optional but can be beneficial in establishing a consistent framework around which custom fields are constructed.

Ideas for Standard Fields

There are several examples of the kinds of fields that are obvious candidates for standardization across all tables. These include:

- A record creation and modification date, time, and name of the account or user responsible for those actions.
- The data entry status of the record. Although the content and available values for a status field will probably change greatly from one table to the next, the idea of having a field that indicates the record's current state can be invaluable. This informs users about the status, allows managers to review work done by others, and allows sorting by or searching for records based on their status.
- A freeform notes field used to store information about the record and/or the entity represented by the record.
- A unique record identification number sometimes called a *primary key* that is used for differentiating, tracking, identifying, and searching. These are useful for precise record identification and when linking records between tables using relationships (Chapter 9).
- A text calculation used to compile error notifications that can be placed on layouts to provide instant feedback to the user about data entry issues that are automatically detected but can't be automatically repaired by other means.
- A record title calculation that combines key fields into a single text string that can help the user quickly identify which record they are currently viewing. These can be displayed prominently at the top of a layout, in the body of list views, and in portal rows as a means of record identification by humans.

As you develop more complex databases and accumulate experience with more and more fields, ideas for standard fields may become plentiful. You'll need to learn to identify fields that should be universal standards from those that are applicable only to some groups of tables (group standards) or those that are applied sparingly as needed (optional standards). A universal standard field should be established only when they are truly useful to all or most all tables.

Naming Standard Fields

Once a list of standard fields is established, take some time to think about how to name them. In addition to the FileMaker imposed rules for naming fields, consider a wider set of rules to govern the naming of standard fields.

Making Standard Field Names Universal

To help make their universal status clear, don't regard their function and format as a standard alone; consider using the *same name* for a standard field in every table.

Many people use and prefer to name a field containing a unique serial number identifier with a table-identifying prefix followed by the word "ID." For example, `Contact ID`, `Company ID`, and `Project ID` would each be a field containing a respective table's primary serial number while this does establish a *standard naming pattern*, it makes locating the primary id field for any given table a little difficult alphabetically, since the field's position will change from one table to the next as the table name changes. Essentially, using this approach, an id field is mixed in with other fields. To help alleviate this, some developers prefer to make "id" a field prefix so that, when scrolling through a list, every serial number fields in the table, are all grouped together, for example, `ID Contact`, `ID Parent Company`, etc.

However, to really emphasize the universal nature of such a field, still others prefer to make the primary key in *every table* have the *exact same name*. Something like `ID Primary` or `Record ID`. This makes standard fields consistently named, quickly recognizable, and easy to locate across all tables.

Using Super Group Prefixes to Emphasize Standard Fields

For table and field names, we discussed the idea of using a prefix to push items into groups when they appear sorted alphabetically. The same idea applies when developing standard fields, but you can take it an additional step further and develop a super group prefix.

A *super group prefix* is a single word or abbreviation used at the beginning of every field name to separate all fields into two fundamental groups: standard and custom.

When naming standard fields, you might notice that some can be grouped together with logical prefixes like those discussed earlier. For example, `Creation Date`, `Creation Time`, and `Creation User` are clearly named and will sort together in a list. Similarly, `Modification Date`, `Modification Time`, and `Modification User` help group those fields together. However, as you begin developing additional standard fields and mix them with other table-specific fields, the need for an additional level of prefix will become clear.

■ **Tip** Professional developers pay as much or more attention to *technical design* and naming as they do to *interface design*.

To help illustrate this point, consider a list of standard and custom fields mixed together in alphabetical order:

```

Creation Date
Creation Time
Creation User
Contact Name First
Contact Name Last
Modification Date
Modification Time
Modification User
Notes
Phone Cell
Phone Home
Phone Work
Record ID

```

While these fields are nicely arranged in subgroups due to the use of prefixes, the intermixture of the custom fields with standard can be visually distracting and even confusing. And from one table to the next, the position of the standard fields will change in relation to different table-specific prefixes.

Now, consider the difference when you add a super group prefix. Since the fields in the example are from a table containing *contact* records, we use Contact as the super prefix for every single custom (non-standard) field. Since all the standard fields contain information pertaining to a *record*, we use Record as the prefix for them. The resulting list of fields immediately appears more orderly.

```

Contact Name First
Contact Name Last
Contact Phone Cell
Contact Phone Home
Contact Phone Work
Record Creation Date
Record Creation Time
Record Creation User
Record ID
Record Modification Date
Record Modification Time
Record Modification User
Record Notes

```

Now we have a clear separation of standard fields from custom fields. Since standard fields have the same name and prefix in every table, they become much easier to locate and select, or to ignore when focusing on a custom field.

Not to belabor the point, but as you add additional fields to both super groups, you may consider additional sub-prefixes to help further group similar fields. For example, the two sets of creation and modification fields both contain a type of status for the record, so adding an additional prefix will further refine the list by pushing these two groups together as shown here:

```
Contact Name First
Contact Name Last
Contact Name Full
Contact Phone Cell
Contact Phone Home
Contact Phone Work
Record ID
Record Status Creation Date
Record Status Creation Time
Record Status Creation User
Record Status Modification Date
Record Status Modification Time
Record Status Modification User
Record Notes
```

Not everyone will appreciate this approach, and certainly there is the risk of getting carried away and adding too many prefixes. However, some sort of visual organization to field names can be a huge efficiency boost for developers when navigating around the programming interface of a complex database.

Managing Fields

From the Fields tab of the Manage Database window, fields can be selected, added, renamed, modified, and deleted.

Selecting Fields

Many field management functions require a selection of at least one field. Selections can be made in the following ways:

- To select a *single field*, click on it with the cursor. To change the selected field, click on a different field or use the Up and Down arrow keys to select the previous or next field.
- To select *multiple consecutive fields*, hold down the Shift key and click on two fields at the outside of the range desired.
- To select *multiple non-consecutive fields*, hold Command (macOS) or Windows (Windows) and click on each desired field, one at a time.
- To select *every field*, click on any one field and type “A” while holding Command (macOS) or Control (Windows).

Adding Fields

There are four methods for adding fields to a database file:

- Creating a new field.
- Duplicating an existing field.
- Importing records (Chapter 5, “Selecting a Target Table”).
- Importing a table (Chapter 7, “Importing a Table”).

Creating a New Field

Manually creating a new field from the Fields tab of the Manage Database dialog is a quick and easy process:

1. From the pop-up menu, select the Table into which the new field will be added.
2. Type the name of the new field into the Field Name text area.
3. Select the data type for the new field from the Field Type pop-up menu.
4. Click the Create button.

The new field should appear in the list, sorted with any other fields already defined. If the new field is a Summary or Calculation field, FileMaker will immediately open the respective options dialog for the selected type (described later in this chapter).

Duplicating an Existing Field

When an existing field in any table is similar enough to the requirements of a new field, consider creating a duplicate of the field rather than starting from scratch. There are two ways to achieve this, depending on the final location of the duplicate being made.

Using the Duplicate Button

To duplicate a field into the same table, follow these steps:

1. Select the field(s) that will be duplicated.
2. Click the Duplicate button.

The new, duplicate field(s) will appear in the field list, highlighted as shown in Figure 8-2. It will have all options configured identically to the original and be named the same with a suffix of “Copy” added to ensure it is unique within the table. However, any content that the original field may have contained across all existing records will *not* be copied into the duplicate during this process, so the new field will be empty for existing records.

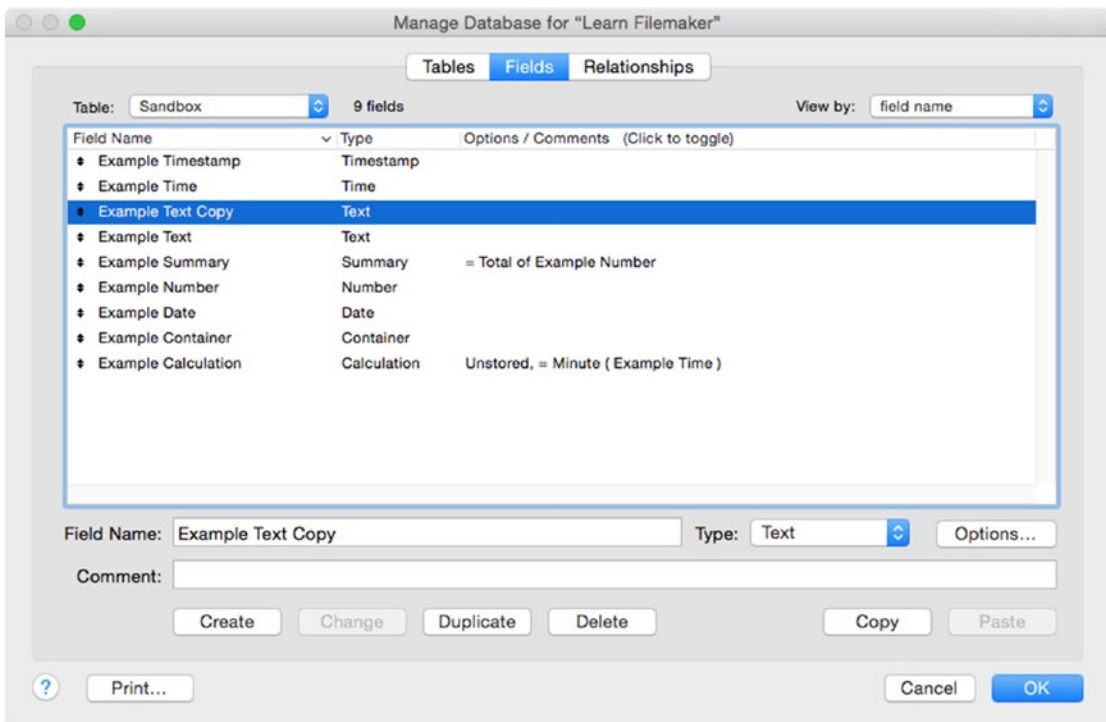


Figure 8-2. A duplicate of the Example Text field is selected

Using Copy and Paste

A field can be duplicated using the Copy and Paste buttons or menu commands. This not only works to duplicate fields *within* a table but can duplicate fields from one table to another and even between tables in two different database files.

To duplicate a field this way, follow these steps:

1. Select the field(s) that will be duplicated.
2. Copy the field(s) by clicking the Copy button.
3. If the destination table is different from the source table, select the source table from the Table pop-up menu. If the destination table is in a different file, close the Manage Database dialog and navigate to the appropriate table's field list in the other file.
4. Paste the field(s) in the clipboard by clicking the Paste button.

The new, duplicate field(s) will appear highlighted in the list. The name of any new field will vary depending if the destination table already contains a field with the same name. If it does, the new field will have a numeric suffix appended. If it does not, the new field will be named the same as the original.

Modifying a Field Name

Changing the name of an existing field is a three-step process.

1. Select the field to be renamed.
2. In the Field Name text area, type the new name for the field.
3. Click the Change button.

Once complete, the name of the field in the list should be updated. Click the OK button to close the dialog and fully commit to this change.

■ **Note** Changing the field's *comment* can be accomplished by following this same process.

Modifying a Field Type

Field type changes should be a rare occurrence, since the data entered in fields generally dictates the types. Minimal planning and attention paid while creating fields will help to avoid the need to change the type later. However, it may become necessary because of a mistake or a structural change to the database. To modify a field's type, follow these steps:

1. Select the field to be modified.
2. Select the new Field Type from the Type pop-up menu.
3. Click the Change button.

Depending on the field type selected, FileMaker may warn that any content already present in the field will be converted to the new type across all existing records. This includes automatic changes such as:

- Changing *to* a date or timestamp will convert dates previously entered as text into formal dates and convert any two-digit years to four digits.
- Changing *from* a container to another data type will remove any non-textual data from the field. For example, the actual embedded file or image data would be removed leaving only the name or path of the file.
- Changing *to* a calculation or summary field will result in the permanent loss of any previous input in favor of the new, automatically calculated result.

■ **Caution** When no warning dialog is displayed, any data in existing records will be automatically converted to the new type and should be fine. However, it is a good idea to at least spot check to confirm this fact.

Deleting Fields

One or more fields can be deleted by following this process:

1. Select the field(s) to be deleted.
2. Click the Delete button.
3. A warning dialog, like that shown in Figure 8-3, will confirm that the deletion process should continue.



Figure 8-3. The warning dialog when deleting fields

4. Click the Delete button to continue with the deletion, or the Cancel button to halt the process.
5. Once completed, the selected fields(s) will disappear from the list. Click the OK button in the Manage Database dialog to fully commit to this change and permanently delete the field(s) and any data previously entered.

FileMaker should clean up any instances of the field on layouts. However, you may need to manually delete any uses of the now <missing field> references found on layouts in other files, script steps, and formulas. Also, any field labels on layouts that were previously edited from the actual name of the field will remain and must be manually deleted.

Setting Field Options

Beyond the obvious properties of a field's name, type, and comment, there are many other options that can be configured through one of three dialogs, depending on type of field being edited:

- Entry fields
- Calculation field
- Summary field

To edit a field's options, select the field in the Manage Database dialog and click the Options button.

Options for Entry Fields

All entry fields — text, number, date, time, timestamp, and container — have similar options that only vary slightly from one to the next. When one of these types of field is selected and the Options button clicked, an Options for Field dialog will open, making available numerous settings spread across four tabs:

- Auto-Enter
- Validation
- Storage
- Furigana

Auto-Enter Options

The Auto-Enter tab of the Options for Field dialog, shown in Figure 8-4, contains settings that control specific data that will be automatically entered into the field when a new record is created or when certain conditions are met.

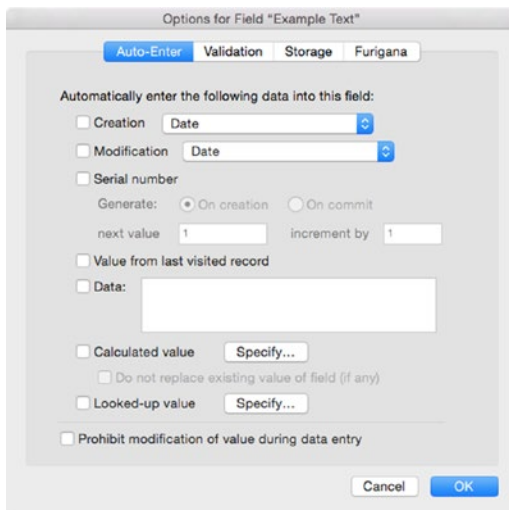


Figure 8-4. The Auto-Enter tab of the options dialog for an entry field

Automatically Enter the Following Data into This Field

Some of these settings are limited or disabled for certain entry fields depending on the specific data type they contain.

The available options include:

- **Creation** — automatically enter one of the creation options available in the adjacent popup: Date, Time, Timestamp, Name, or Account Name.
- **Modification** — automatically enter one of the modification options available in the adjacent popup: Date, Time, Timestamp, Name, or Account Name.

- **Serial Number** — configure the entry of a serialized number into the field based on the value in the `next value` text area that will automatically increment for each new record based on the value in the `increment by` text area.
- **Value from last visited record** — automatically enter a value into the field for a new record from the same field on the last record the user viewed.
- **Data** — automatically enter the static text value in the adjacent text area.
- **Calculated value** — used to define a calculation formula that evaluates and places a result into the field. The formula is entered in a `Specify Calculation` dialog (Chapter 12), which is accessible by clicking the adjacent `Specify` button. When checked, the `Do not replace existing value of field (if any)` check box below will cause the field to *not* evaluate the calculation if the field already has a value from an option above or contains data previously entered by the user. Calculated values are auto-entered when a record is created and when key fields used in the formula are modified.
- **Looked-up value** — used to copy the value from a related field into this field. Values are inserted when a record is created, immediately after the relationship is established. Once copied into a field, the value is retained regardless of the state of the related record and can be edited by users and scripts (assuming access and layout settings). However, the value can be updated when the `Records > Relookup Field Values` menu item is selected.

■ **Caution** Although all seven options are check box controls, the first five operate as radio buttons, allowing only *one* to be selected at a given time. The last two operate as radio buttons with each other but can be combined with the first set as needed.

Prohibit Modification of Value During Data Entry

When this check box is selected, it makes the field non-editable at the data level regardless of a user's security settings or the field's configuration on a layout. After any auto-enter option(s) have been performed the field will become non-editable.

Lookup for Field Dialog

When the `Looked-up value` check box is selected on the `Auto-Enter` tab of the `Options for Field` dialog, a `Lookup for Field` dialog, shown in Figure 8-5, will open allowing the selection of a related field that will be looked up through what relationship.

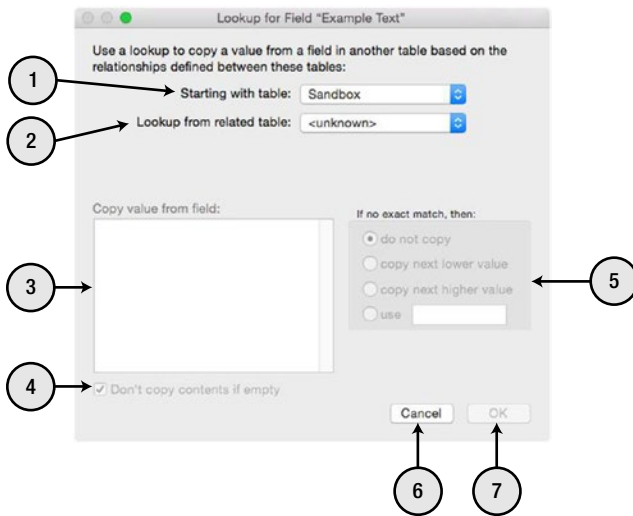


Figure 8-5. The *Lookup for Field* dialog

The *Lookup for Field* dialog contains the following controls:

1. **Starting with table** — This pop-up menu contains a list of every table occurrence that exists in the file for the table of the field being defined. The selection establishes the starting context, the table containing the field being defined.
2. **Lookup from related table** — This pop-up menu contains a list of every table occurrence that is related to the starting table selected above. The selected occurrence identifies the relational conduit through which a field value will be copied.
3. **Copy value from field** — This area will display a list of all the fields in the table occurrence selected in the *Lookup from related table* pop-up. Select the field whose value should be copied into the field being defined.
4. **Don't copy contents if empty** — Check this box to halt the lookup process if the selected field in the related record is empty. This will avoid deleting a value in the local field in favor of an empty value in the related field.
5. **If no exact match then** — These options allow an alternate course of action if no related record is found:
 - a. **Do not copy** — will not attempt to copy any value and will not modify the local field at all.
 - b. **Copy next lower value** — will look through a relationship for a record with the next lower value in the source field.
 - c. **Copy next higher value** — will look through a relationship for a record with the next higher value in the source field.
 - d. **Use** — will enter the literal text typed in the adjacent text area into the local field.
6. **Cancel** — Click this button to close the dialog without saving changes.
7. **OK** — Click this button to save changes and close the dialog.

Validation Options

The Validation tab of the Options for Field dialog, shown in Figure 8-6, contains settings that define the data entry requirements for the field, allowing the database to automatically validate what the user enters and enforce certain rules. When a field fails a validation check, FileMaker will notify the user of the error and give them a choice of subsequent action.

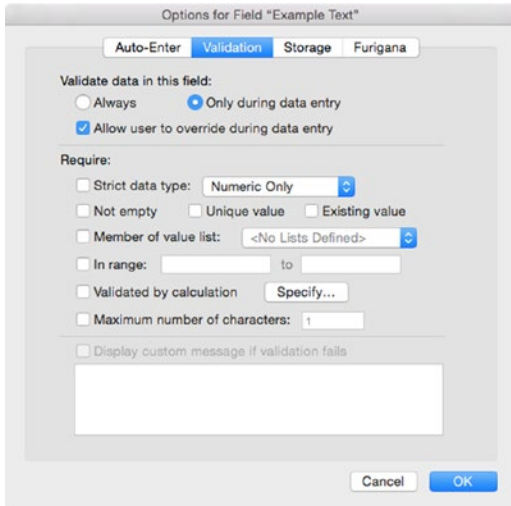


Figure 8-6. The Validation tab of the options dialog for an entry field

Validation Control

The options at the top of this dialog control when field content is validated and whether a user can override a validation warning. The two options are:

- **Always** — When selected, data will *always* be validated as it is entered by a user or when importing records.
- **Only during data entry** — When selection, data will *only* be validated as it is entered by a user.

■ **Caution** When importing records with validation set to **Always**, if a field can't be validated, FileMaker will *not* import the record and report it in the **Import Results Dialog** as a count of the total incidence of import errors without providing any additional detail!

Checking the Allow user to override during data entry will change the validation warning dialog as shown in Figure 8-7.



Figure 8-7. The two versions of the validation warning dialog

With the override option *unchecked*, the only options in the warning dialog are:

- Revert Field — removes the recent data entry changes and restores the previous value to the field.
- OK — returns the user to the uncommitted record where they can edit their entry to comply with the validation requirements before attempting to commit again.

With the override option *checked*, the Revert Field option remains the same but the OK option transforms into a Yes/No button combination and the message text changes, phrased as a question rather than a strict statement:

- No — Does the same thing as the OK button in the other version of this dialog, returning the user to the uncommitted record where they can edit their entry.
- Yes — Clicking this button will ignore the validation warning, accept the changes as entered, and continue with the Record Commit process.

Validation Requirements

This series of check boxes on the lower portion of the Validation tab of the Options for Field dialog allows one or more of the following requirements to be imposed on data entry in the field:

- Strict data type — This option restricts data entry to one of the following options from the adjacent pop-up menu: Numeric Only, 4-Digit Year Date, or Time of Day.

■ **Tip** The 4-Digit Year option forces them to enter a date with a full year *before* FileMaker performs its automatic date conversion based on a 100-year span. This can be helpful in avoiding entry errors and inadvertent conversions for fields that may contain dates outside of that span.

- Not empty — This option requires a value to be present in the field before the record can be committed. It can contain any value but must contain some value.
- Unique value — This option requires that the value entered in the field for the current record be unique for the same field across *all* records in the table.

- **Existing value** — This option requires that the value entered in the current record already exist in the same field for at least one other record in the table.
- **Member of value list** — This option requires the value entered in the field to be present within the value list selected from the adjacent pop-up menu. This feature can stop a user from typing or pasting a value into a field to bypass the assigned value list.
- **In range** — This option requires that the value entered fall within a range mandated by the entries in the two adjacent text areas. This works well not only with numeric values but with text, dates, and times.
- **Validated by calculation** — This option allows a totally customized validation by way of a formula (Chapter 12) entered by clicking the adjacent *Specify* button. The formula entered *must* evaluate to `true` (non-zero) for the field to pass the validation test.
- **Maximum number of characters** — This option requires that the value entered have a character length that is equal to or less than the number entered in the adjacent text area. This option is not available for container fields.
- **Maximum number of kilobytes** — This option replaces the one above for container fields, allowing you to specify in the adjacent text area an upper limit on the size of a file that can be placed into the field.

Display Custom Message if Validation Fails

When the validation of a field entry fails, FileMaker will generate a dialog message like those shown previously in Figure 8-6. The default message will include details about the specific validation failure, which can be helpful when multiple validation criteria are applied to a single field. However, for any circumstance where a custom message is desired, check this box and type the message below.

Exploring Validation Alternatives

There are alternatives to using FileMaker's validation options that some developers prefer. For example:

- Some entry errors can be *automatically corrected* with an Auto-Enter > Calculated value formula that analyzes and modifies the data instantly after the user enters it. This allows the database to instantly correct certain common mistakes rather than report them and burden the user with such things.
- When an error can't be auto-corrected, another alternative is to put a calculation field on the layout that displays a list of all the validation errors detected within the record. This removes the obtrusive dialog but makes it possible for the user to ignore the error that may not be acceptable in some situations.
- A `OnObjectValidate` script trigger (Chapter 37) assigned to a field can run a script (Chapter 34) that checks the field for any validation errors. The script can then perform a combination of auto-correction and/or reporting with a fully custom dialog before returning the user to the field until they correct the problem(s) detected.
- Similarly, an `OnRecordCommit` script trigger can perform a script that checks the entire record for any validation errors and then auto-correct and/or report.

Storage Options

The *Storage* tab of the Options for Field dialog, shown in Figure 8-8, contains options that control various aspects pertaining to how information is stored within a field.

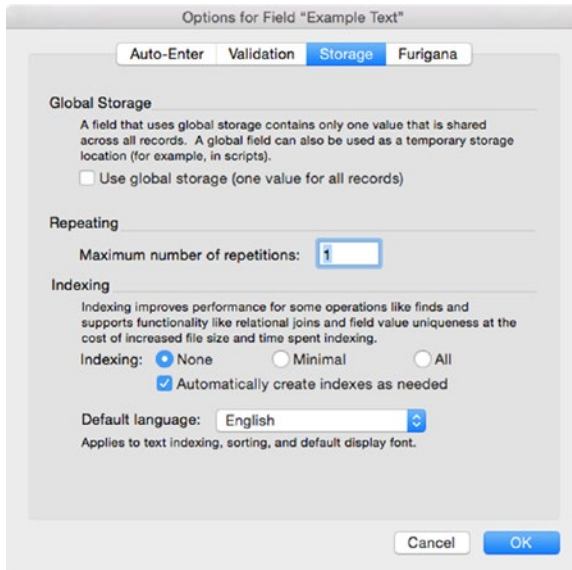


Figure 8-8. The *Storage* tab of the options dialog for an entry field

Global Storage

When first created, every field is local. Being a *local field* means the field is essentially replicated as a unique instance of itself for each record, like a cell in a spreadsheet. The value placed within the field is *local* to a single record, which is separate and different from the same field on the next record.

By contrast, a field can be defined as a *global field*, which is a field that contains one value that will be shared by every record in the database. The value placed within the field is *global* to every record and usable anywhere in the file. Using the spreadsheet analogy where a column represents a defined field and a row a record, a global field is like a column with the same value in every cell for every row.

Check the `Use global storage` check box to convert the field to use global storage. When converting a previously defined field to a global FileMaker will present a warning dialog reminding you that any values contained within the field will be lost when the change occurs.

Global fields have various purposes, including:

- Making a fixed value available to *any* calculation formula regardless of the context or a lack of a relational connection.
- Creating a popup menu on a layout to allow user access to actions or options. Being a global field means a selection can remain the same for any record in the table and can even be added to unrelated tables.

- Providing temporary storage of information used by or assembled by scripts. Although, except for exporting, variables can usually be more effective for this purpose.
- Acting as the match field for a parent table in a relationship to vary the match criteria based on user input or other factors.
- Storing static text or graphics that will be displayed on a layout as iconography, branding, or other purposes.
- For storing temporary print report headings or other information.

Global field values are persistent from one session to the next only when entered locally with the database open in a copy of FileMaker Pro or FileMaker Pro Advanced. That value will be the default value each time the file is opened across a network from a FileMaker Server. Any changes made by one user are only applicable to their session and do not persist in future sessions.

■ **Tip** In the past, global fields were more heavily used for a variety of functions that are better managed with variables (Chapter 12), merge variables (Chapter 30), and script parameter passing (Chapter 34). Global fields should not be used for any task that is more efficiently handled using these modern features.

Repeating

A *repeating field* is any field defined to store multiple values for a single record. Instead of creating new, separate fields for additional values of a certain type, a single field can be defined *repeat* any number of times.

To make any type of *entry field* repeat, enter the number of desired repetitions in the text area provided on the Storage tab of the Options for Field dialog. That number indicates the maximum number of separate values the field can contain. To make a calculation field repeat, enter the number of repetitions in the Specify Formula dialog for a calculation field (Chapter 12). Summary fields can be configured as repeating in an unusual way, described later in this chapter.

Field repetitions have many uses such as storing multiple phone numbers or email addresses into a single field. However, there are a few implications that should be considered, including:

- Whatever upper limit you choose for the field and make accessible on a layout will establish the *maximum* number of values that can be typed into the field. You *can* increase this number later, but then you need to make the additional repetitions visible on any relevant layouts, which may require rearrangement since you can't make repeating values scrollable within a finite layout space.
- A Find process will search in all repetitions, including those not displayed on the current, or any layout. It is not possible to limit a find to just a specific repetition.
- A Sort process will *only* look at the first repetition.
- Calculations that use a combination of repeating and non-repeating fields may not work together without the usage of the Extend function (Chapter 19).
- Auto-enter calculation will only work on the first repetition.
- Repeating field values can't be easily summarized when creating reports. You need to condense them into a single value with a calculation using an aggregate such as List (Chapter 19).

Due to the above limitations and their status as a somewhat archaic tool, repeating fields are largely superseded today by portals (Chapter 30) to display fields as a scrollable and expandable list of related records. While there are still valid uses for repetition, it should be avoided in most cases.

Indexing

A *field index* is a list of words or values generated from a non-global field's contents behind the scenes by FileMaker that are used when performing searches, determining the uniqueness of a field value, and connecting records through relationships. Although index data will increase the file size of a database, they are required by the nature of these actions and desirable to improve performance. An index for a database field serves a function similar to those we are familiar with at the end of physical books. The value of a database index can be analogously grasped by anyone considering the different amount of effort required to find a subject in a large book by scanning each page from front to back instead of finding the topic in the book's index and flipping to the indicated page number.

FileMaker uses two different kind of indexes, depending on the type of field and the option selected:

- A *value index* is created from every carriage return-delimited line in a text, number, date, time, and timestamp field, including calculation fields that return any of these data types. This index is used to match related records and to search any field type except text. Here, it helps to think of "value" as a paragraph.
- A *word index* is created from each unique word in a text field and a calculation field that returns a text value. It is used only for searching.

Indexing options on the Storage tab of the Options for Field dialog include:

- Indexing: None — This option prevents FileMaker from indexing the field at all. Use this for fields that don't require searching and won't be used to form relationships.
- Indexing: Minimal — This option, available for text fields only, will limit FileMaker to creating a *value index* for the field.
- Indexing: All — This option will create a *value index* for number, date, time, and timestamp fields and both a *word and value index* for text fields.
- Automatically create indexes as needed — This check box allows FileMaker to automatically create indexes as needed. For example, when a user performs a search on a previously unindexed field set to None, checking this box would allow it to switch automatically to All and generate and store the index.
- Default language — This pop-up menu allows specification of the language to be used for indexing and sorting values in a text field. Initially the selected value will be that of the operating system language of the computer upon which the file was created.

■ **Tip** There will always be situations where you might need to customize the indexing settings. However, in many cases, you can leave the default indexing settings as they are and FileMaker will adjust them as needed based on user and developer activity.

Container Storage Options

Container fields can't be indexed so, the Storage tab of the Options for Field dialog does not have the above options available when editing them and instead display *container storage options* (Chapter 10).

Furigana Options

The Furigana option specification of a field that will receive a phonetic translation of Japanese text typed into the field. For more information on Japanese functions, see FileMaker Pro’s documentation website.

Options for Display Fields

Calculation and summary fields are different than other field types and their options reflect this fact. Clicking the Options button or double-clicking on a calculation or summary field will open an alternative dialog of options specifically focused for the field type.

Calculation Field Options

The Specify Calculation dialog allows the entry of a calculation formula and selection of the resulting value’s data type, number of repetitions, storage options, and an option to not evaluate if all referenced fields in the formula are empty (Chapter 10).

Summary Field Options

The Options for Summary Field dialog, shown in Figure 8-9, allows specification of the type of summarization and field to be summarized as well as other options.

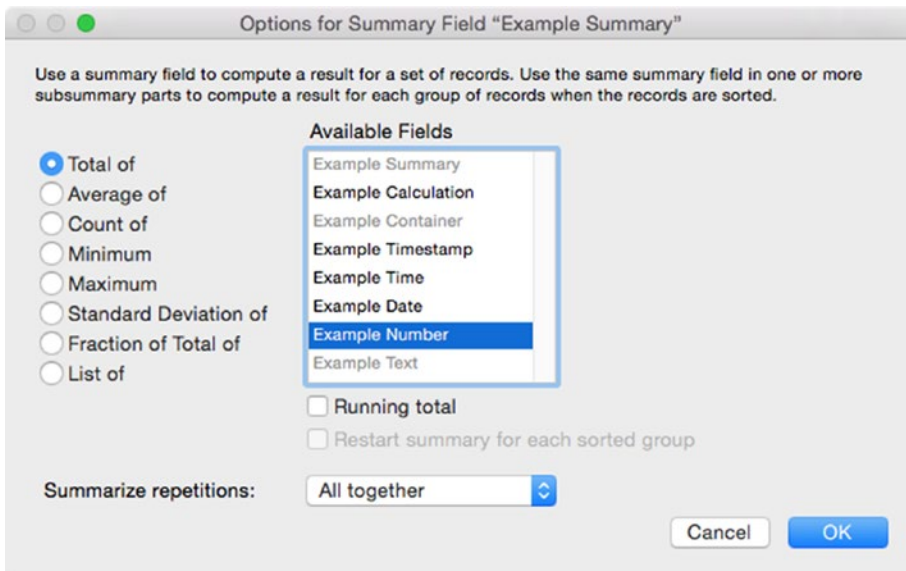


Figure 8-9. The Options for Summary Field dialog

Summarization Types

The following summarization options are available in a set of radio buttons along the left side of the dialog:

- **Total of** — calculates the total value for the selected field in the context-set.
- **Average of** — calculates the average value for the selected field in the context-set.
- **Count of** — counts the number of records in the context-set that have any value.
- **Minimum of** — extracts the lowest value in the selected field for the records in the context-set.
- **Maximum of** — extracts the highest value in the selected field for the records in the context-set.
- **Standard Deviation of** — calculates the standard deviation from the mean of all the values in the selected field for the records in the context-set.
- **Fraction of Total of** — calculates the ratio of the value in the selected field to the total of all the values in that field.
- **List of** — creates a carriage return-delimited list of every non-blank value in the selected field.

Options for Summary Field Dialog Overview

The **Options for Summary Field** dialog can be somewhat deceptive in appearance. There are many conditional controls and a surprising variety of functionality available. The dialog can be divided into five general sections, shown in Figure 8-10.

■ **Caution** This section contains an overview description of the functions for *defining* the summary field but may be rather abstract until one integrates it with an understanding of how these fields behave when placed on different layout parts (Part 4).

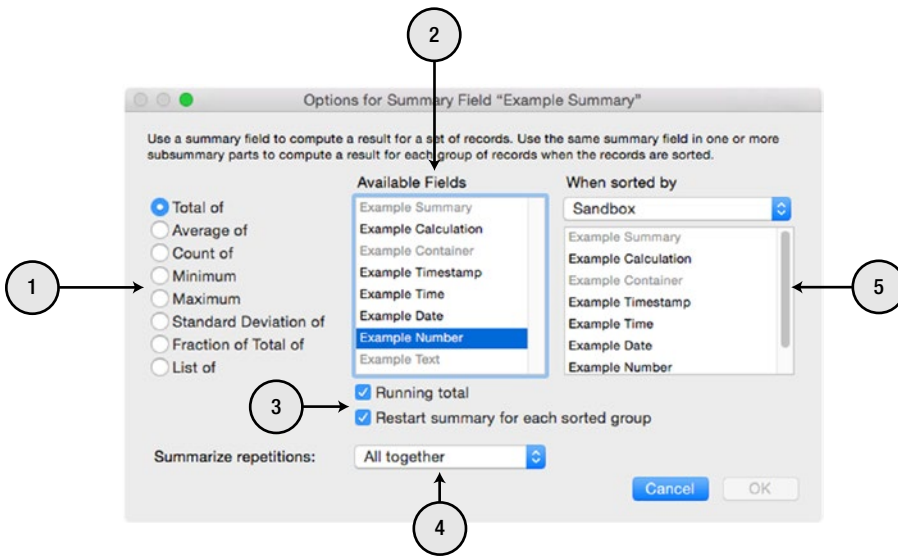


Figure 8-10. The Options for Summary Field dialog with additional options visible, marked up for a sectional overview

The Options for Summary Field dialog contains the following controls:

1. *Summarization Type* — Choose the type of summarization that will apply to the selected field to generate a value for the field you are defining. Depending on your choice, the conditional options will vary.
2. *Available Fields* — This list displays the available fields for the table in which the summary field being defined exists. Ineligible fields, such as text, container, and other summary fields will be gray and non-selectable. Click to select the field that should be summarized into the field being defined.
3. *Conditional Options* — This area will display either *one*, *two*, or *no* options depending on which summarization type is selected. The potential options in this area include:
 - Running total
 - Restart summary for each sorted group
 - Weighted average
 - Running count
 - By population
 - Subtotaled
4. *Repeating Field Handling* — This pop-up menu controls how a selected repeating field will be handled: All together or Individually. Selecting Individually in this pop-up menu will cause the summary field to be displayed with repetitions matching the selected field. Choosing All together will cause the sum of all the values in repetitions of the field for a given record to be treated as a single unit in the summarization.

5. *Conditional Secondary Field Choice* — Depending on the options selected, a second list of fields will appear in this area allowing the selection of a secondary mathematical function that should occur within the found set of records being summarized. The options here are:
 - a. *When sorted by* — Some summarization options allow the selection of a field in the same table or a related table that will restart the summarization when a new sort value is detected. This creates summaries for multiple subgroups of records within a single found set. For example, you might want to summarize sales per employee or revenue per client using this feature. This option is available with the *Total of* or *Count of* summarization with the *Restart summary for each sorted group* option selected, or when the *Fraction of Total of* summarization is used with the *Subtotaled* option selected.
 - b. *Weighted by* — When using the *Average of* summarization type with the *Weighted average* option selected, a field can be selected that will be used to weight the results.

Expanding the Learn FileMaker Object Model

To provide an example of planning fields and to keep the Learn FileMaker example file updated with resources that will be required later in this book, revisit the sparse object model we outlined in the last chapter with three tables: *Company*, *Contact*, and *Project*. Using the information presented in the current chapter, we can expand this model to include a list of some of the fields required for each and then add these to the database.

It isn't important that one plan *every* field for every table before starting to develop a database. However, doing so can help you visualize the proposed system and better anticipate other resources that may be needed. For now, we will list some basic fields that make sense in tables of these types. Define these fields in your copy of the database but don't worry yet about putting them on layouts.

Defining Standard Fields

Every table will have these standard fields.

■ **Tip** You might consider creating the *Company* table with standard fields only and then duplicating that for each additional table before adding the custom fields below to each.

- *Record ID (text or number)* — This field will contain a unique identification number for the record, defined to receive an auto-enter value of a serial number, starting at some value and incremented by 1 for each subsequent record. Modification should be prohibited.

■ **Tip** Serial numbers can be text or numbers, but whichever you use you should be consistent throughout a database. Keeping all values the same width can be visually pleasing for developers and any users who may be exposed to them on layouts. If you choose to format these as a number, consider starting at the first number that contains the number of digits desired; start at 1000 for four-digit numbers. To start a sequence at 1, use text formatted serial numbers with a leading zero; start at 0001. The examples in the rest of the book will assume six-digit, text-based serial numbers with leading zeros.

- Record Status Creation Date (*date*) — Receive an auto-enter value of the creation date and prohibit modification.
- Record Status Creation Time (*time*) — Receive an auto-enter value of the creation time and prohibit modification.
- Record Status Creation User (*text*) — Receive an auto-enter value of the creation user name and prohibit modification.
- Record Status Modification Date (*date*) — Receive an auto-enter value of the modification date.
- Record Status Modification Time (*time*) — Receive an auto-enter value of the modification time.
- Record Status Modification User (*text*) — Receive an auto-enter value of the modification user name.
- Record Notes (*text*) — A freeform text field allowing users to enter notes about the entity represented by the record or the record itself.

Defining Custom Fields

Each table in the Learn FileMaker database should have at least a few custom fields defined that are specific to the type of entity the table represents. While not comprehensive enough for a real world solution, these fields will provide enough of an example for our continued exploration.

Defining Company Fields

A company table should have some obvious entry fields:

- Company Name (*text*) — The full name of the company.
- Company Name Code (*text*) — A short code that can be a stand-in for the company name in cases where that is beneficial. Perhaps a stock ticker symbol for companies that are publicly traded.
- Company Industry (*text*) — The name of the industry in which the company operates.
- Company Website (*text*) — The company website address (URL).

Defining Contact Fields

A contact table should have some obvious entry fields:

- Contact Name First (*text*) — The first name of the person.
- Contact Name Last (*text*) — The last name of the person.
- Contact Name Honorific (*text*) — The title prefixing the person's name: Mr., Mrs., Ms., etc.
- Contact Company ID (*text*) — The ID of the parent Company record to which a contact is connected.
- Contact Address Street (*text*) — The street address of the person.
- Contact Address City (*text*) — The city where the person lives.
- Contact Address Zip (*text*) — The zip code where the person lives.
- Contact Address Country (*text*) — The name of the country where the person lives.

Defining Project Fields

A project table should have some obvious entry fields:

- Project Company ID (*text*) — The Record ID of the parent company.
- Project Name (*text*) — The name of the project.
- Project Budget (*number*) — The dollar amount of a project budget.

Summary

In this chapter, we discussed the basics of working with fields.

CHAPTER 9



Connecting Tables with Relationships

FileMaker Pro databases can be relational. A *relational database* allows tables to be interconnected to form relationships between records that act as a conduit for data access.

This chapter covers the following topics pertaining to relationships:

- Introducing relationships
- Managing data sources
- Introducing the manage database dialog (Relationships)
- Working with table Occurrences
- Building relationships
- Adding notes to the graph
- Organizing the learn FileMaker object model

Introducing Relationships

Relationships are powerful mechanisms of *integration* and add a new dimension to data management. Without relationships, a database is like a bunch of separate documents containing one or more separate lists. By virtue of being contained within a single database file, these lists are certainly easier to search, sort, print, edit, and share. The layout and scripting functions allow information to be merged into various visual formats and automatically manipulated in many useful ways. However, without relationships, the lists are separated from each other and can't easily interact or share information. Relationships allow data to be dynamically connected.

Defining Relationships

A *relationship* is a single, bidirectional connection that integrates two tables. Relationships establish the criteria for matching records between two tables with one or more field pairs called match fields.

A *match field* is any field that is used to form one side of a relationship. Two match fields, conjoined with an operator, form a single criterion for establishing a relationship between records. In this example, a Company record will be related to a Contact record when the former's Record ID field matches the latter's Contact Company ID field:

Company::Record ID = Contact::Contact Company ID

To be established, a relationship must contain at least one pair of match fields. However, they can be made up of various such pairs. When more than one criterion is defined, records will only match when they *all* criteria return a true value.

While match fields can contain any type of value, most relationships are established with one set containing serial numbers. These are formally referred to as local keys and foreign keys. When a match field containing a serial number is native to a table, such as the Record ID representing a company record in a Company table, it is referred to as a *local key* or a *primary key*. This means that the value in the field represents the local record. Conversely, when a match field contains a serial number that represents a record from the table on the other side of the relationship, it is called a *foreign key*. This means that the value in the field represents a record in a different table. For example, a Company ID entered into a field in a Contact table is a value from the other side of the relationship and is, therefore, foreign to the Contact record.

There are some strict indexing requirements for match fields containing foreign keys. While a field containing a primary key need not be indexed and can even be a global field, those containing foreign keys *must* be indexed and, therefore, cannot be global fields.

Relationships are viewed and managed in a graphical worksheet area called the *relationship graph*, which is where table occurrences are placed, connected, and configured. A *table occurrence* is one instance of a table represented on the graph by a visual object. Occurrences are pulled in from a *data source*, which is a file containing the table. The data source can be the default instance of the database you are working within or accessed from an external FileMaker file or ODBC compliant database.

By default, every FileMaker table is represented in the relationship graph by one occurrence automatically when created. However, they can be added into the graph as many times as necessary to each provide a unique relational context. This will be necessary when creating multiple relationships between the same set of tables to avoid forming a circular structure, which is strictly forbidden.

Later, when developing user interfaces, layouts are assigned a table occurrence to establish the records it will display and its context in relation to other connected occurrences. The context of an occurrence is important to remember when designing layouts and writing scripts as it will control which fields in other tables are accessible from a given layout. Every layout must be assigned a table occurrence; however, it isn't mandatory that every table occurrence is assigned a layout unless the context it establishes requires an interface for user or script access. In cases where a table doesn't require *any* occurrences, the default instance can be deleted.

Generally, relationships connect two *different* tables to form a hierarchical connection. However, it is possible and sometimes beneficial to connect a table to another instance of *itself* to form connections between *records* within the *same table*.

Table occurrences can be joined together into a chain of successive relationships, as illustrated in Figure 9-1. This is sometimes referred to as *multiple hops*, referring to the number of occurrence steps removed from the first.

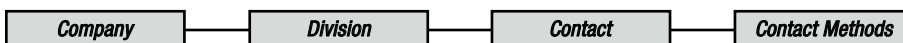


Figure 9-1. A diagram showing an example of a relationship chain with four “hops”

Relationships within a file can be grouped into one multi-pronged chain or into multiple separate chains depending on the specific needs of the database and your preferences as a developer.

Defining Uses for Relationships

Once a connection is established between two tables, records and fields from one can be accessed from the other. This has many positive implications for fields, layouts, scripts, and users.

- *Fields* can be defined to copy content from another field in a related record as easily as they can from fields within the same table. This includes:
 - Using the Lookup command.
 - Pulling information into an auto-enter Calculation value.
 - Using the Validated by calculation option.
 - Calculation fields can access and manipulate related fields in their formulas.
- *Layouts* can display individual fields from related records for viewing or editing. Using portals, they can display scrollable lists of related records. They can also use values from related fields within calculation formulas for conditional formatting, placeholder text, script parameters, tooltips, and hide functions (Part 4).
- *Script steps* can reach through relationships connected to the context of the current layout's table occurrence to pull information into formulas, change related values, sort, search, and more (Part 5).
- *Users* can include fields from related records in search requests, sort orders or exports depending on their level of security access (Chapter 40).

Defining Connection Types

There are three types of relational connections that each focus on a quantitative potential for matches. These are not governed by settings in FileMaker but are conceptually useful to understand when working with relationships. They are established by the nature of the interconnection and/or the limitations imposed in layout and script configurations. The three types are:

- One-to-one
- One-to-many
- Many-to-many

One-to-One Relationship

A *one-to-one relationship* is a type of connection where any one record in either table can only be connected to a single record in the other table. Essentially, this means that the foreign key is limited to a single use of any one primary key in the related table. In the example illustrated in Figure 9-2, the Contact table's primary key is contained in the Record ID field. The id for each contact record can only be assigned to a single Cubicle record's Contact ID field. Therefore, each Contact can be assigned one Cubicle and each Cubicle on Contact.

Contact		Cubicle				
Record ID	Contact Name	Contact ID				
00001	John Smith	00025				
00002	Alex Martin	00002				
00003	Jane Doe	000152				

Figure 9-2. Illustrating a one-to-one relationship between a Contact and Cubicle table

The one-to-one arrangement is perhaps the least common of the connection types since it can result in the equivalent of two sets of information in separate tables for the same entity. While one shouldn't use such a technique arbitrarily, there are situations where a one-to-one configuration makes sense.

For example, consider the case illustrated in Figure 9-2 with a Contact table connected to a Cubicle table for office space assignments. In this case, even though one person can only be assigned to a single cubicle and each cubicle is designed for a single individual, there are still reasons to keep the two sets of fields in *separate* tables. First, they are two different types of entities, each with different properties. Further, over time, people tend to change offices due to job changes or during a reorganization. If the fields describing a cubicle were merged into the Contact table, it would not only confuse the entity class being described by making a cubicle a property of a person, but also create extra work when a person moves to a new location. By keeping them in separate tables, users avoid the tedious job of deleting and reentering cubicle information into a different Contact record and instead simply delete the identification number from one Cubicle record and reenter it on another, thereby linking the person to a new cubicle.

In this kind of situation, there is no clear requirement of which side of the relationship the primary key should be placed. A field from either side can be selected as the entry value for the other side. As a developer, you need to choose to build the database so *either* a cubicle is assigned to a person *or* a person is assigned to a cubicle, depending on your preference. Whenever the physical union between a contact and a cubicle changes in reality, a user would enter or select the new id assignment from whichever side of the relationship your interface design allows.

The one-to-one relationship is usually achieved by means of layout design restrictions. For example, requiring the foreign key field to be a unique value will make it impossible for a user to assign a user to multiple cubicles or a cubicle to multiple users.

Another less common reason for a one-to-one relationship occurs when a record is connected to itself. This may be done for error detection purposes or when generating a value list from a field value. Linking the unique serial number field between two occurrences of the same table can *only* result in the establishment of a one-to-one relationship.

One-to-Many Relationship

A *one-to-many relationship* is a type of connection where one record in the primary key table can be connected to many records located in the related table. In the example illustrated in Figure 9-3, the Company table's primary key Record ID field for a given record can be assigned to one or more Contact records in the foreign key Company ID field, while each Contact record is limited to a single company assignment. So, *one* Company record can be linked to *many* Contact records.

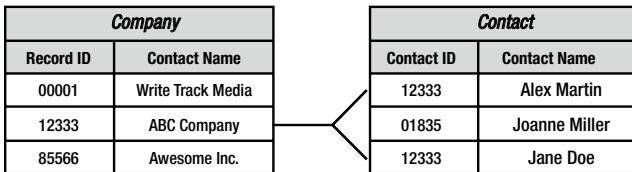


Figure 9-3. Illustrating a one-to-many relationship between a Company and Contact table

This setup is the most common of the three because there are so many entities that can be related to multiple entities in this way. A single company can have many employees, many products, many clients, and many projects. One person generally has many methods of contact in the form of phone numbers, email addresses, and web pages. A parent can have many children.

This last example is why this arrangement is often referred to as a *parent-child* type relationship since, biologically, a child only has a single father and a single mother but either parent can have many children. So, the table containing the primary key contains the parent record and the table containing the foreign key contains the child record(s). Technically speaking, this term can be applied to a one-to-one relationship as well, where the side of the relationship containing the primary key is referred to as the parent and the side containing the foreign key is referred to as the child.

With this arrangement, records can be assigned from either side of the relationship. From the parent record, a portal can be configured to allow the creation of multiple records; each assigned the parent's primary id as their foreign key. From an existing child record, a field can be configured to allow a parent's primary key to be entered as the foreign key. Both will assign a child record to the parent and form the relational connection.

A one-to-many relationship is controlled completely by layout design.

Many-to-Many Relationship

A *many-to-many relationship* is a type of connection where records in *both tables* can be connected to any number of matching records in the other. There are two methods of achieving this type of relationship:

- Using a join table
- Using a multi-key field

■ **Note** In addition to these, it is possible to create the equivalent of a many-to-many by establishing two separate relationships, a one-to-many and a reverse many-to-one. However, strictly speaking, this is not a relationship type since it is spread across two separate relationships.

Using a Join Table

A *join table* is a third table added between two tables with *two* foreign key fields that contains both primary keys from either of the other two tables being connected. Join tables are the most common method for establishing a many-to-many connection. In the example illustrated in Figure 9-4, the primary key from both Table 1 and Table 2 are each present in a foreign key field in the join table, thereby establishing one connecting record between the two. If the interface is designed to allow it, every record in the tables on either side can be connected to any number of records from the other through the join table.

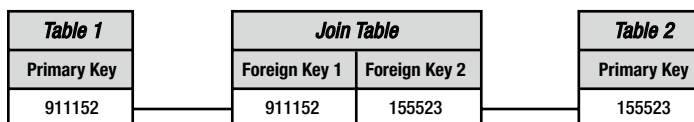


Figure 9-4. Illustrating a many-to-many relationship with the use of a join table

A many-to-many relationship has many practical uses. For example, when connecting people to a company, consider a situation where you would want to not only connect a person to their *current* employer but also keep track of *past* employers. This is possible with a join table as illustrated in Figure 9-5.

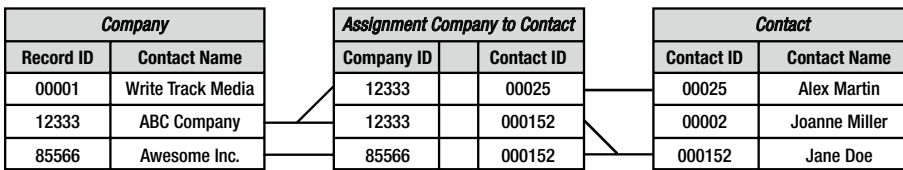


Figure 9-5. Illustrating an example of a practical many-to-many relationship

In this example, a Company table and a Contact table are connected through an Assignment Company to Contact join table. The ABC Company has two contacts connected to it, Jane Doe and Alex Martin. Jane Doe has two companies connected to her record: ABC Company and Awesome Inc.

In addition to allowing many-to-many connections, one of the benefits of a join table is that fields can be added in the join table that contain information specific to the *union* of the two records. So, instead of a value pertaining to a company or to a contact person, fields in the join table allow a value to be stored with the connection between a company *and* a contact. For example, a field that indicates which employment is current could exist in the join table and be visible from the Contact side, so Jane Doe would show two company connections, one “Active” and the other “Inactive.” Also, the contact’s current (or last) title at a company would need to be stored in a field in the join table since that would change from one job to the next.

Once the table occurrences are connected, a portal displaying records from the join table would be placed in both the Company and Contact tables allowing a record on either side to be assigned to any number of records on the other.

Using a Multi-Key Field

FileMaker allows the use of a multi-key or complex key list field to expand the possible matching values through a relationship. A *multi-key field* is a text field or a calculation field resulting in a text value that contains a carriage-return delimited list of keys and is used to form a relationship. This has the effect of creating a many-to-many relationship *without* the need of an intervening join table.

When a match field contains a list of values, each paragraph is treated as its own value and used to detect matching records in the other table, as illustrated in Figure 9-6.

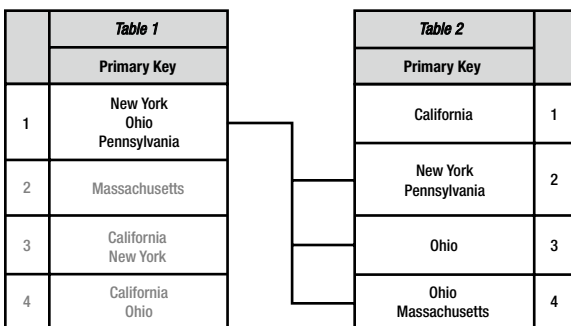


Figure 9-6. An illustration of a multi-key field relationship showing matched relationships for the first record in Table 1

In this example, we see the power of a multi-key field relationship. The primary key field in the first record in Table 1 matches three records in Table 2 even though none of the primary key fields in those records contain *all* the same values. When separated by a carriage return, each value is used to

form a match, so the presence of any one value from one match field found in the other will form a valid relationship. It is important to note that, in this example, Table 1 establishes the context perspective for the lines indicating a match. From a Table 2 perspective, for example, the fourth record in Table 2 would connect to both records 1 and 2 in Table 1.

The primary drawback to this technique is that, without a join table, there is no convenient location to add fields that store information about a specific union of two records. However, if a relationship doesn't need such fields, this technique can be enormously useful for logistical reasons.

One example is the ease of having a duplicate of a record retains all its relationships. Normally, when duplicating a record in a table, none of the join records will be duplicated so the new duplicate record will no longer be connected to the records in the other table unless a custom script is created and used to override the Duplicate menu item to explicitly perform that task. However, with a multi-key relationship, when the foreign keys of all the related records are contained in a single field located on the record you are duplicating, there is no additional development work required to maintain the connection to related records.

This technique is useful when creating a temporary storage and display mechanism of related information in a selection dialog where a cancel option needs to quickly wipe out the temporary relationship. With a single field in a temporary record containing multiple keys that form a relationship for an adjacent portal, it is easy to remove the relationship by deleting the keys from the field.

Organizing Relationships

There are two primary methods for organizing relationships in the graph:

- Hub-and-spoke monolith
- Table occurrence groups

■ **Tip** Although your choice between these will depend on your preferences and the complexity of the database you are building, the table occurrence group method is often chosen on principle for the sake of establishing an easily scalable relationship graph and to maintain good performance.

Defining Hub-and-Spoke Monoliths

The *hub-and-spoke monolith*, illustrated in Figure 9-7, is a method of organizing table occurrences by directly attaching them together, usually into one massive interconnected web, and utilizing any of the occurrences for the context of layouts. The use of the word monolith here refers partially to the lack of separation between groups but also to the lack of distinction between a *primary* and *secondary* instance of a table for calculation and layout context. It is possible to separate occurrences into groups and have them *still* be a hub-and-spoke that is only slightly less monolithic.

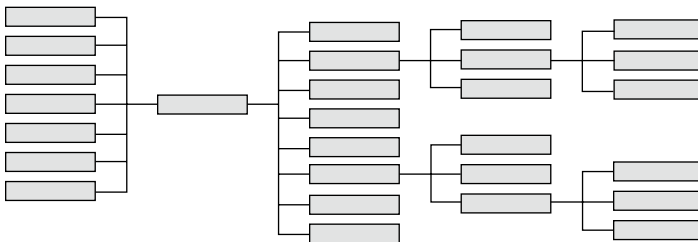


Figure 9-7. An illustration of the pattern of a generic hub-and-spoke monolithic relationship web

Due to the bidirectional nature of relationships, a field calculation or layout *can* use any occurrence in the graph for its context. For beginners or simple databases, this tends to be the default approach and is perfectly valid, although it is not recommended.

As more and more tables are added and interconnected, it can quickly become confusing for even the most skilled developer to keep straight which context is which, especially if secondary duplicate occurrences are poorly named. Each time an occurrence is duplicated, the default name is the same as the original with a numeric suffix added to ensure it is unique. But distinguishing between Contacts-1 thru Contacts-8 can be confusing six months later. However, even with the best naming scheme applied, it can still get confusing. In extreme situations, a giant mass of interlocking table occurrences can even cause bad performance.

Defining Table Occurrence Groups

The *table occurrence group* method, illustrated in Figure 9-8, involves lining up a row of non-connected occurrences, one for each table in the database, which are regarded as *primary table occurrences*. These occurrences are *never* interconnected. Instead, any time two primary tables must be related, a duplicate of the primary instance of one table is created and attached as a secondary instance to another table. This is sometimes called an *anchor-buoy* model since the primary occurrences act as an anchor to which related table’s float off to the right.

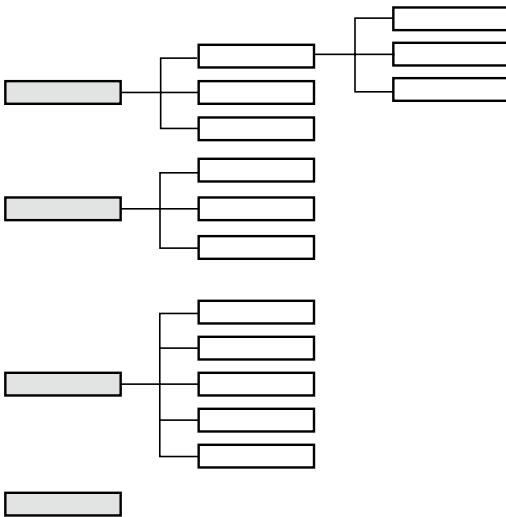


Figure 9-8. An illustration of the pattern of a generic table occurrence group organization

■ **Note** The phrase “table occurrence group” is somewhat awkwardly acronymized as TOG and referred to collectively as TOGs.

In this illustration, the primary tables are shaded gray and running in a column down the left side and secondary tables are clear and hanging off to the right in one or more hops. When using this method, all calculation fields and layouts should be *strictly limited* to using only a primary instance of a table to establish their context. From there, they can reach out and access related material from connected occurrences within the group.

Primary occurrences can be named the same as the actual table, *Company*, *Contact*, *Project*, etc. Secondary occurrences, which will be duplicates of these primary ones, should be named in a way that indicates their relational path back to the primary. This will cause them to sort together in pop-up menus and lists. For example, a secondary instance of the *Contact* table attached to a primary instance of a *Company* table could be called *Company | Contacts* indicating it is a secondary occurrence of the *Contact* table connected to the primary *Company* table occurrence.

Using this method will necessitate some redundancy in the relationship graph that might at first seem cumbersome and counterintuitive in making the graph more cluttered than it needs to be. However, as a database grows and becomes more complex, this method will avoid the performance issues that plague monolithic relationship design and, when combined with consistent naming rules for secondary occurrences, it can eliminate developer confusion when trying to select the appropriate table from a pop-up list in the development interface.

Introducing the Unifying Selector-Connector Concept

The previous statement about *never* connecting primary occurrences shouldn't be accepted as a dogmatic rule, superimposed upon any situation without an understanding of its purpose. Instead of viewing it as a rule to be blindly followed, consider it a principled technique that avoids certain problems inherent with a monolithic interconnection of table occurrences and generally a good rule to follow.

In recent years, an interesting new idea called *Selector-Connector* has been introduced by Todd Geist (Geist Interactive) and Jason Young (Seedcode). This idea grew out of many frustrations with the inefficiencies in the occurrence graph that were *created* by the strict group separation model. By keeping groups strictly separated, tables that provide centralized resources must be repeated on the graph, one for each group that requires access to the table. The Selector-Connector solution represents an evolutionary return to an *integrated* occurrence graph. The technique is specifically designed to carefully unite the performance benefits of group separation with the structural efficiencies of sharing resources in a monolithic structure.

The technique calls for the creation of a *Connector* table, containing only global fields, which is cross-connected to every primary interface occurrence with a Cartesian Join (discussed later in this chapter). This provides a conduit through which any interface context can access central repositories of information for uses such as the lists, navigation portals, accessing utility fields, and more. These central resources are then customized using interface controls such as portal filtering.

Selector-Connector doesn't remove *every* redundancy from the graph, only the unnecessary duplication required to share centralized resources.

Although it is a somewhat advanced idea, it is a rather straightforward approach that shows enormous promise for avoiding the problems of a monolithic model while maintaining efficiency within the occurrence graph. This technique makes the monolithic vs. grouping argument appear like a false alternative and it should probably have been the original solution: a method of solving performance issues without completely severing the possibility of efficiently sharing resources.

For more information about using this technique, search the Internet for "FileMaker Selector-Connector."

■ **Tip** Another method to provide access to central resources without extraneous duplication of occurrences or even the need for any connection is using the new *Card* option of the *New Window* script step to display records and selection controls from a context other than that of the current window (Chapter 35, "Createing a New Window").

Planning the Learn FileMaker Relational Object Model

As a practical example, apply these organizational methods to the Learn FileMaker example database. If we were to use the hub-and-spoke method to organize table occurrences, they would be interconnected as illustrated in Figure 9-9. A Contact and Project record can be assigned to a Company record and each occurrence would represent a layout: nice and simple.

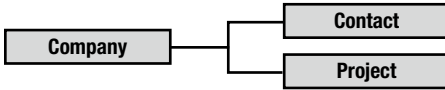


Figure 9-9. A simple hub-and-spoke configuration of the three example tables in the Learn FileMaker database

By contrast, the table occurrence group method would be organized based on the illustration in Figure 9-10. Each of the three primary occurrences on the left will be used as the context for calculations and a layout while calculations, portals, and scripts would draw values from a related secondary occurrence on the right. Even though this appears unnecessarily complex now, by adopting this method, it prepares for more complexity over time and serves as a good object lesson.

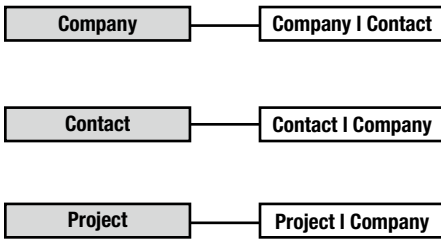


Figure 9-10. A more complex table occurrence group configuration of the three example tables in the Learn FileMaker database with duplicate, secondary occurrences attached prepares your database for growth

Managing Data Sources

A *data source* is a defined reference to an external database file that allows access to other FileMaker and ODBC databases from within a FileMaker database. Although FileMaker refers to all data sources as *external* data sources, the current FileMaker file, which is most certainly internal, is the default source.

If all table occurrences are contained in the one database file that you are working in, don't worry about defining data sources. The data source will always be that same file. However, when creating complex multi-file systems or integrating ODBC tables, data sources are important.

Introducing the Manage External Data Sources Dialog

Data sources are defined and managed in the Manage External Data Source dialog, shown in Figure 9-11. From this dialog, data sources can be created, edited, and deleted.

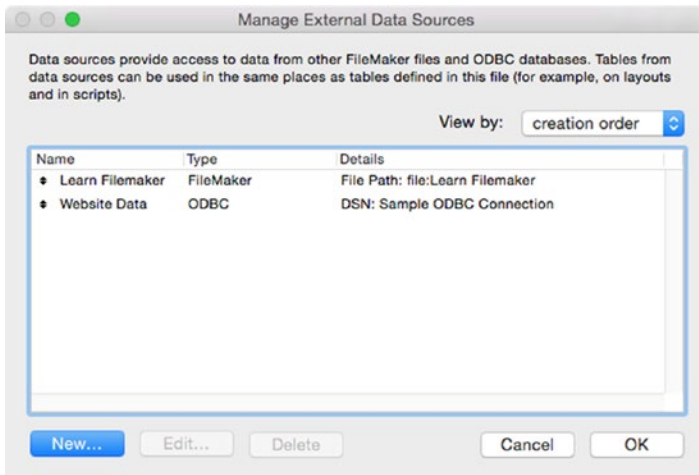


Figure 9-11. The *Manage External Data Sources* window

Once a database is open, this dialog can be accessed by any of the following methods:

- Select the **File > Manage > External Data Sources** menu item while in browse or layout mode.
- Run a script or click a button that uses the **Open Manage Data Sources** script step.
- Select one of three options at the bottom of the **Data Source** pop-up menu on the **Specify Table** dialog, shown in [Figure 9-12](#), which is used to add a table to the relationship graph.

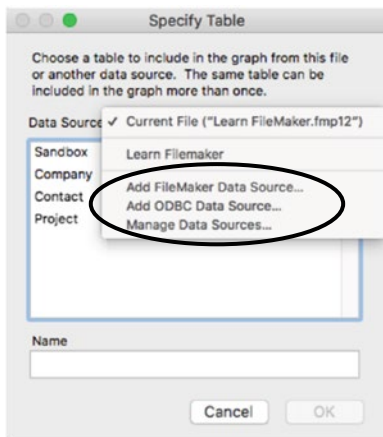


Figure 9-12. The *Data Source* pop-up menu in a *Specify Table* dialog

Exploring the Edit Data Source Dialog

Creating or editing a data source is performed in an `Edit Data Source` dialog, shown in Figure 9-13. This dialog can be opened by any of the following methods from the `Manage External Data Source` dialog:

- Click the `New` button to open a new empty dialog.
- Click the `Edit` button to open the dialog for the selected data source.
- Double-click on a data source in the list to open the dialog for that data source.

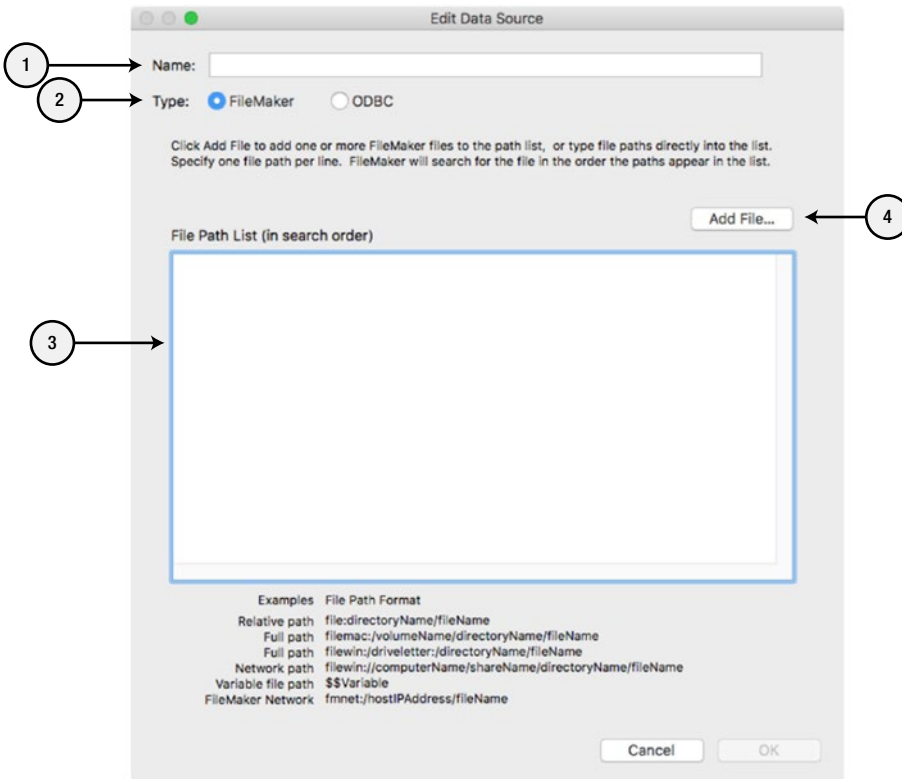


Figure 9-13. The `Edit Data Source` dialog

The controls on the `Edit Data Source` dialog are:

1. **Name** — In this text area, enter a name for the data source. This is how the source will appear in `Data Source` pop-up menus throughout the development interface and is *not* required to be the actual name of the external database.
2. **Type** — Select `FileMaker` or `ODBC` depending on the type of data source you are adding.

The lower part of the dialog changes depending on the `Type` selected. The remainder of this section assumes a selection of `FileMaker`. For details on adding an `ODBC` type data source, see Chapter 7.

3. **File Path List** — This text area is used to enter the path to the file containing the source table.
4. **Add File** — This button is used to select a file path through an Open File dialog, which can be useful to avoid errors one may encounter when manually entering a path.

■ **Tip** Beginning with FileMaker Pro 16, the file path entered can be a global variable allowing for dynamically changing data sources.

Introducing the Manage Database Dialog (Relationships)

Like tables and fields, table occurrences and relationships are created from within the Manage Database dialog. Once open, make sure the Relationships tab is selected, as shown in Figure 9-14.

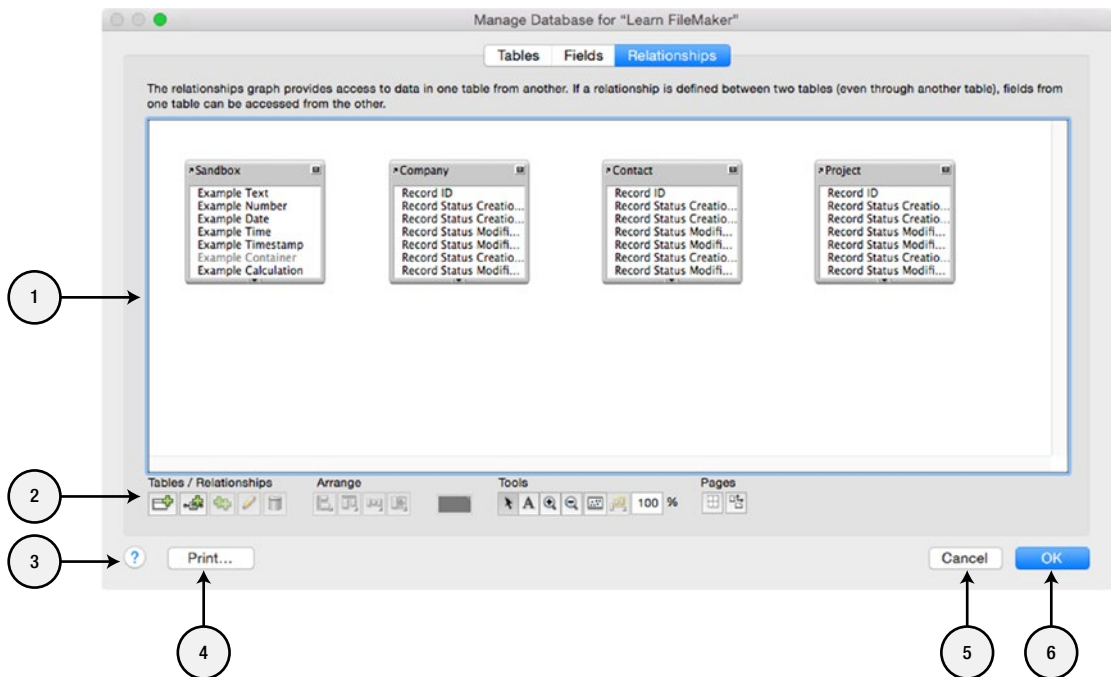


Figure 9-14. The Relationships tab of the Manage Database window

The controls on the Relationship tab are:

1. **Relationship Graph** — This expanding and scrollable graphical workspace is where you create and edit table occurrences and relationships. Each defined table will already be present in the graph with no relationships connecting them.

2. **Button Controls** — These 19 icon buttons perform various tasks within the graph area (see “Managing Relationships,” this chapter).
3. **Help** — This button opens the FileMaker help guide to the “Creating and Changing Relationships” page.
4. **Print** — This button will print the relationship graph.
5. **Cancel** — This button will close the window, without saving changes after displaying a warning dialog.
6. **OK** — This button will save changes and close the window.

Working with Table Occurrences

A *table occurrence* is graphical representation of a table displayed as a box containing a list of fields defined for the table. Figure 9-14 shows an example of the Learn FileMaker database relationship graph with four table occurrences, one for each table currently contained within the file. Before we discuss relating occurrences, let’s review the following topics for working with them:

- Selecting table occurrences
- Interacting with table occurrences
- Arranging table occurrences
- Viewing options
- Formatting table occurrences
- Editing table occurrences
- Adding table occurrences
- Deleting occurrences

Selecting Table Occurrences

When an occurrence is selected, it will be highlighted with a blue shaded border. You will then be able to modify its name and data source as well as manipulate its size, position, and formatting. The **Object Selector Cursor** tool with the icon of a standard cursor, shown in Figure 9-15, is used to make selections. This tool will be selected by default when the **Manage Database** window is open, but can be re-selected as needed.



Figure 9-15. *The Object Selector Cursor tool*

Once the **Object Selector Cursor** tool is chosen, occurrences can be selected using any of the following methods:

- To select a single occurrence, click on it.
- To select multiple non-grouped occurrences, hold the **Shift** key and click on each desired occurrence. A second click on a previously selected occurrence will deselect it.

- To select multiple occurrences clustered together in a group, click on the background in a space just outside the upper corner of the group, then hold and drag. A box with a thin black line will be traced out based on the direction and distance you drag the mouse. When this box encompasses all the desired occurrences, let go of the mouse button and everything within the box or touched by the box frame should be selected. Holding the Command (macOS) or Windows (Windows) key while dragging will require that an item be *completely* within the boundaries of the box before it will be selected.
- With one occurrence selected, you can use the options in the menu under the Select Tables tool, shown in Figure 9-16. The two options here are:
 - Select related tables 1-away — This option will select every table occurrence that is *directly related* to the selected table.
 - Select tables with the same source table — This option will select every other table occurrence that shares the *same source table* as the selected table.



Figure 9-16. The Select Tables tool

Interacting with Table Occurrences

Table Occurrences can be manipulated in a variety of ways. The box representing them can be moved, collapsed, resized, and scrolled at the points highlighted in Figure 9-17. A table occurrence box can be toggled between two states when not connected to another table or three when connected.

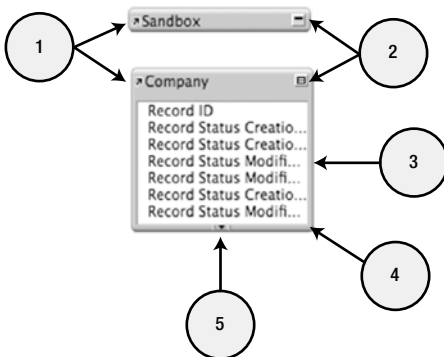


Figure 9-17. The manipulation points on a table occurrence

For each of these manipulations, the Object Selector Cursor tool must be selected:

1. The *heading* of a table occurrence is a thick gray bar at the top of the box in which the name of the table appears. Regardless of whether the occurrence is collapsed or expanded, clicking and holding on this region allows you to drag the box around within the relationship graph to reposition it.

2. On the right of the heading is a small expansion toggle icon. A single-click on this icon will toggle the state of the occurrence between one of three states, two if it isn't yet connected to any other occurrence. Without a connection, one click collapses the box completely and another click expands it fully. When connected to other tables, there is an intermediate state where the box collapses to show only the match fields used to form relationship(s) to other occurrences.
3. The thin gray bar on the sides and bottom of the box is used to resize the box in one direction. Click and drag on either side to expand or contract the width of the box or on the bottom to adjust the height.
4. Click and drag from the bottom corners to resize the box in two directions at once, adjusting the width and height to the amount based on the direction and duration of the drag motion.
5. Depending on the current scroll state of the field list within the box, a small arrow icon will appear at the top, bottom, or both. This is used to scroll up or down the list to locate a desired field. Individual clicks will move up or down one field at a time while a click and hold will continuously scroll the list until released.

To prepare our four Learn FileMaker occurrences, collapse them all and move them one above the other so they are roughly lined up in a vertical stack, as shown in Figure 9-18.



Figure 9-18. The four table occurrences collapsed and arranged in a stack

Arranging Table Occurrences

Keeping the relationship graph neat and orderly is important if you want to maintain an efficient development environment. The relationship graph provides four tools under the “Arrange” heading that are used to organize groups of selected occurrences relative to each other using terminology that is standard in most layout applications.

The Horizontal Alignment tool, shown in Figure 9-19, contains a pop-up menu with the following functions:

- Align Left Edges
- Align Centers
- Align Right Edges



Figure 9-19. *The Horizontal Alignment tool icon*

The Vertical Alignment tool, shown in Figure 9-20, contains a pop-up menu with the following functions:

- Align Top Edges
- Align Centers
- Align Bottom Edges



Figure 9-20. *The Vertical Alignment tool icon*

The Distribute tool, shown in Figure 9-21, contains a pop-up menu with the following functions:

- Distribute Horizontally
- Distribute Vertically



Figure 9-21. *The Distribute tool icon*

The Resize tool, shown in Figure 9-22, contains a pop-up menu with the following functions:

- Resize to Smallest Width
- Resize to Smallest Height
- Resize to Smallest Width & Height
- Resize to Largest Width
- Resize to Largest Height
- Resize to Smallest Width & Height



Figure 9-22. *The Resize tool icon*

Viewing Options

Under the “Tools” heading, to the right of the Object Selection Cursor and Text Note tools are four tools, shown in Figure 9-23 that allow manipulation of the graph view.

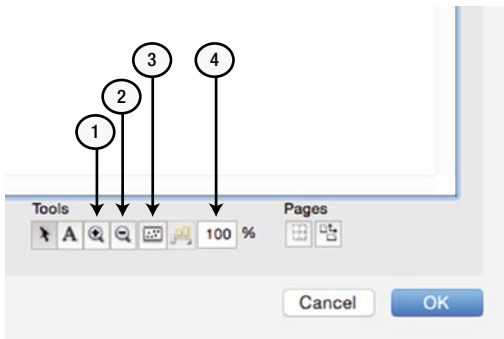


Figure 9-23. Viewing tools used to adjust the view of the relationship graph

Working from left to right, the viewing tools include:

1. **Zoom in** — Click this tool to activate the size increasing cursor, then click in the graph to increase the zoom by 25%.
2. **Zoom out** — Click this tool to activate the size decreasing cursor, then click in the graph to decrease the zoom by 25%.
3. **Adjust Magnification** — Click this tool to instantly adjust the magnification up or down to allow every table occurrence to fit in the view without scrolling.
4. This text area allows manual adjustment of the magnification to any percentage from 1 to 400.

Formatting Table Occurrences

There is only one formatting option for table occurrences and that is to assign a custom color. As the number of table occurrences increases, it can be enormously helpful to apply a color scheme to distinguish between groups of occurrences. These ideas may be helpful in deciding how to color code the relationship graph:

- Apply one color to primary table occurrences and another to secondary. Other colors might be used to denote the type of table or how it is used. For example, all join tables might be one color, all tables used in the interface another, while any for data relationships a third.
- Apply one color to each group of tables. For example, any occurrence of *Company*, *Contacts*, and other tables pertaining to the management of people might be one color while occurrences for *Project*, *Budget* or other similar tables another.

- Colors can indicate the type of operator used to connect a secondary occurrence to a primary one. For example, any occurrences that use an equal match operator could be a different color from those using other operators.
- Colors can be used to indicate the development status of groups within the graph.

To apply a color, select the occurrence(s), click the color tool, and choose the desired color from the palette, shown in Figure 9-24.

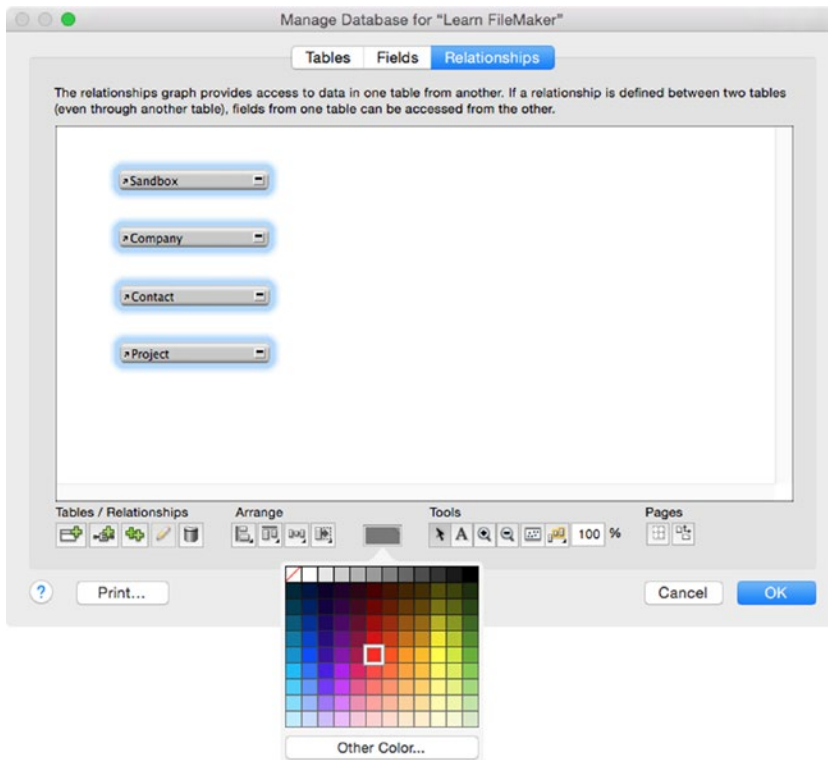


Figure 9-24. The color palette tool used to apply a color to an occurrence

Editing Table Occurrences

Table occurrences can be edited by opening the Specify Table dialog. This dialog is opened by either double-clicking an occurrence box in the graph or by selecting one and clicking the Edit Selection button icon, shown in Figure 9-25.



Figure 9-25. The Edit Selection button icon

Introducing the Specify Table Dialog

From the Specify Table dialog, shown in Figure 9-26, the data source, assigned table, and occurrence name can all be modified as needed.

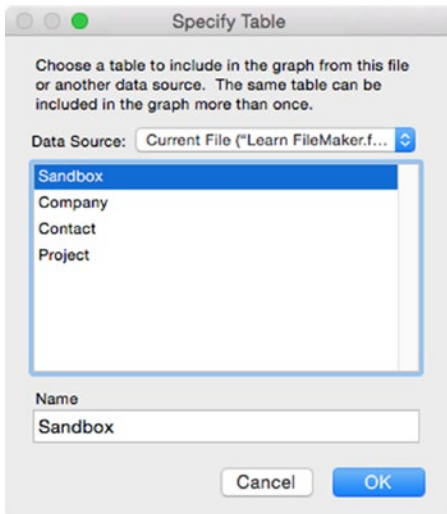


Figure 9-26. The Specify Table dialog

Adding Table Occurrences

Table occurrences can be added to the graph for any table that exists in a data source defined in the current database file. This can be done one of two ways:

- Creating a new table occurrence
- Duplicating an existing table occurrence

Creating a New Occurrence

To create a new table occurrence, follow these steps:

1. Click the Add Table button icon, shown in Figure 9-27. This will open the Specify Table dialog.



Figure 9-27. The Add Table button icon

2. Choose the data source containing the table you want to add, and then click to highlight the table.

3. The name field will automatically be filled with a suggested name. This will be the name of the table with a numeric suffix as needed to ensure a unique table name. Optionally, type a new name into this field to clarify the specific context and usage of the new instance of the table in the graph.
4. Click the OK button to complete the process. The new table occurrence will appear in the relationship graph for the selected table with the specified name.
5. Resize and position the new occurrences as needed.

Duplicating an Existing Occurrence

There are advantages to duplicating existing occurrences instead of creating new ones. First, if an occurrence exists for the same table you want to add, duplicating it saves you the step of assigning the table manually. However, even if the table doesn't have a current representation in the graph, duplicating an existing table's occurrence and assigning to it a different table will still save time sizing and formatting the new occurrence as it will retain these properties from the original. This is helpful in maintaining a neat and visually pleasing relationship graph.

To duplicate the selected occurrence(s), click the Duplicate Table button icon, shown in Figure 9-28.



Figure 9-28. *The Duplicate Table button icon*

Deleting Occurrences

Deleting an occurrence will *not* delete the actual table definition, but it will cause several actions throughout a database, including:

- The occurrence itself will be deleted.
- Any layouts that used the occurrence as its relational context will have an <unknown> table assignment, and will need to be reassigned an existing occurrence or deleted.
- Any field on a layout that is based on the context of the occurrence will become a <missing field>.
- Any portals displaying records from the occurrence will be broken.
- Any script steps that relied on the context of the occurrence will be broken.

To delete one or more selected occurrences, type Delete on the keyboard or click the Delete Table tool shown in Figure 9-29.



Figure 9-29. *The Delete Selection button icon*

Before deleting the selected occurrence(s), FileMaker will present a warning dialog confirming the request. If you mistakenly confirm the deletion, click the **Cancel** button at the bottom of the Manage Database dialog to discard all changes, including the deletion.

Printing the Relationship Graph

The relationship graph includes several functions that control printing of table occurrences and the relationships that connect them. Instead of using the standard print menus, printing the graph is controlled by tools in the bar.

Click the **Page Setup** tool, shown in Figure 9-30, to open a standard dialog for selecting printer type, paper size, orientation, and scale.



Figure 9-30. The *Page Setup* button icon

Click the **Page Break** tool, shown in Figure 9-31, to toggle the visibility of page breaks in the graph area.



Figure 9-31. The *Page Breaks* button icon

Once the page setup is properly configured for the desired paper and printer, click the **Print** button at the bottom of the dialog to open a standard Print dialog.

Building Relationships

Once table occurrences are added and neatly organized within the graph, it is time to begin *connecting* them to form relationships.

Adding Relationships

There are two methods available that connect occurrences and form a relationship: pulling a connection between two occurrences or using the **Add Relationship** button.

Pulling a Connection Between Occurrences

The most intuitive way of creating a connection between two occurrences is to use the cursor to drag a connection from a field in one occurrence to a field in another, thereby establishing a relationship based on those two match fields.

To do this, follow these steps:

1. Adjust the view of the graph so that the two occurrences are viewable, preferably side-by-side although this is not required.
2. Click the icon in the upper-right corner of both occurrences until they are fully expanded.

3. Scroll each list of fields so that the match fields for the relationship are visible.
4. Click and hold on a match field from one occurrence. As you begin to drag the cursor toward the other occurrence, a line will appear connected to the first match field, shown in Figure 9-32.

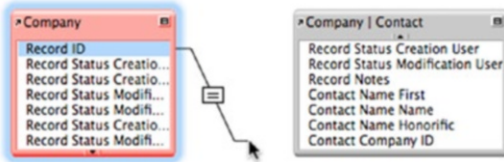


Figure 9-32. The process of dragging one match field toward the other with a relationship connection in tow

5. Once the cursor is on top of the desired match field for the other occurrence, release it, and the line connecting the two will be fully established.

Using the Add Relationship Button

The other method to creating a relationship is using the Add Relationship button in the toolbar, shown in Figure 9-33.



Figure 9-33. The Add Relationship button icon

Clicking this button will immediately open an empty Edit Relationship dialog with both tables set to <unknown>. From there, you can select the two table occurrences from the pop-up menus and selected the match fields.

■ **Tip** Consider using a combination of these two techniques to improve your efficiency. To avoid having to scroll the occurrence boxes or enlarge them to locate the desired match fields, drag to connect *any* two fields that are currently visible to quickly establish the connection. Then, open the Edit Relationship dialog to select other fields from the larger and more easily scrollable lists. Since you will probably open the edit dialog anyway to configure other settings, this method offers the fastest and still rather intuitive method for creating a new relationship.

Manipulating Relationships

Once a relationship is established, it is represented as a line connecting the two occurrences with a box in the middle, as shown in Figure 9-34.

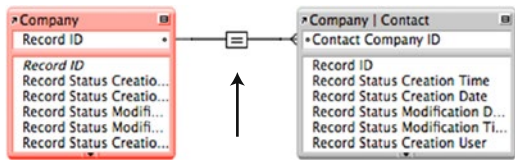


Figure 9-34. The relationship appears as a line with a small selector box in the middle

The box will display the operator used to form a match between the two fields, with the default being an equal sign as shown.

At each end of the line is an indication of the type of relationship established, shown enlarged in Figure 9-35. A straight line, like that on the left, indicates a “one” connection and the “crow’s feet” on the right side indicates a “many.” Therefore, we have a one-to-many relationship between the Company and Contact table occurrences. FileMaker determines this by looking *only* at the auto-enter settings for the fields. Since the Record ID field in Company is configured to auto-enter a unique serial id for each record, FileMaker assumes that this is a “one.” On the other side, it doesn’t see this setting so it assumes that it is a “many.” Although these symbols are accurate in the case of our example, this is the extent of the logic used to determine the type of relationship. FileMaker does *not* consider your layout rules, field content, or script behaviors, which may enforce a different type of relationship.

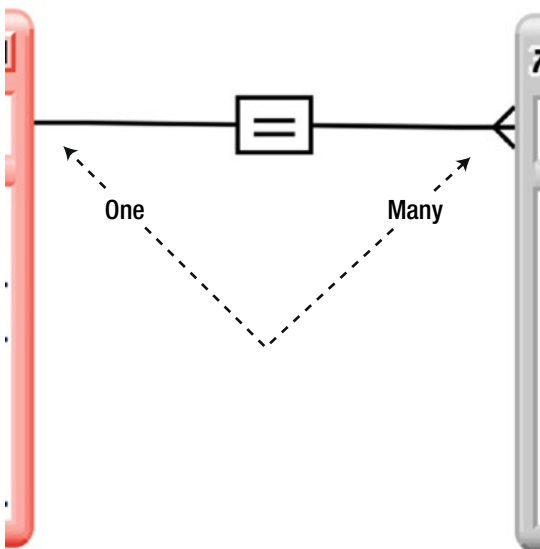


Figure 9-35. End-of-line symbols indicate the type of connection for that side of the relationship

When moving occurrence boxes, the relationship line will stay connected on both sides, regardless of one box's position relative to the other box. The single, straight line will split into three segments and change angles so each remains straight and maintains the connection, as shown in Figure 9-36. At a certain point, the line will snap from one side of the occurrence box to the other to maintain a connection.

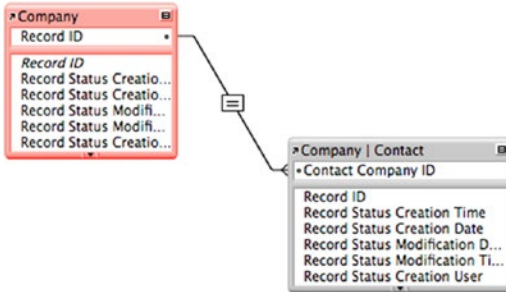


Figure 9-36. As a box moves, the line splits into three straight segments that angle in relation to each other

To select a relationship for editing, click on the small box in the middle of the relationship line. When selected, the line will darken and the box will turn blue. In the toolbar, the **Edit Selection** and **Delete Selection** tools will become enabled.

To open the **Edit Relationship** dialog, either double-click on the box or click the **Edit Selection** tool, shown in Figure 9-25. To delete the relationship between the occurrences, either type **Delete** or click the **Delete Selection** tool, shown previously in Figure 9-29. Deleting a relationship will *not* affect the two occurrences.

■ **Caution** Typing **Delete** or clicking the **Delete Selection** button will delete any a selection of table occurrence(s), relationship or note(s). Although a warning dialog is presented prior to deleting a table occurrence, it is still a good idea to be aware of what is selected before using these.

Introducing the Edit Relationship Dialog

Relationships are edited in the **Edit Relationship** dialog, shown in Figure 9-37, which can be opened by either double-clicking on the relationship line's box or by selecting the relationship and clicking the **Edit Selection** tool.

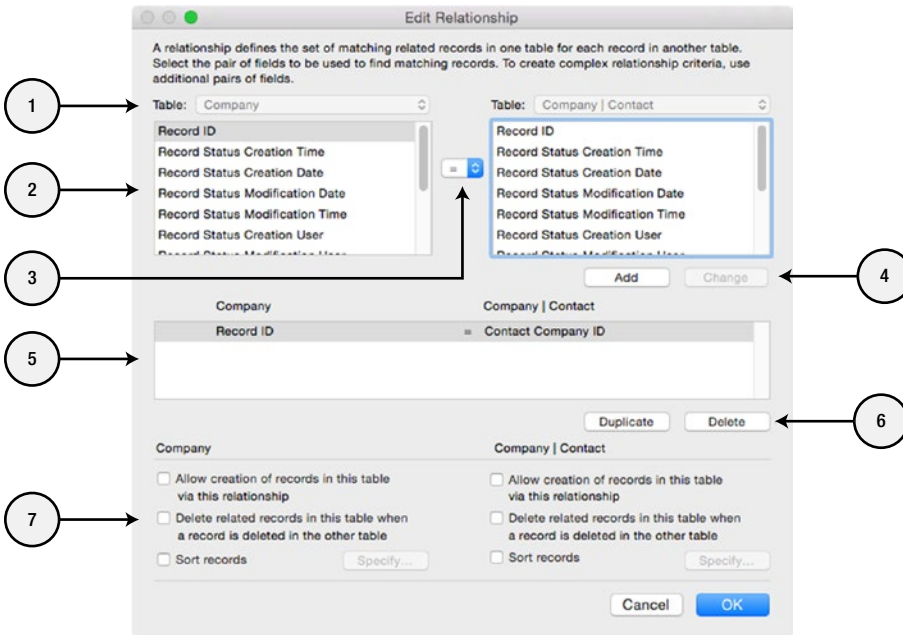


Figure 9-37. The *Edit Relationship* dialog

The following controls are available in this dialog:

1. **Table Occurrence** — The pop-up menus on either side of the dialog contain a list of every table occurrence in the file. These are only enabled when creating a new relationship. When editing an existing relationship, they are disabled and non-editable since the table for each occurrence is already set.
2. **Selected Match Field** — Below each occurrence menu is a list of the fields contained within its table. This is where match fields are selected before being added or changed in the list below.
3. **Comparative Operator** — This pop-up menu contains a list of available operators that determine how the two selected match fields will be compared (described later in this chapter).
4. **Selected Match Field Buttons** — These two buttons are used to Add a match field combination from the lists above into the list below or to Change the selected combination below to the match field combination above.
5. **Link Field(s)** — This area contains a list of one or more match field combinations that make up the total relationship criteria.
6. **Link Field Buttons** — These two buttons are used to Duplicate or Delete the selected Link Fields above.
7. **Relationship Options** — Two identical sets of three check boxes allow specification of certain behaviors for each side of the relationship (described later in this chapter).

Comparative Operators

A *comparative operator* is used to compare two match fields, one on each side of a relationship, to determine the type of equation used to detect a match. The available operators, shown in Table 9-1, are standard operators familiar to any student of math.

Table 9-1. A description of the comparative operators available for forming relationships

Operator	Description
=	The values in both match fields must be equal to each another.
≠	The values in both match fields must be not equal.
<	The values in the left match field must be less than the value in the right match field.
≤	The values in the left match field must be less than or equal to the value in the right match field.
>	The values in the left match field must be greater than the value in the right match field.
≥	The values in the left match field must be greater than or equal to the value in the right match field.
X	All records on the left side of the relationship will be matched to all records on the right side, regardless of the actual values contained in the selected match fields. This is often referred to as a <i>Cartesian Product</i> , <i>Cartesian Join</i> or <i>Cross Join</i> , which means a join between two tables that is unrestricted by any criteria since every record will be a match no matter the criteria. Some prefer to think of the “x” as a wheel of an old stage coach or cart where the spokes branch out in many directions, relating everything to the hub in the center.

Relationship Options

The three options available at the bottom of the Edit Relationship dialog, shown in Figure 9-38, control basic functionality for each side of the relationship.

The image shows a screenshot of the 'Edit Relationship' dialog box. It is divided into two columns: 'Company' on the left and 'Company | Contact' on the right. Each column contains three checkboxes and a 'Specify...' button. The checkboxes are: 'Allow creation of records in this table via this relationship', 'Delete related records in this table when a record is deleted in the other table', and 'Sort records'. The 'Specify...' buttons are located to the right of the 'Sort records' checkboxes.

Figure 9-38. The options at the bottom of the Edit Relationship dialog

Allowing Dynamic Creation of Related Records

When selected, the first check box on either side will allow a user to click into a field from one side of a relationship (Contact) on a layout context from the other side (Company) and enter data even though the field is not yet associated with a related record. When the user types, the related record is instantly created with all required match field values in place to establish the relationship to the record being edited.

This is commonly used in portals. It has the effect of displaying a blank, empty row at the bottom of the portal (Chapter 30). Users can create a new related record by typing in any field in that empty portal row. Without this feature, you must create a script assigned to a button that creates the new, empty related record before a user can begin typing.

Deleting Related Records

The second check box specifies that records on one side should be automatically deleted when the related record on the other side is deleted by a user or script. This is especially useful when considering one side of a relationship a “parent,” hierarchically above the other “child.” For example, when a parent Company record is deleted, any related Contact records will be deleted as well if this check box is selected under Company | Contact in Figure 9-38.

Sorting Related Records

The final box allows a *data level* sort of records when viewed through the relationship. A check in the box will bring up a Sort dialog in which you can specify a sort order that will be automatically applied as the default sort order when displaying records for that relationship. Alternatively, or to override this setting, a portal can be assigned a sort order for related records.

Adding Notes to the Graph

A relationship *note* is a free floating, colored box that can be added in the graph, as shown in Figure 9-39. These can be used to add titles to divide groups of occurrences, store developer notes, and other information.

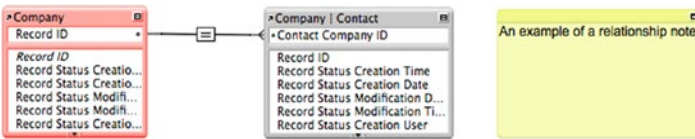


Figure 9-39. An example of a relationship note

A new note can be created following these steps:

1. Select the Text tool, shown in Figure 9-40.

A

Figure 9-40. The Add Note toolbar icon

2. Using the mouse, click on the background of the graph, then hold and drag. A note will appear, steadily growing based on the direction and distance you drag the mouse. When it is large enough, let go of the mouse button.

■ **Tip** A note can be quickly created by clicking and dragging on the background while holding Option (macOS) or Alt (Windows).

Once created the note will appear on the graph and an Edit Note dialog will automatically open. The dialog, shown in Figure 9-41, allows entry and editing of the note text as well as specification of the font, size, text color, and background color of the note.

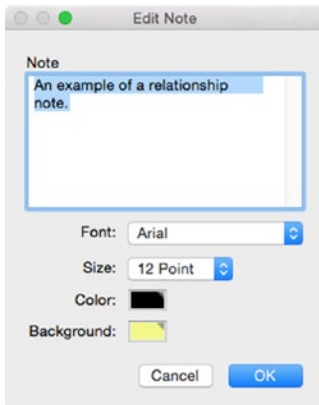


Figure 9-41. The *Edit Note* dialog

When finished editing, click the OK button to save changes. Clicking the Cancel button will cause one of two results depending on the status of the note:

- If the note has not yet been saved, clicking Cancel will close the dialog and delete the new note.
- If the note has previously been saved, clicking Cancel will close the dialog without saving any changes to the text or settings, leaving the note unchanged by any actions taken in the dialog.

Once saved, a note can be moved, resized, or minimized using the cursor as you would a table occurrence or window. Double-clicking on the note or selecting it and clicking the Edit Selection tool will reopen the Edit Note dialog. To delete a note, select it and type the Delete key or click the Delete Selection icon in the toolbar.

Finalizing the Learn FileMaker Relational Model

Finally, we can implement the relationship model plan shown previously in Figure 9-10 into the Learn FileMaker database, as shown in Figure 9-42.

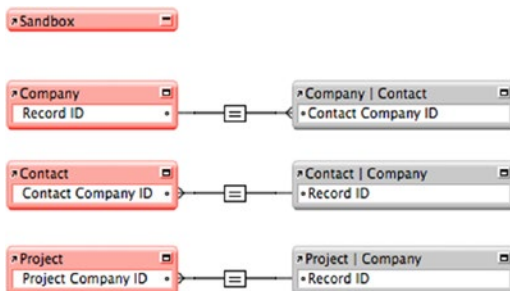


Figure 9-42. The *Learn FileMaker* relational model

These basic connections will allow each table to connect to another table as follows:

- **Company** — Each record for a company will be able to display a portal listing every related “child” contact record: the people who work at the company.
- **Contact** — A record for a contact will be able to connect to its “parent” company and display the name of the company on the layout.
- **Project** — A record for a project will be able to connect to its “parent” company record as well.

Summary

In this chapter, we discussed the basics of table occurrences and connecting them to form relationships.

CHAPTER 10



Managing Containers

A *container field* is used to store any kind of document. Traditionally, and in some circles to this day, the content of a container is referred to as a *binary large object* or *basic large object*, both commonly expressed as the acronym “BLOB,” which is a collection of binary data representing image, audio, or other multimedia files and stored as a single entry in a database.

The options for and behaviors of a container field depend on several factors. Various settings influence how users or scripts insert a file into a container, how that material is stored, and how it is displayed in a field on a layout. These factors often intersect in a somewhat confusing mixture, and your choices may be influenced by the competing needs of functionality, user preference, and installation type – all juggled while trying to maintain optimum performance by maintaining a lean file size.

This chapter provides an overview of the options for defining and managing containers, covering the following topics:

- Inserting files into containers
- Understanding container storage options
- Using managed external storage
- Extracting files from containers

For more information about the options for containers on a *layout*, see Part 4.

Inserting Files into Containers

When a container field is visible and accessible to a user on a layout, a document file can be inserted using the following methods:

- Using an Insert Menu option
- Copying and pasting
- Dragging and dropping

Depending on the container management settings for a field, each method has different options or limitations, which can impact how the document is inserted and the size of the database file.

Insert Menu

The Insert menu contains four options for inserting a file into a container field, shown in Figure 10-1. Each of these is also available in the field's contextual menu.

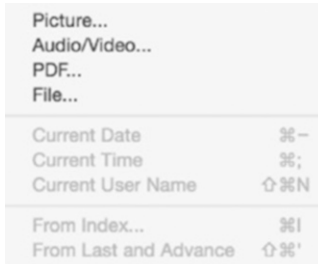


Figure 10-1. The Insert menu options for placing content into a container field

Choosing any of these options will open a Choose File dialog, optimized for the type of file corresponding to the menu item. The four options are:

- **Insert Picture** — This option allows the selection and insertion of an image file into the field, which will be displayed as a picture rather than a file.
- **Insert Audio/Video** — This option allows the selection and insertion of an audio or video file and is only visible when the field's layout settings are configured to allow for **Interactive content**. This allows the content to be played directly from within the field without having to open it in another application. Audio and video files can be placed into a field using the **Insert File** option or other methods but will not be interactive.
- **Insert PDF** — This option allows the selection and insertion of a PDF file and is visible only when the field's layout settings are configured to allow for **Interactive content**. This allows pages of the PDF document to be viewed directly from within the field without having to open it in another application. PDF files can be inserted using the **Insert File** or **Insert Picture** menus as well as other methods but will not be interactive.
- **Insert File** — This option will inset any type of file into a field. The file will appear as it does in a directory folder, represented by an icon with no option to access or view its content.

Drag and Drop

Files can be dragged and dropped into a container field that is visible and configured to be accessible on a layout. When dropping a file, FileMaker will automatically use the appropriate storage method based on the file type and the configuration of the field.

For example, if you drop an image it will be inserted and displayed as if you selected the Insert Picture menu item. If you drop a PDF or audio file into a field that has not been configured for Interactive content, the file will be placed into the field and displayed as a preview of the first page of the PDF or as a file icon. However, if the field is configured for interactivity, the file will be inserted as if you chose the menu corresponding to the file type.

Using this method there is no way to manually specify external storage except by using FileMaker-managed external storage, discussed later in this chapter.

Copy and Paste

Files can be placed in a container field using the Copy and Paste commands when a container field is visible and accessible on a layout. The clipboard can contain an actual file copied from a folder or the content of an image copied from a picture file. The placement of this material into the field is identical to that of dragging and dropping.

Using this method there is no way to manually specify external storage unless using FileMaker-managed external storage, and unless pasting the text required to establish an external reference instead of a file.

Understanding Container Storage Options

A container field can store files in one of two ways:

- Storing the actual file *inside* the database.
- Storing a *reference* to the file located *outside* the database.

Storing the Actual File

By default, newly defined fields are configured to store files within the actual database file. When a user or script inserts a document into a container field using any method, a copy of that document will be stored inside the field as part of the database file, increasing its overall file size. This isn't a major problem, especially when a database has few records or the embedded files are small. In time however, it may become an issue.

A FileMaker database *can* be up to 8 terabytes in size. However, as the size of the file grows, performance can become degraded, especially when many users share access to a database over a network. One of the goals of an efficient technical design is to maintain the smallest possible file size, so this method of storage should be limited in use to situations where a file must be completely self-contained and portable.

Storing a Reference to a File

The best way to maintain a lean database and help support optimal performance is to *not* store container documents *inside* the database. Instead, store only a reference to the file as a text string like this:

```
size:731,960
image:../../../../../../../../Desktop/Flower Picture.jpg
imagemac:/Macintosh HD/Users/john_smith/Desktop/ Flower Picture.jpg
```

The string above contains a typical reference to an image file but may vary slightly depending on computer platform and whether the database is hosted locally or accessed from a remote server. It contains three pieces of information:

- If the file is an image, the dimensions of the original file.
- A path to the file relative to the database.
- An absolute path to the database.

There are two ways to store references; one chosen manually during file insertion and the other by defining a property of the field itself.

Storing a Reference to the File in Any Directory

When a document is selected for insertion into a container, whether by a script or a person, there is an option to insert a reference to the file instead of copying the actual file. This choice is made in the Insert dialog by selecting the *Store only a reference to the file* checkbox, shown in Figure 10-2.

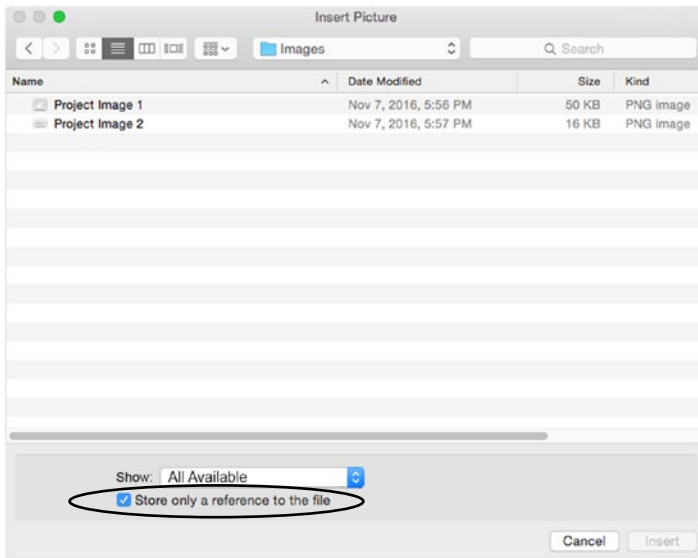


Figure 10-2. The check box option to store a reference instead of copying the entire file

When this option is selected, the container field will *display* the document as if it is stored within the database, depending on the type of file and layout settings of the field. However, it will have stored only a small amount of text within the field that points to the file in whatever directory it is stored.

This helps to maintain a lean database since the textual reference to the file is minuscule in size. However, it has some serious drawbacks that need to be considered. Most notably, if the document being referenced is moved to a new folder directory, the field will no longer be able to locate it and will display a text message indicating that the file cannot be found. Also, if one user of a networked database inserts a reference to a document that is stored locally on their computer or on a server location to which other users don't have access, other users will see the same error.

When references point to files in a static server location that every user of the database has equal access to, this method will work fine.

Storing a Reference to the File in a Managed Directory

The best choice for container field storage, especially for networked databases, is to use *FileMaker-managed external storage*. With this feature the database *automatically* makes an external copy of the inserted document into a managed folder and inserts a reference into the field.

It doesn't matter if other users have access to the actual folder directory that contains the files, although there are options to allow for that. Depending on configuration, FileMaker acts as a broker between the end user and the stored material, allowing documents to remain safely in place while being displayed for or downloaded by users in a "pass through" manner. There is an choice for each field to store files securely encrypted or open and accessible, which provides a nice degree of flexibility depending on your needs.

This feature is configured by defining a storage location in the database and configuring each container field to use that location.

Using Managed External Storage

To use FileMaker-managed external storage for container fields, there are several steps that need to be taken. These include defining one or more base directories and assigning these to container fields.

Defining Base Directories

A *base directory* is a developer-defined path to a folder directory that acts as the root location into which one or more container fields can store documents. The folder defined is fully managed by FileMaker, which will automatically create the folder if it doesn't yet exist and then manage the internal directory structure of subfolders and files depending on the settings defined for the base directory.

Every new database file contains one default directory defined with the same name as the database and will automatically have a formula prefix of [database location]. In addition to this default location, any number of alternate directories can be defined.

■ **Note** If a database file is renamed, the default base directory will not automatically update the folder name to match. This can be done manually.

Exploring the Manage Containers Dialog

The Manage Containers dialog is where you define the folder locations into which FileMaker will store and manage documents. To open the dialog, select the File > Manage > Containers menu item. This dialog has two tabs: Storage and Thumbnails.

Exploring the Storage Tab

The Storage tab of the Manage Containers dialog, shown in Figure 10-3, contains a list of defined base directories.

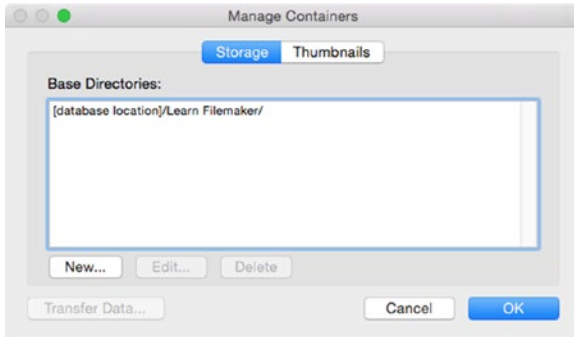


Figure 10-3. The Storage tab of the Manage Containers dialog

From this tab, you can create, edit, and delete base directories. You can initiate a transfer of documents using the Transfer Data button, which becomes highlighted after a field’s assigned base directory is changed and documents are detected in the old location (see “Changing a Field Container Settings” later in this chapter).

Creating a New Base Directory

To create a new base directory, click the New button on the Storage tab of the Manage Containers dialog. This will open an Edit Base Directory dialog, shown in Figure 10-4.

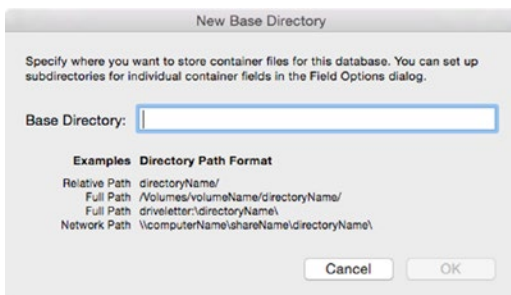


Figure 10-4. The Edit Base Directory tab dialog

This window contains a single text area into which you can either type a directory path or drop a folder to automatically insert its path. A path must be formatted as shown in the examples (Chapter 6, “Specifying File Paths”).

When finished, click OK to save the path or Cancel to close without saving or creating the new base directory.

Editing a Base Directory

Once a base directory is created, double-clicking on it in the list or selecting it and clicking the **Edit** button will reopen the dialog. However, if the selected directory is already in use by a field and has managed files stored within it, an error dialog will appear stating this fact and denying you the ability to open the directory and modify the path. The only way around this is to create a new directory, point any field using the old one to the new one; and transfer the existing container documents into the new folder, which will release the editing restriction on the old.

Deleting a Base Directory

When a base directory is no longer used, its reference can be deleted from the list by selecting it and clicking the **Delete** button. The same restrictions imposed when attempting to edit a directory apply when deleting. The directory can't be in use or contain any referenced files. Also, FileMaker requires there to be at least one base directory defined, so a delete request will be rejected if there is only one in the list.

■ **Note** FileMaker will not delete the actual folder, just the reference to it in the base directory list.

Exploring the Thumbnails Tab

The Thumbnails tab of the Manage Containers dialog, shown in Figure 10-4, contains the option to have FileMaker automatically generate thumbnails for larger images to speed up network transfers. Since it not usually necessary to display an image in full resolution when the field it is stored in is displayed on a layout, using this option can greatly improve the performance of window refreshes.



Figure 10-5. The Thumbnails tab of the Manage Containers dialog

Once the **Generate and store thumbnails for images** check box is selected, FileMaker will automatically begin generating and displaying a thumbnail for images when a layout attempts to display them. There are two storage options:

- **Permanent storage** — This option will cache thumbnails both on-disk and in-memory, with the on-disk portion retained when the database is closed.
- **Temporary storage** — This option will cache thumbnails in-memory only. When the database is closed, the cache is discarded.

Defining a Field to Store Container Data Externally

Once the appropriate base directory is created, it can be assigned to container fields.

Exploring the Field Options for Containers

A container field's external storage settings are configured through the Storage tab of the Options for Field dialog, shown in Figure 10-6. To open this dialog, follow these steps:

1. Select the File > Manage > Database menu item.
2. Click on the Fields tab of the Manage Database dialog.
3. In the Table pop-up menu, choose the table in which the container field exists. For example, in the Learn Filemaker database, choose the Sandbox table.
4. Double-click on a container field to open the Options for Field dialog.
5. Click on the Storage tab.

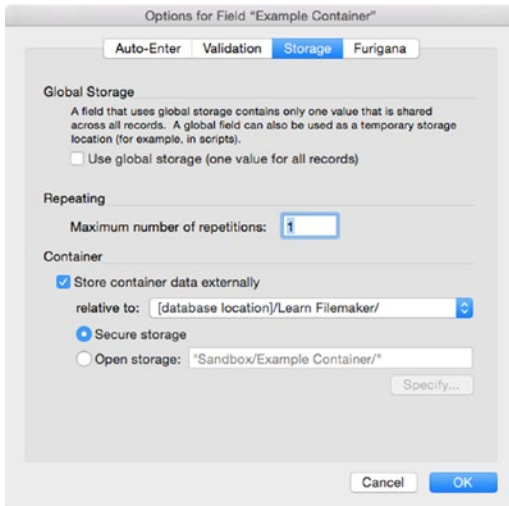


Figure 10-6. The Options for Field dialog for a container field

Select the Store container data externally checkbox to activate managed external storage.

The relative to pop-up menu contains a list of all the directories defined in the Manage Containers dialog with the first in the list selected by default. Select one to specify which base directory will be used to store documents for the container field.

Once the directory is selected, there is a choice between two types of storage methodology: secure or open.

Using Secure Storage

This option will encrypt the document data and automatically distribute the files randomly across subdirectories that are automatically created in a “Secure” subdirectory within the base directory.

Using Open Storage

This option will keep document data in their original file format and store them in the developer-specified subdirectory indicated in the adjacent text area.

■ **Caution** When more than one field uses the same base directory, each should be assigned a unique subdirectory.

While Secure storage automatically handles duplicate file names, with open storage your choice of a subdirectory may need to take this into consideration if similarly named files will be placed into a field for different records. To protect against this, click the adjacent Specify button and define a subdirectory with a dynamic calculation formula that includes a unique field, such as a serial id like this:

```
"Sandbox/Example Container/" & Record ID
```

This will ensure that each file placed in that field for any record is stored in its own subdirectory and that no naming conflicts will ever occur.

Changing a Field Container Settings

When a container field's settings are modified, FileMaker will automatically recognize that a transfer of data is required. This applies to any change, including:

- Switching from unmanaged internal storage to managed external storage.
- Switching from managed external storage to unmanaged internal storage.
- Changing the field's assigned base directory.

Once these changes are made and the field definition is saved, FileMaker will present a Container Data Transfer dialog, like the one shown in Figure 10-7.

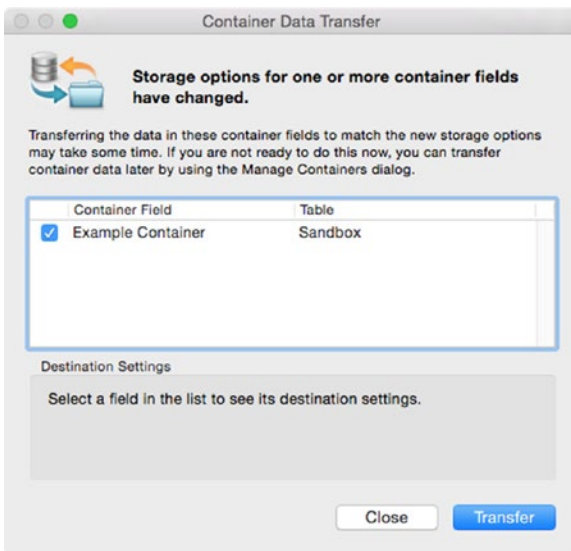


Figure 10-7. The Container Data Transfer dialog informs you that documents need to be transferred to, from, or between managed directories

This dialog lists all the container fields that have been changed. Selecting a field from the list will display that field's new destination settings, indicating the type of storage and new base directory. To immediately transfer the new documents to their new base directory, click **Transfer**. If you close instead, these changes will be retained in a queue and can be transferred later with the **Manage Containers** dialog **Transfer Data** button, which becomes enabled any time there are pending transfers in the queue.

Extracting Files from Containers

When a user has access to a container field on a layout, they can extract a file contained within and save it into a folder directory. This is done by one of the following methods:

- Select the **Edit > Export Field Contents** menu item.
- Click the field with the **Control** key down and choose **Export Field Contents** in the contextual menu.
- Run a script or click a button that uses the **Export Field Contents** script step.

Once selected, an **Export Field to File** dialog will open, allowing the user to choose a folder in which to save the document. The file name will default to the name of the document in the container but can be renamed in this dialog. From the dialog, the user can choose to automatically open the file and/or create an email with it as an attachment after the save is complete.

Summary

In this chapter, we discussed the various methods of managing containers and the ramifications of different document storage methods.

CHAPTER 11



Managing Value Lists

A *value list* is a carriage-return delimited group of unique text values that can be assigned to layout objects to give users a fast and reliable method of entering standard or frequently used information into a field in a consistent and error-free manner. With only one exception, value lists are essentially *context insensitive* so a list can be generated from any table occurrence regardless of the current layout context.

Lists can be defined to contain static, predefined values or values dynamically compiled from the contents of a field gathered directly from a table or through a relationship.

This chapter covers the following topics:

- Using value lists
- Exploring the value list interface
- Creating value lists

Using Value Lists

Values lists can be assigned to text fields as one of four different control styles from within the Layout Inspector. These are:

- Pop-up menu
- Drop-down list
- Radio buttons
- Check boxes

For each of these, the items in a value list become easily selectable options that can be entered in a field with a click. However, this chapter focuses on creating and managing value list definitions. For more information on how to assign and configure these on a layout, see [Chapter 30](#).

Exploring the Value List Interface

Value lists are defined, managed, and edited from two primary interface dialogs.

Introducing the Value Lists Dialog

Value lists are defined from the Manage Value Lists dialog, shown in Figure 11-1. Once a database is open, this dialog can be opened by any of the following methods:

- In browse or layout mode, select the File > Manage > Value Lists menu item.
- Click the pencil icon button in the Layout Inspector when a value list control style is applied to the selected field (Chapter 29).
- Run a script that uses the Open Manage Value Lists script step.

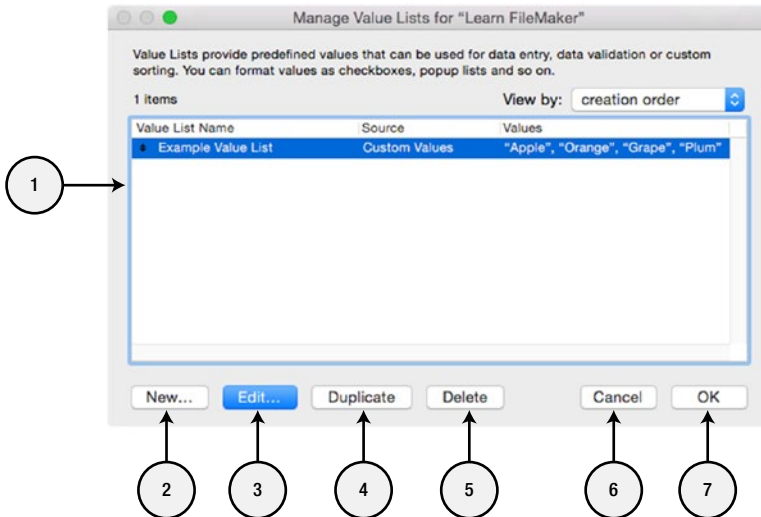


Figure 11-1. The Manage Value Lists dialog

The controls on the Manage Value Lists dialog are:

1. Value Lists — displays all the value lists that are defined within the file. The list can be sorted by either clicking on a column heading or choosing an option from the View by pop-up menu in the upper-right corner.
2. New — creates a new value list and opens it in the Edit Value list dialog.
3. Edit — opens the selected value list in the Edit Value list dialog.
4. Duplicate — creates a duplicate of the selected value list.
5. Delete — deletes the selected value list.

6. **Cancel** — closes the dialog without saving changes after presenting a warning.
7. **OK** — saves changes and closes the dialog.

■ **Tip** As of FileMaker Pro 16, you can copy and paste value lists within a file or between two files.

Introducing the Editing Value List Dialog

After creating a new value list or when editing an existing one, the **Edit Value List** dialog, shown in Figure 11-2, will open. This dialog is used to define the values that make up the list.

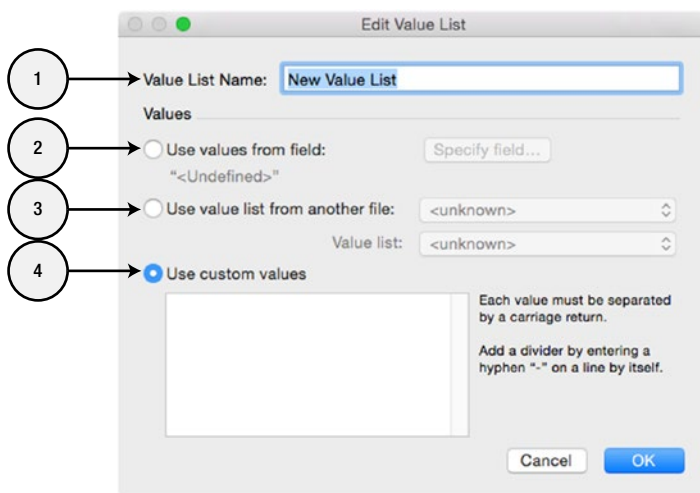


Figure 11-2. *The Edit Value List dialog*

The controls on the **Edit Value List** dialog are:

1. **Value List Name** — contains the developer-assigned name of the value list.
2. **Use values from field** — causes the value list to be generated automatically from values contained in a field selected in a **Specify Field** dialog.
3. **Use value list from another file** — activates the adjacent pop-up menus that allow a selection of a value list from an external FileMaker database file which will provide the values for the value list being edited in the current file.
4. **Use custom values** — activates the text area below into which you can manually enter a carriage-return list of custom values that will become the available values in the value list.
5. **Cancel** — closes the dialog without saving changes.
6. **OK** — saves changes to the value lists and close the dialog.

Creating Value Lists

There are three settings that control where and how value lists generate their content:

- Using custom values
- Using values from another file
- Using values from field

Using Custom Values

Creating a value list with static values is an easy process. Once the Manage Value Lists dialog is open, follow these steps:

1. Click the New button. This will open the Edit Value List dialog.
2. Enter a name for the value list.
3. Select the Use custom values radio button.
4. Enter some values into the text area. In the example shown in Figure 11-3, three values are entered: Red, Green, and Blue.

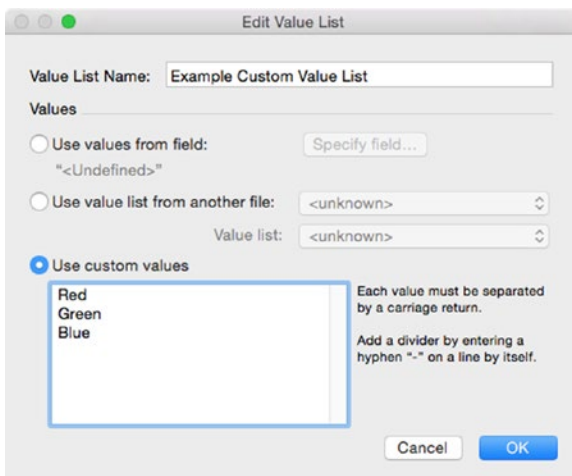


Figure 11-3. An example of a value list using custom values

5. Click the OK button to save the changes.

Using Values from Another File

When building multi-file database systems, using a value list from another file can help eliminate redundancy between the different files. A value list can be created in one database file and then be used by other files. Any changes to the list in the primary file are automatically updated in any values lists that subscribe to it.

Once you have another file with at least one value list entered, open the Manage Value Lists dialog in the file that will use it and then follow these steps:

1. Click the New button. This will open the Edit Value List dialog.
2. Enter a name for the value list.
3. Select the Use value list from another file radio button.
4. If you have already defined the other database as an external file source, it will show up in the first pop-up menu. If not, you will need to define that now by selecting Add FileMaker Data Source from that menu. In the Open File dialog, locate and select the other database and click OK. This will add the file to the list of external data sources and add it to the pop-up menu.
5. Once the file has been selected from the first pop-up menu, the Value list pop-up menu below will become activated. From this menu, you can select any of the value lists in the external file, as shown in Figure 11-4.

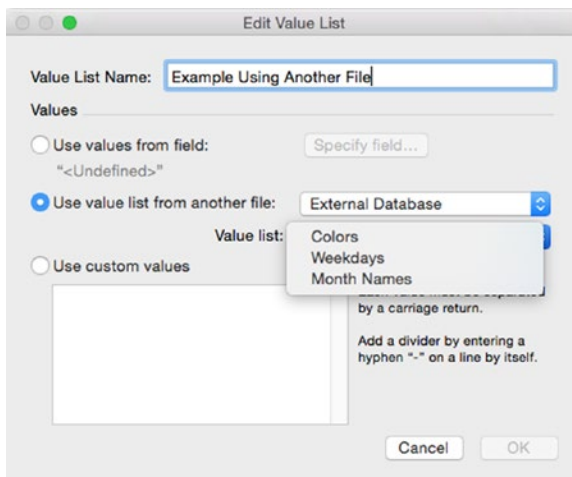


Figure 11-4. The value list menu displaying all the lists defined in an external database file

6. Click the OK button to save the changes.

Using Values from a Field

The most dynamic of the list options, Using values from field, will automatically create a list of values from the actual contents of one or two fields and manage that content based on various settings. The values in the list will be constantly updated to reflect the data in the selected fields. Like the other list types, most of the options available when using values from a field do not care about context.

Create a new value list, assign it a name, and select the Use values from first field radio button. This will automatically open a Specify Field dialog, shown in Figure 11-5.

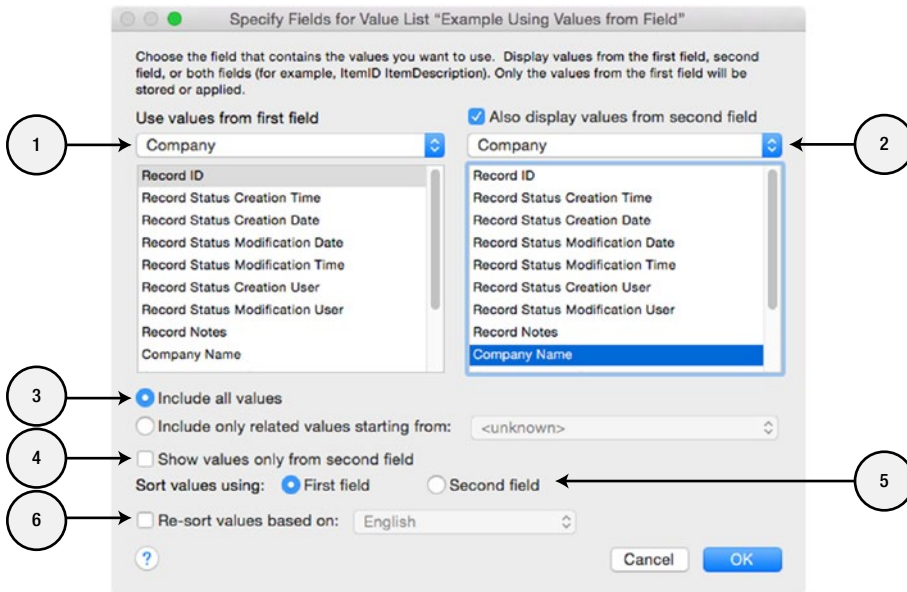


Figure 11-5. The Specify Fields for Value List dialog

The top portion of the dialog contains controls for specifying which field(s) will be used to generate the value list. The lower portion contains settings that control which values are included and how they are sorted and displayed.

1. Use values from first field — This combination of a table occurrence pop-up menu and corresponding field list allows the selection of the *primary field* that will be used to generate values for the list.
2. Also use values from second field — Optionally, a value list can be a generated from a combination of *two* fields by checking this box and selecting another table occurrence and field. The second field, which can be in the same table as the first or a related one, is used for display purposes only on the list of values and is not entered in a field when the user selects a value. The most common example of a use for this feature is when the first field uses a serial number field to generate a list of record ids that can be selected for placement into a field when assigning one record to another. By choosing a second field, the list will include a descriptive component so the user knows what they are selecting even though the serial number is what will be entered in the field. For example, if the first field contains 052136 and the second field contains a name John Smith, it will show up in the value list as a combination of the two: 052136 John Smith. The user looks at the second field to identify the record they are selecting while the serial number from the first field is what is entered in the field when they do so.

3. These two radio buttons control which values are included. The choices are:
 - a. Include all values — Choosing this option will generate a list of unique values from the selected field for *all records* in the table.
 - b. Include only related values starting from — Choosing this option will generate a list of unique values from the selected field for *only* those records related through the context of a table occurrence you select from the adjacent pop-up menu. This provides the ability to create a *contextual value list* that changes values depending on which record a user is viewing (see “Creating Conditional Lists,” this chapter).
4. Show values only from second field — When enabled, this option will hide values from the first field and only display values from second field. However, when the user selects a value, the value from the first field is still entered. For example, when the first field contains a serial number and the second field a name, the user only sees the name and makes their selection, but the serial number is entered in the field they are editing.
5. Sort values using — When generating a value list from two fields, these radio buttons allow you to control which field will be used to sort the list. These are disabled and set by default when the list doesn’t use two fields or when the Show values only from second field option is selected.
6. Resort values based on — This check box activates an adjacent pop-up menu allowing the selection of a language by which the values should be sorted. This is especially useful when using languages where the dictionary sort order is different from the indexed sort order, like languages that distinguish between characters with and without diacritical marks in a dictionary sort but not in an index sort. For more information on the implications of this setting, consult the FileMaker documentation.

Building Value Lists from Field Values

When building a value list from a field, there are two primary choices:

- Creating lists of all records in a table
- Creating conditional lists

Creating Lists of All Records in a Table

The easiest type of field-generated value list is one that displays values pulled from *every record* in a table. This type of value list is useful in a variety of situations, for example:

- A Contact record may have a field allowing the selection of a parent Company.
- A Product record may be selected and assigned to an Invoice record.
- A Template record may be selected as the source body content for an Email record.

In each of these cases, the value list must show every available record in the other table giving the user the option to choose the one that should be assigned. Such a list would be configured using the `Include all values` radio button and selecting the desired field(s) from the primary occurrence of the table. For example, a value list in the `Learn FileMaker` database named `Company All` would be configured as shown in shown in Figure 11-6.

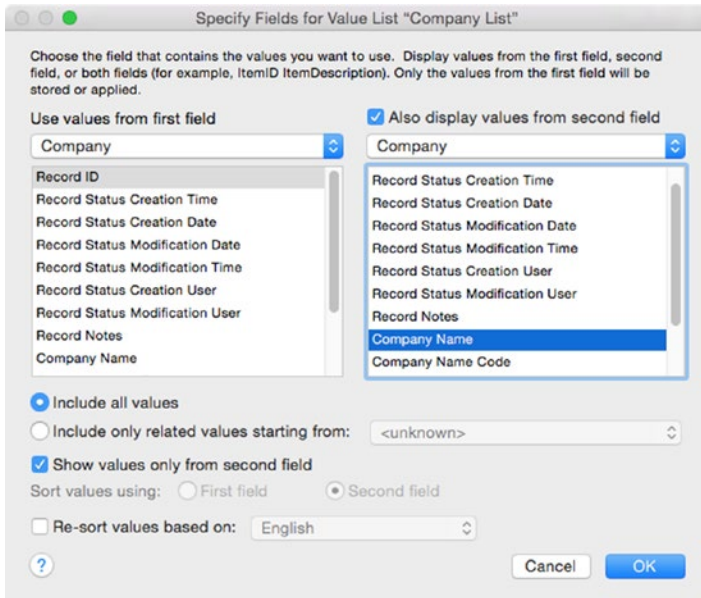


Figure 11-6. A value list configuration that will show all company records

Once saved, this value list can be assigned to a field as a pop-up menu in any table. For example, in the `Contact` table, the `Contact Company ID` field is used to store a foreign key that connects a contact to a company through the `Contact | Company` relationship. So, to allow a user to quickly enter the `Record ID` for a company, the value list can be assigned to that field as a pop-up menu on the `Contact Form` layout.

When the user clicks on the field, a list of every company will be displayed. Because the first field is the `Record ID`, a selection will cause the ID of the company to be entered in the field, thereby establishing the foreign key and connecting the two records.

Creating Conditional Lists

The previous example generates a value list of *every* `Company` record in a table. However, to generate a list that varies, you need to use some type of conditional value list.

A *conditional value list* is a list of field values that is limited to include only a subset of the total values contained in the specified field(s).

This type of value list is useful in a variety of situations, for example:

- When assigning a parent **Company** to a **Contact** limiting the value list to only include active companies.
- When assigning a **Product** to an **Invoice** limiting the value list to only include a certain category of products.
- When selecting a **Template** to use as the source body content for an **Email** limiting the value list to only include approved records for categories that are relevant to the type of email being generated.

There are two ways to create a conditional value list:

- Using a dedicated relationship.
- Using a calculation field.

For examples of each, we will use the **Company Status** field in the **Company** table of the **Learn FileMaker** database to create a value list that is conditioned to only display the **Company Name** of records that have a status value of **active**.

Using a Dedicated Relationship

The first method of creating a conditional value list was briefly mentioned in our exploration of the **Specify Fields for Value List** dialog, using the **Include only related values starting from** radio button.

It requires a dedicated relationship that connects a table occurrence acting as the layout context where the user will see and interact with the value list and another related table occurrence from which the values will be pulled. In this configuration, the relationship acts as the limiting agent; the list is constructed from only those records that are matched to the current record through the relationship to another table. This may sound confusing, but it is easy to set up.

Before working on the value list, performed these steps to create the required relationship:

1. Create a new calculation field in the **Contact** table named **Contact Company Status Match** with a formula that evaluates to a literal text string of "Active" and has a calculation result type of text. Since the resulting value will be the same for every record and since it will be a local key, the field can be configured to store a global value.
2. Create a new occurrence of the **Company** table and name it **Contact | Company for Active List** and position it to the right of the **Contact** table. This will be the occurrence from which the list will pull values.
3. Create a relationship that connects the new **Contact::Contact Company Status Match** field to the **Company Status** field in the new occurrence, as shown in **Figure 11-7**. From the perspective of the **Contact** table, this relationship limits the accessible values to only those where the match fields both contain a value of **Active**, thereby filtering the values so that only active companies are accessible through that relationship.

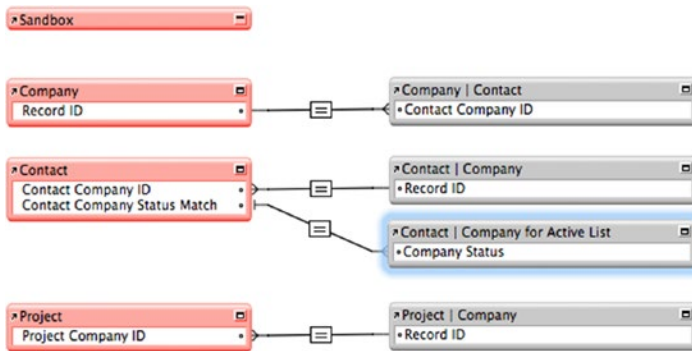


Figure 11-7. The new relationship that will be used to pull values into a conditional value list

Next, create a new value list, as shown in Figure 11-8, which uses the new related occurrence to generate a conditional list of active companies.

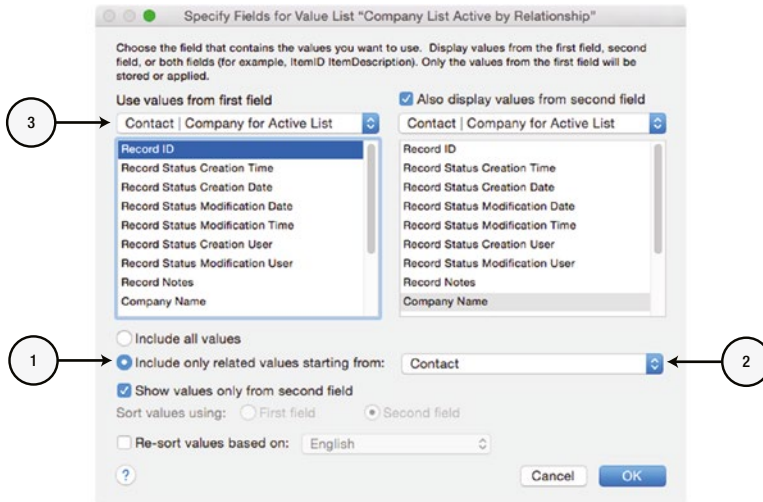


Figure 11-8. The new value list that will generate a list of active companies using the new relationship

The new list should be configured as follows:

1. The Include only related values starting from radio button is selected to indicate that the values used to generate a list will be pulled through a relationship the context specified in the adjacent pop-up menu.
2. The Contact table occurrence is selected to establish the starting context, which indicates the side of the relationship represented by the layout upon which the user will be perceiving the value list and making a selection. In other words, the user will be looking at a Contact layout when editing a field using the value list.
3. The selection of the new table occurrence instructs the value list to pull values *through* our new limiting relationship instead of directly from the table.

Once saved, this value list can be assigned to the Contact Company ID field on a Contact layout as the source for a pop-up menu and it will display a list of only the active companies (Chapter 30).

Understanding the Drawback of Using a Dedicated Relationship

Using a dedicated relationship is a great way to create a conditional value list and is required in many cases. However, in the example above and many similar situations, it is not always the best choice.

Consider that we added a calculation field to the Contact table with a hard-coded value, a table occurrence, and a relationship. These resources would be necessary if the user was given the option to select a status of Active or Inactive and then have the value list filter accordingly. However, when the list is hard-coded to Active only, creating these resources is unnecessary.

Furthermore, by gathering values through a relationship, the list is inherently limited to only being used on layouts for the starting table occurrence, in this case, Contacts. But a list of active companies could be used in *every* table that needs to connect a record to a company. In a typical database, this might include quite a few tables including Contact, Project, Budget, Resource, Equipment, and more.

The problem with the relational method is that, since the filtering mechanism is a relationship, which is always based on the usage context, each use of the same list would need its own set of duplicate resources: a link match field, a dedicated new table occurrence of Company, a relationship between that and the usage occurrence, and a dedicated value list. So, implementing a value list of active Company records in six different tables would require one set of those four resources for each use, giving you a total of twenty-four additional components. This is horribly impractical and easily avoidable.

In cases like this, using a calculation field in the remote table alleviates these extraneous resources.

Using a Calculation Field

Where possible, it is more technically efficient to create a conditional value list using a calculation. This is especially true when there is a *single condition*, determined by a local indexed field, which determines whether a record should be included on or excluded from the list. The example from the last section is a perfect candidate for this approach.

A conditional value list controlled by a calculation field does *not* require the new match field, the additional table occurrence, or the new relationship. Also, because it is not dependent on a relationship to filter values, once a single value list is built using a calculation field, it can be used on *any* layout without requiring any additional resources!

The technique works exactly like a value list that generates values from every record except that it draws values from a calculation field with a formula that *includes* or *excludes* the value from another field based on some filtering criteria. For example, a calculation field can result in a Company Name if the Company Status field has a value of Active and an empty string or nothing if the value is Inactive. When the value list is generated from this calculation field, it ignores all the blank records that makes a list of the names of only those companies who are active.

■ **Note** If using a secondary field when generating the conditional list, only the second field needs to be generated from a conditional calculation.

To create a calculation-driven conditional value list that provides a result identical to the previous example, follow these steps:

1. Create a new calculation field named Company Active ID that returns a text result and has a formula of:

```
Case ( Company Status = "Active" ; Company Name ; "" )
```


2. Create a value list that uses the new calculation field for the second field and choose Include all records, as shown in Figure 11-9.

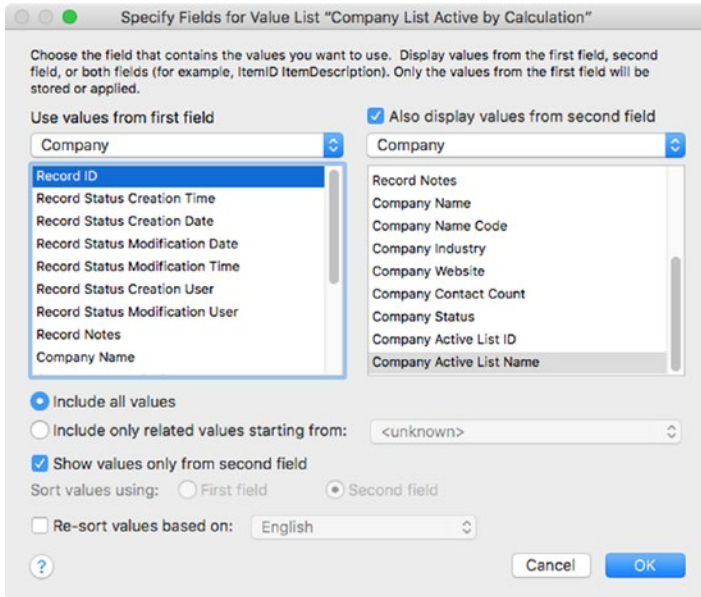


Figure 11-9. The new value list that will generate a list of active companies using the new calculation field

The result will be a conditional value list that can be used from *any* layout context, regardless of the source table and without a relationship between that table and the Company table.

Summary

In this chapter, we discussed how to define value lists from the perspective of the data structure.

PART III



Writing Formulas and Using Functions

FileMaker provides a library of over 300 built-in functions that are ready to insert into calculation formulas, custom functions, and script step configuration panels. These chapters discuss formula creation and provide a comprehensive, categorized reference to every available function:

12. *Writing Formulas*
13. *Text Functions*
14. *Text Functions (JSON)*
15. *Text Formatting Functions*
16. *Number Functions*
17. *Date, Time, and Timestamp Functions*
18. *Container Functions*
19. *Aggregate, Summary, and Repeating Functions*
20. *Financial and Trigonometric Functions*
21. *Logical Functions*
22. *Get Functions*
23. *Design Functions*
24. *Mobile Functions*
25. *Creating Custom Functions*
26. *Introducing ExecuteSQL*

CHAPTER 12



Writing Formulas

A *calculation formula* is an equation made up of one or more text statements expressing operation(s) that will be performed to manipulate values and arrive at a result. Relationships add to the basic data framework by allowing for the *interconnection* of tables and fields. Formulas unlock another level by allowing the *manipulation* of the information stored within that structure.

Formulas can be written to analyze, compare, concatenate, condense, convert, expand, format, parse, replace, or summarize any of FileMaker's supported data types. They can range in size from a simple mathematical equation to an extremely complex collection of interrelated expressions and nested logical clauses. The only limitation is that a single formula cannot exceed FileMaker's limit of 30,000 characters.

In this chapter, we will discuss the following topics:

- Introducing formulas
- Defining formula components
- Exploring the calculation interface
- Writing formulas
- Adding calculations to the Learn FileMaker file

Throughout this chapter, use the Sandbox table and layout in the Learn FileMaker database to experiment with the example formulas mentioned in this chapter.

Introducing Formulas

Formulas are written in a Specify Calculation dialog window, like the one shown in Figure 12-1.

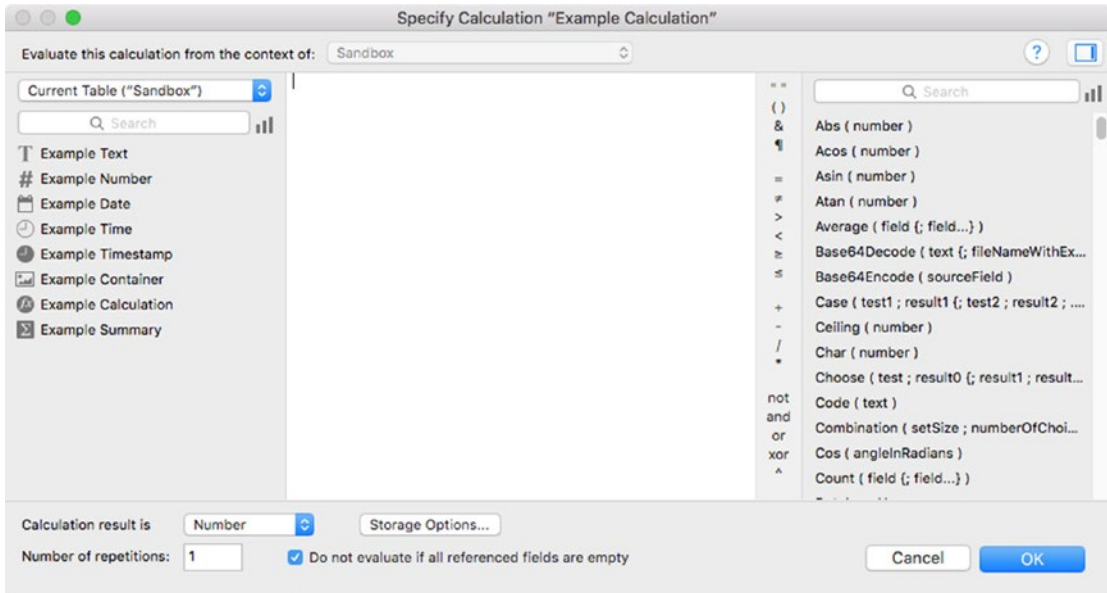


Figure 12-1. A Specify Calculation dialog window for defining a calculation field formula

These dialogs are found throughout the FileMaker development environment in the locations listed in Table 12-1. Generally, any of the following interface elements indicate that a calculation formula can be entered:

- An Options button.
- A Specify button.
- A Fx icon button.
- A pencil icon button.
- A Specify... option at the bottom of a pop-up menu.

Table 12-1. Places where formulas are used to dynamically calculate a value within a database

Location Category	Uses
Custom Menus	<ul style="list-style-type: none"> • Menu name. • Menu visibility.
Field Definitions	<ul style="list-style-type: none"> • Auto-enter value. • Calculation field formula. • Field validation formula.
Specify Dialogs	<ul style="list-style-type: none"> • Repetition number in a Specify Field dialog. • Parameter in a Specify Script dialog.
Layout Inspector Options	<ul style="list-style-type: none"> • Hide object formula (Data panel). • Placeholder text formula (Data panel). • Tooltip text formula (Position panel).
Layout Objects	<ul style="list-style-type: none"> • Button Bar name formula. • Button name formula. • Popover name formula. • Tab name formula. • Portal filtering formula. • Chart (various dynamic content). • Web Viewer (various dynamic content).
Miscellaneous Locations	<ul style="list-style-type: none"> • Replace Field Contents dialog. • Custom function formula (old style specify calculation dialog). • From the Data Viewer, the Edit Expression dialog. • Numerous script steps. • The Edit Privilege Set dialog access through the Manage Security dialog has numerous options to dynamically specify actions that a privilege can perform.

How Formulas Work

Much like their mathematical counterparts, calculation formulas are composed by assembling various components together to form one or more expressions that make up the overall formula statement.

First, a formula must be written and saved to become part of the database's functionality. During the save process, FileMaker will scan the statement and check its expressions for syntax errors such as broken field references, missing function or incorrectly structured statements. When a problem is detected, the save process is halted, the error highlighted in the statement, and a dialog message explains the details of the error. After the developer corrects the error and another save is attempted, this process will be repeated until there are no remaining errors and the code can be successfully compiled and saved.

Once saved, a formula remains idle until it is *called* when some process requests that it evaluate and produce a result. A call may come because of the following:

- A layout attempting to render a calculation field.
- The execution of a script that contains a step with a calculation.
- A user typing into a field with an auto-enter calculation.
- A call from one calculation that uses another in its formula.

When called, the formula's statement is *evaluated*, which means that the code is converted into a *result* by working through each expression's operations from left to right based on specific operator precedence.

Formulas are evaluated within a *context* to allow local and related fields to be accessed from the perspective of a specific table occurrence. A formula that provides a value for an interface element will use the context of the table occurrence of the layout upon which the object is being rendered. A formula used in a field definition requires a manually selected context of any table occurrence for the table in which the field exists.

The formula's result is a single value that will be returned to whatever process requested the execution of the formula. A calculation field's formula displays the result on a layout as if it were data typed into the field. The Set Variable script step places the result of its formula into a variable. The type of object that called the formula determines the destination or use of the result.

If the formula is unable to create a result due to an error, it will return a question mark. For example, if a formula is using a field that has been deleted or is inaccessible from the current context, the formula may return an error. Also, using a custom function that has been deleted or returning a data type inconsistent with the expectations of the calling process will return an error.

Defining Formula Components

Formula expressions can be built using any combination of the following items:

- Comments
- Constants
- Field references
- Functions
- Operators
- Reserved keywords
- Variables

Comments

A *comment* is a text string inserted into a formula that is ignored when the code is evaluated. Comments are intended to act as integrated documentation for developers and can be beneficial as a reminder both to yourself and to others who may see your code in the future. Comments can provide details about how the formula works, how to use it, and to list recent changes and any other information you may think necessary to embed in your code. A comment may be placed at the top of the formula or between different statements.

There are two styles of comment available; multi-line (C style) and single-line (C++ style).

Multi-Line Comment

A *multi-line* comment uses an *initiating* and *terminating* symbol to indicate the start and end of the comment and can, therefore, comment out multiple lines of text. To enter a multi-line comment, use a `/*` to indicate the *start* of the comment and `*/` to indicate the *end* of the comment. Any text between these two symbols will be ignored by FileMaker when the formula is evaluated.

```
/*This is a comment
The comment continues on additional lines
Until you terminate it*/
```

If desired, extra carriage returns can be added to make the text more readable.

```
/*
This is a comment

The comment continues on additional lines

Until you terminate it
*/
```

Single-Line Comment

A *single-line* comment uses a symbol at the beginning of a paragraph, commenting out any text between it and a next carriage return.

To enter a single-line comment, place two forward slashes at the start of the line or at the point on a line where a comment should begin. Any text to the right of these symbols but before the next carriage return will be ignored by FileMaker when the formula is evaluated.

```
// This is a comment
// To continue on a second line you must use another set of symbols
2 + 2 // This comment starts in the middle of a line, after a formula
```

Throughout this book, the result of a formula will be included after the code as a comment. For example:

```
<CodeExpression> // result = <value>
```

Constants

A *constant*, sometimes referred to by developers as a *literal*, is a static, unchanging value that is literally typed into a formula. The value can be any of FileMaker's data types: text, number, date, time, or timestamp. A quantity, city name, event date, and start time are all examples of constants when they are typed *directly* into a formula as part of the formula rather than being represented dynamically by a variable or field.

Numeric constants can be typed directly into a formula. A text constant must be contained within quotation marks so that it is interpreted as a literal value instead of as a field or function name. Date and time constants must be typed within quotation marks and, to be recognized and treated as something other than text, they may need to be converted into date and time objects using a built-in function: `GetAsDate`, `GetAsTime`, and `GetAsTimestamp` (Chapter 13).

The following are examples of each type of constant, as they might appear in a formula:

```
150
"New York"
"1/15/2017"
GetAsDate ( "1/15/2017" )
"10:15:00"
GetAsTime ( "10:15:00" )
"1/15/2017 10:15:00"
GetAsTimestamp ( "1/15/2017 10:15:00" )
```

Field References

A *field reference* is a dynamic pointer to a field. References inform FileMaker to extract a value from a specific field, through a specific context, and to use that value in place of the reference when evaluating the expression.

Depending on the type of calculation and the location of the field relative to the evaluation context of the calculation formula, a field reference is considered either *local* or *related* and can be stated as either the name of the field or the name of the table occurrence and field.

A *local field* is a field that exists within the table of the current context. For a calculation field, the current context is the table containing the field. For a formula used in a custom menu, script step, custom function, etc., the context will always be the source table of the table occurrence assigned to the layout displayed in the current window at the time of evaluation.

A local field reference inside of a calculation field's formula can take the form of only the name of the field. For example:

```
First Name
Project Name
Invoice Number
```

All other field references must include the name of the table occurrence preceding the field name and separated by two colons. The addition of the table occurrence informs FileMaker of precisely which field you are referring to and allows it to attempt to acquire the value through the appropriate relational conduit.

For example:

```
Contact::First Name
Project::Project Name
Invoice::Invoice Number
```

This applies to both:

- References to a *related field* inside of a calculation field's formula.
- Both *local* and *related* field references used in any other object's formula.

Keeping Field References Dynamic

Unlike a static text string, a field reference is *dynamic* and will automatically updated to reflect any change in the name of the field or table occurrence. A field reference *can* be placed within quotation marks as a constant and still work as expected in most cases. However, as a text string, it will no longer be updated if a name change were to occur. Putting a field reference in a formula as a constant imposes the burden on you

to manually change every field reference each time a table or field name is changed. Therefore, it is *highly recommended that all field references remain dynamic*.

To illustrate the difference between a dynamic field reference and a constant field reference, consider the example below, which shows two methods of accessing a field value using the `GetField` function (Chapter 21). The first is a text-based reference, hard-coded into a static text string, while the second is a dynamic reference. Once set up, they each return the identical field value.

```
GetField ( "Contact::First Name" )    // result = John
GetField ( Contact::First Name )     // result = John
```

However, when the name of the field is changed from `First Name` to `NameFirst`, only the second will automatically update since it is a dynamic reference instead of static text. Since the first example no longer refers to an actual field, it will evaluate to an error.

```
GetField ( "Contact::First Name" )    // result = ?
GetField ( Contact::NameFirst )      // result = John
```

The same problem will occur if the table name is modified.

The problem is that some built-in functions *require* a text-based field reference. In these cases, use the `GetFieldName` function (Chapter 21) to convert a dynamic field reference into a text string for that purpose, as shown here:

```
GetFieldName ( Contact::NameFirst )   // result = "Contact::NameFirst"
```

A dynamic reference converted this way can be used in other expressions that require a text-based reference without a concern for future name changes.

Functions

A *function* is a predefined, named formula that may accept parameters and is evaluated to return a single result. Functions are like subroutines in other programming languages. They allow a subset of functionality to be offloaded to a process outside of the current formula and make possible efficient organization of code. Within FileMaker, there are two kinds of functions: *Built-in functions* (Chapters 13-24) and *Custom functions* (Chapter 25). In this section we will briefly illustrate how to call any function from within a formula.

Calling a Function from a Formula

Any function can be called from within a formula by invoking its name in the statement. Generally, functions are named with two or more capitalized words together without spaces that briefly describe the processing performed. For example, in pattern:

```
FunctionName
```

■ **Note** The capitalization and lack of spacing shown in this example are used consistently for FileMaker's built-in functions but are not a requirement for custom functions. Although not required, for the sake of consistency and readability, you may want to adhere to this convention for your custom functions.

Calling a Function with Parameters

Most functions will accept parameters. A *function parameter* is a value that can vary with each call to the function. This allows information to be sent by the calling formula, pushed into the function. It might be raw material that the function will manipulate, instructions about the desired processes that should be performed, or information about what is being requested.

■ **Note** Except for any limitations imposed by a specific function or by the nature of the data a parameter is designed to accept, a parameter can accept data as a *constant*, *field reference*, *expression*, *variable*, *literal*, or even a nested *function*.

Parameters are listed after the function name, enclosed within a set of parentheses, as shown here:

```
FunctionName ( parameter )
```

If a function accepts more than one parameter, they are listed in order, separated with semicolons after the function name, enclosed within a set of parentheses, as shown here:

```
FunctionName ( parameter1 ; parameter2 ; parameter3 )
```

The value inserted into a parameter can be an expression. In this example, the one parameter is made up of two fields being multiplied together, the result of which will be sent to the function as the single parameter:

```
FunctionName ( Invoice Tax Rate * Invoice Subtotal )
```

Optional Parameters

Some built-in functions offer optional parameters that can be used or ignored as needed. When inserting a function call into a formula, FileMaker will surround optional parameters with braces as shown in this example:

```
FunctionName ( parameter1 {; parameter2 ; parameter3} )
```

In this case, the braces indicate that *parameter1* is required while *parameter2* and *parameter3* are optional. Optional parameters that you *won't* be used can be removed by deleting them along with the braces. If they *will* be used, only the braces should be removed. This example illustrates the pattern of a single function call without optional parameters and a second call to the same hypothetical function with the optional parameters included:

```
FunctionName ( parameter1 )
FunctionName ( parameter1 ; parameter2 ; parameter3 )
```

Nesting Function Calls

Since a parameter can accept an expression as a value, calls to *other functions* can be nested as a parameter value. In this example, *Function2* will be evaluated first and that result will be used as the parameter for *Function1*.

```
Function1 ( Function2 ( parameter ) )
```

Operators

An *operator* is a symbol used to express a type of operation within a formula. The most familiar examples are mathematical operators where the plus symbol in $2 + 2$ is the operator. FileMaker has many different operators, each executing in an order of precedence.

Defining Operators

FileMaker has four categories of operators:

- Comparison operators
- Logical operators
- Mathematical operators
- Text operators

■ **Note** For clarity, most of the examples provided in this section use operators with formulas containing only constants. However, they can certainly be used with any combination with other formula components.

Comparison Operators

A *comparison operator* is a symbol used to construct an equation that compares two values, either a literal value or the evaluated results of two expressions. The result of an expression using a comparison operator will be a true or false Boolean value, represented respectively by a 1 or 0.

Equal To

An *equal to* symbol is used to compare two values for similarity. The equation will evaluate to true if the values are identical and false if they are not.

```
100 = 100           // result = 1
150 = 100          // result = 0
```

```
"New York" = "New York" // result = 1
"New York" = "NY"       // result = 0
```

```
GetAsDate ( "1/15/2017" ) = GetAsDate ( "1/15/2017" ) // result = 1
GetAsDate ( "1/15/2017" ) = GetAsDate ( "7/20/2018" ) // result = 0
```

```
GetAsTime ( "9:30:00" ) = GetAsTime ( "9:30:00" ) // result = 1
GetAsTime ( "9:30:00" ) = GetAsTime ( "12:15:00" ) // result = 0
```

Not Equal To

Comparing two values for dissimilarity can be performed using either a *not equal to* symbol or a *less than* and *greater than* symbol side by side. An equation will evaluate to false if the values are identical and true if they are not, as demonstrated in these examples:

```
100 ≠ 100           // result = 0
150 ≠ 100           // result = 1

"New York" <> "New York" // result = 0
"New York" <> "NY"       // result = 1

GetAsDate ( "1/15/2017" ) ≠ GetAsDate ( "1/15/2017" ) // result = 0
GetAsDate ( "1/15/2017" ) ≠ GetAsDate ( "7/20/2018" ) // result = 1

GetAsTime ( "9:30:00" ) ≠ GetAsTime ( "9:30:00" ) // result = 0
GetAsTime ( "9:30:00" ) ≠ GetAsTime ( "12:15:00" ) // result = 1
```

■ **Tip** The ≠ symbol can be created by typing an equal sign while holding the Option key on macOS.

Greater Than

A *greater than* symbol is used to compare the relative alphabetical or numerical positions of two values. The equation will evaluate to true if the value on the left is greater in position compared to the value on the right.

```
150 > 100           // result = 1
100 > 100           // result = 0

"Bear" > "Automobile" // result = 1
"Atlanta" > "New York" // result = 0

GetAsDate ( "1/15/2017" ) > GetAsDate ( "7/20/2016" ) // result = 1
GetAsDate ( "1/15/2017" ) > GetAsDate ( "7/10/2017" ) // result = 0

GetAsTime ( "12:30:00" ) > GetAsTime ( "9:15:00" ) // result = 1
GetAsTime ( "9:30:00" ) > GetAsTime ( "10:45:00" ) // result = 0
```

Greater Than or Equal To

Comparing the relative alphabetical or numerical positions of two values can be performed using either a *greater than or equal to* symbol or a *greater than* symbol followed by an *equal* sign. The equation will evaluate to true if the value on the left is greater in position or the same value compared to the value on the right, as shown in these examples:

```
100 ≥ 100           // result = 1
100 ≥ 150           // result = 0
```

```

"Bear" >= "Automobile"    // result = 1
"New York" >= "Ohio"      // result = 0

GetAsDate ( "1/15/2017" ) ≥ GetAsDate ( "7/20/2016" )    // result = 1
GetAsDate ( "1/15/2017" ) ≥ GetAsDate ( "7/10/2017" )    // result = 0

GetAsTime ( "12:30:00" ) ≥ GetAsTime ( "12:30:00" )      // result = 1
GetAsTime ( "9:30:00" ) ≥ GetAsTime ( "10:45:00" )       // result = 0

```

■ **Tip** The \geq symbol can be created by typing a *greater than* symbol while holding the Option key on macOS.

Less Than

A less than sign $<$ is used to compare the relative alphabetical or numerical positions of two values. The equation will evaluate to true if the value on the right is greater in position than the value on the left.

```

100 < 350          // result = 1
100 < 100          // result = 0

"Automobile" < "Car"    // result = 1
"New York" < "New York" // result = 0

GetAsDate ( "7/20/2016" ) < GetAsDate ( "1/15/2017" )    // result = 1
GetAsDate ( "7/10/2017" ) < GetAsDate ( "1/15/2017" )    // result = 0

GetAsTime ( "9:15:00" ) < GetAsTime ( "12:30:00" )       // result = 1
GetAsTime ( "10:45:00" ) < GetAsTime ( "9:30:00" )       // result = 0

```

Less Than or Equal To

Comparing the relative alphabetical or numerical positions of two values can be performed using either a *less than or equal to* symbol or a *less than* symbol followed by an *equal* sign. The equation will evaluate to true if the value on the left is greater in position or the same value compared to the value on the right, as shown in these examples:

```

100 ≤ 100          // result = 1
100 ≤ 150          // result = 0

"Automobile" <= "Bear"    // result = 1
"Ohio" <= "New York"      // result = 0

GetAsDate ( "7/20/2016" ) ≤ GetAsDate ( "1/15/2017" )    // result = 1
GetAsDate ( "7/10/2017" ) ≤ GetAsDate ( "1/15/2017" )    // result = 0

GetAsTime ( "12:30:00" ) ≤ GetAsTime ( "12:30:00" )      // result = 1
GetAsTime ( "10:45:00" ) ≤ GetAsTime ( "9:30:00" )       // result = 0

```

■ **Tip** The \leq symbol can be created by typing a *greater than* symbol while holding the Option key on macOS.

Logical Operators

A *logical operator* is a keyword used to build compound conditions into a formula by joining two or more separate expressions into a single expression or, in one case, to negate a single expression, thereby reversing a Boolean result. The result of an expression using a logical operator will be a Boolean.

AND

The AND operator is used to combine two separate Boolean expressions into a combined equation that will evaluate to true only if the result of both expressions are true.

```
150 > 100 AND "Bear" > "Automobile" // result = 1
150 > 100 AND "Bear" = "Automobile" // result = 0
```

OR

The OR operator is used to combine two separate Boolean expressions into a combined equation that will evaluate to true if the result of at least one of the two expressions is true.

```
150 > 100 OR "Bear" < "Automobile" // result = 1
150 < 100 OR "Bear" = "Automobile" // result = 0
```

XOR

The XOR operator is used to combine two separate Boolean expressions into a combined equation that will evaluate to true if the result of only one of the two equations is true. If both equations are true, it will return false.

```
150 > 100 XOR "Bear" < "Automobile" // result = 1
150 > 100 XOR "Bear" ≠ "Automobile" // result = 0
150 < 100 XOR "Bear" = "Automobile" // result = 0
```

NOT

The NOT operator is a unique logical operator that will negate the result of any expression to its right, thereby reversing the Boolean result of that expression. If an expression evaluates to true, putting this operator in front of it will reverse it to return false.

```
NOT 150 < 100 // result = 0
NOT "Tuesday" < "Monday" // result = 1
```

A similar reversal can also be achieved by the following statement, which will return true if the expression in parenthesis evaluates false.

```
( 150 < 100 ) = 0
```

■ **Tip** While the order of operator precedence should always be kept in mind, some developers prefer to enclose the expression(s) after a NOT operator in parentheses to avoid any confusion and ensure the appropriate result, *especially* with compound expressions.

Mathematical Operators

A *mathematical operator* is a symbol used to perform or control arithmetic computations with one or more values or expressions. The mathematical operations can be performed not only on numbers but on other data types such as dates, times, and timestamps.

Addition

The *plus* symbol is used to add the value on the right to the value on the left.

```
100 + 50 // result = 150
GetAsDate ( "1/5/2017" ) + 5 // result = 1/10/2017
GetAsTime ( "1:15:00" ) + 300 // result = 1:20:00
```

Subtraction

The *minus* symbol is used to subtract the value on the right from the value on the left.

```
100 - 50
GetAsDate ( "1/5/2017" ) - 2 // result = 1/3/2017
GetAsTime ( "1:15:00" ) - 300 // result = 1:10:00
```

Multiplication

The *multiply* symbol is used to multiply the value on the left by the value on the right.

```
100 * 50
```

Division

The *division* symbol is used to divide the value on the left by the value on the right.

```
100 / 50
```

Raising to a Power

The *power of* symbol is used to raise the value on the left to the power of the value on the right.

```
100^2
```

Precedence

A set of *parentheses* is used to change the order of evaluative precedence (discussed later in this chapter). FileMaker will evaluate formulas from left to right after first evaluating expressions that are enclosed within parentheses, working from the inside out and based on an order of precedence. Although these two equations perform the same mathematical operations on the same numbers in the same order, the results are drastically different due to the control of precedence imposed with parentheses:

```
100 * 2 + 50 / 25           // result = 202
( ( 100 * 2 ) + 50 ) / 25   // result = 10
```

Text Operators

A *text operator* is a symbol that is used to construct an equation that combines text items into a single item and performs other text-related functions.

Quotations Mark

A pair of *quotation marks* is used to indicate a literal text constant. Text entered into a formula without quotation marks will be interpreted as a field, function, or variable name. Date and time constants must be enclosed in quotation marks.

```
"John"
"1/15/2017"
```

Concatenate Text

The ampersand symbol & is used to join two text values into a single value.

```
"John" & " " & "Smith"      //result = "John Smith"
```

Carriage Return

The *paragraph carriage return* symbol is used to insert a carriage return into a text value.

```
"John Smith" & "¶" & "Jim Smith"
// result =
John Smith
Jim Smith
```

■ **Tip** The carriage return symbol can be created by typing a ¶ while holding down the Option key on macOS.

Backslash

The backslash symbol \ is used to make an operator character be used literally instead of as an operator. For example, a backslash+quote inserted at the end of this example can represent 5 inches as a text string with the extra quote mark treated as part of the resulting text value.

```
"5\" // result = 5"
```

Without the backslash, FileMaker will interpret the formula as having a syntax error and not allow it to be saved.

```
"5"
```

Reserved Name

Although it is not advisable to do so, you can use a reserved function name or keyword as a table name or field name. However, when doing so and using these in a formula, FileMaker must be informed of the fact so it interrupts the information as the value you intend rather than the function or keyword it also represents. This is accomplished by wrapping the name within curly brackets with a preceding dollar symbol.

The following will be interrupted as a *field* or *table* name rather than an AND operator:

```
${AND}
```

Understanding Operator Precedence

Like with standard mathematical equations, operators in a calculation formula are evaluated in the following order:

1. Comments
2. Space, backslash, paragraph return, reserved name
3. Parentheses
4. NOT
5. Power of (^)
6. Multiplication, division
7. Addition, subtraction
8. Ampersand
9. Equal to, not equal to, greater than, less than, greater than or equal to, less than or equal to
10. OR, XOR

To force an expression to be evaluated in a different order, use parentheses.

Reserved Words

A *reserved word* is a word, term, or symbol that is used by FileMaker for function names, predefined parameters, operators, and to construct SQL statements. These should not be used as names for tables, fields, custom functions, and other objects to avoid problems when referring to these objects in a formula. FileMaker reserves the words shown in Table 12-2.

Table 12-2. FileMaker's reserved words

AllStyles	And	Bold	CentralEurope	Condense
Cyrillic	DatabaseNames	DoubleUnderline	Extend	False
Greek	Higher	HighlightYellow	Italic	Lower
Lowercase	Not	OEM	Or	Other
Pi	Plain	Random	Roman	Self
ShiftJIS	SimplifiedChinese	SmallCaps	Strikethrough	Subscript
Superscript	Symbol	Titlecase	TraditionalChinese	True
Underline	Uppercase	WordUnderline	Xor	

Operators and other symbols, shown in Table 12-3, are reserved.

Table 12-3. FileMaker's reserved operators

+	-	*	/	^	&
=	≠	<	>	≤	≥
(,	;)	[]
"	::	\$	}		

FileMaker reserves SQL keywords, shown in Table 12-4. To avoid syntax errors with the ExecuteSQL function, avoid these as well.

Table 12-4. FileMaker's reserved SQL keywords

ABSOLUTE	ACTION	ADD	ALL	ALLOCATE
ALTER	AND	ANY	ARE	AS
ASC	ASSERTION	AT	AUTHORIZATION	AVG
BEGIN	BETWEEN	BINARY	BIT	BIT_LENGTH
BLOB	BOOLEAN	BOTH	BY	CASCADE
CASCADE	CASE	CAST	CATALOG	CHAR
CHARACTER	CHARACTER_LENGTH	CHAR_LENGTH	CHECK	CHR
CLOSE	COALESCE	COLLATE	COLLATION	COLUMN
COMMIT	CONNECT	CONNECTION	CONSTRAINT	CONSTRAINTS
CONTINUE	CONVERT	CORRESPONDING	COUNT	CREATE
CROSS	CURDATE	CURRENT	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR	CURTIME	CURTIMESTAMP
DATE	DATEVAL	DAY	DAYNAME	DAYOFWEEK
DEALLOCATE	DEC	DECIMAL	DECLARE	DEFAULT
DEFERRABLE	DEFERRED	DELETE	DESC	DESCRIBE
DESCRIPTOR	DIAGNOSTICS	DISCONNECT	DISTINCT	DOMAIN
DOUBLE	DROP	ELSE	END	END_EXEC
ESCAPE	EVERY	EXCEPT	EXCEPTION	EXEC
EXECUTE	EXISTS	EXTERNAL	EXTRACT	FALSE
FETCH	FIRST	FLOAT	FOR	FOREIGN
FOUND	FROM	FULL	GET	GLOBAL
GO	GOTO	GRANT	GROUP	HAVING
HOUR	IDENTITY	IMMEDIATE	IN	INDEX
INDICATOR	INITIALLY	INNER	INPUT	INSENSITIVE
INSERT	INT	INTEGER	INTERSECT	INTERVAL
INTO	IS	ISOLATION	JOIN	KEY
LANGUAGE	LAST	LEADING	LEFT	LENGTH
LEVEL	LIKE	LOCAL	LONGVARBINARY	LOWER
LTRIM	MATCH	MAX	MIN	MINUTE
MODULE	MONTH	MONTHNAME	NAMES	NATIONAL
NATURAL	NCHAR	NEXT	NO	NOT
NULL	NULLIF	NUMERIC	NUMVAL	OCTET_LENGTH
OF	OFFSET	ON	ONLY	OPEN
OPTION	OR	ORDER	OUTER	OUTPUT

(continued)

Table 12-4. (continued)

OVERLAPS	PAD	PART	PARTIAL	PERCENT
POSITION	PRECISION	PREPARE	PRESERVE	PRIMARY
PRIOR	PRIVILEGES	PROCEDURE	PUBLIC	READ
REAL	REFERENCES	RELATIVE	RESTRICT	REVOKE
RIGHT	ROLLBACK	ROUND	ROW	ROWID
ROWS	RTRIM	SCHEMA	SCROLL	SECOND
SECTION	SELECT	SIZE	SMALLINT	SOME
SPACE	SQL	SQLCODE	SQLERROR	SQLSTATE
STRVAL	SUBSTRING	SUM	SYSTEM_USER	TABLE
TEMPORARY	THEN	TIES	TIME	TIMESTAMP
TIMESTAMPVAL	TIMEVAL	TIMEZONE_HOUR	TIMEZONE_ MINUTE	TO
TODAY	TRAILING	TRANSACTION	TRANSLATE	TRANSLATION
TRIM	TRUE	UNION	UNIQUE	UNKNOWN
UPDATE	UPPER	USAGE	USER	USERNAME
USING	VALUE	VALUES	VARBINARY	VARCHAR
VARYING	VIEW	WHEN	WHENEVER	WHERE
WITH	WORK	WRITE	YEAR	ZONE

Variables

A *variable* is a developer-named letter, word, or phrase that can store and be used in place of a value in a calculation formula and elsewhere in the development environment, including some less-obvious places such as:

- Script steps that include a formula.
- File paths dialogs.
- Find request criteria dialogs.
- On layouts as merge variables.
- Data source dialogs.

Types of Variables

FileMaker has three different variable types, each with different naming requirements and scope:

- Local Let statement variables
- Local script/function variables
- Global database variables

The last two have a specific name prefix requirement, a single and double dollar sign respectively. In addition, these general naming guidelines for all variables should be considered:

- Avoid keywords and field names to avoid confusion.
- Be concise; keep the name as short as possible to save space and avoid clutter.
- Be descriptive; make sure the name is clear and indicates what data it will contain.
- Be consistent in case; choose “camel case” (`variableNameExample`), dot-delimited (`variable.name.example`), underscore delimited (`variable_name_example`) or some other format and then stick with it throughout the database.

■ **Caution** Using dot-delimited naming may cause conflicts with JSON functions!

Local Let Statement Variable

A *local Let Statement* variable is a variable that is created and accessible within a single Let statement (Chapter 21). These variables exist temporarily during the evaluation of the calculation portion of the statement. Except for the normal restrictions, FileMaker imposes no naming requirements to a local variable defined in a Let statement. Names can be as short as a single character or a lengthy, multi-word phrase. For example:

```
x
data
firstName
first_Name
dateToProcess
table.name
```

Since these variables are temporary and only accessible from within the confines of the Let statement in which they are defined, there is no fear of them conflicting with variables in other formulas or within the same formula outside of the Let statement containing them. Therefore, to maintain consistency, standardized names can be reused from one formula to the next without concern about them overwriting each other.

Local Function or Script Variable

A *local variable* is a variable that persists temporarily during the evaluation of a custom function or the execution of a script. A local variable can be initialized by two different methods:

- Within a Let function (Chapter 21)
- Using the Set Variable script step (Chapter 35)

Local variables are used for a variety of purposes, including:

- Storing information at one position in a script for use later in the same script
- Sharing information between a script and a custom function
- Storing results or control values from one iteration to the next in a recursive custom function (Chapter 25)

FileMaker requires that local variables be named with a single dollar symbol prefix. Beyond that, the name can be as short as a single character or a lengthy, multi-word phrase and can use any of a variety of delimiters. For example:

```
$data
$firstName
$recordNumber
$dateToProcess
$date.to.process
$date_to_process
```

Since these variables are shared within a script but not between scripts, there is no fear of them conflicting with variables in other scripts. For consistency, standardized names can be reused from one script to the next without concern about them conflicting with each other.

Local variables can be monitored in the Data Viewer during the execution of a script (Chapter 36).

■ **Caution** Local variables with the same name can conflict when a script calls a custom function multiple times.

Global Database Variable

A *global database variable* is a variable that is globally accessible from any formula anywhere within a single database file and will persist until the file is closed. Like local variables, a global variable can be initialized by two different methods:

- Within a Let function.
- Using the Set Variable script step.

They are used for a variety of storage purposes, such as:

- The storage of custom preferences and settings for variable behavior within a database.
- The storage of temporary data being prepared for some output or reporting use.
- The storage of troubleshooting information to help track down problems in complex multi-script processes.

■ **Tip** To avoid clutter and confusion, global variables should not be overused. Limit their use to information that is truly required *universally* within the file due to widespread use by scripts and calculations. Avoid using global variables to pass information between scripts by using parameter passing instead (Chapter 34).

FileMaker requires that global variables be named with a double dollar symbol prefix. Beyond that, names can be as short as a single character or a lengthy, multi-word phrase and can use any of a variety of delimiters. For example:

```
$$data
$$firstName
$$recordNumber
$$dateToProcess
$$date.to.process
$$date_to_process
```

Since these variables are global, they must each be uniquely and carefully named to avoid confusion and conflicts with other global variables in the same file. However, since they are not shared between files, standardized names can be reused from one database to the next without concern about them conflicting with each other.

Global variables can be monitored in the Data Viewer (Chapter 36).

Exploring the Calculation Interface

For now, we will write formulas exclusively in the Example Calculation field of the Sandbox table in our Learn FileMaker database.

To get started experimenting with the examples in this chapter, follow these steps:

1. Open the Learn FileMaker database.
2. Switch to the Sandbox layout.
3. Select the File > Manage > Database menu item.
4. Click on the Fields tab.
5. Choose Sandbox from the Tables pop-up on the upper left.
6. Click on the Example Calculation field.
7. Click the Options button.

That should open the Specify Calculation dialog window.

Exploring the Specify Calculation Dialog

All calculations are written in a Specify Calculation dialog, shown in Figure 12-2. The empty text area in the center of the dialog is where you enter and edit formula code. Surrounding this are several lists that can assist in building a formula and other elements that give you control over certain behaviors.

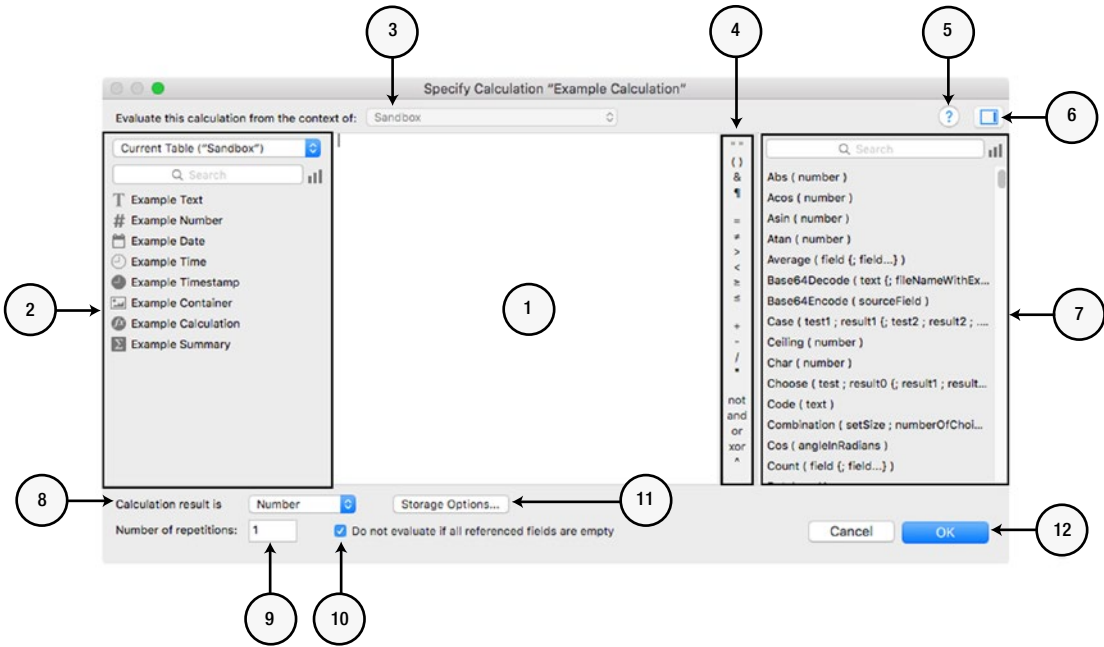


Figure 12-2. The Specify Calculation dialog window

While these controls will be explained in more detail as we create some example formulas, let's take a brief tour to become familiar with these controls:

1. *Formula Text Area* — This central text area is where the calculation formula is written. You can type, drag-drop, copy-paste, or insert content by clicking on the panels to the left and right.
2. *Field Selector Pane* — These controls assist with the insertion of a selected field reference into the formula.
3. *Context Indicator* — When defining a field, formulas require the selection of a table occurrence to act as the context from which the formula will be evaluated. If only one occurrence is available for the table the menu will be disabled. If the formula evaluates from the context of the current window, the menu will be hidden.
4. *Operators* — Click on any of these buttons to add the corresponding operator at the insertion point within the text of the formula.
5. *Help Icon* — Click for the documentation web page of information about this window and its functions.

6. *Functions Pane Toggle Button* — Click this icon to toggle the visibility of the Functions Pane below.
7. *Functions Pane* — These controls allow functions to be selected and added into the formula.
8. *Calculation Result Data Type* — Field definition formulas require selection of the resulting data type from this pop-up menu; other formulas mandate a data type based on the object or property to which the formula pertains. For example, button labels must be text, layout number must be a number, etc.
9. *Number of repetitions* — Field definition formulas can be made repeating by entering a number into this text area.
10. *Do not evaluate checkbox* — Checking this box will cause the calculation to not evaluate and automatically return no value if every field referenced within it is empty.
11. *Storage options* — This button opens a dialog containing indexing and other storage options for calculation fields.
12. *Buttons* — Click the Cancel button to close the window without saving changes. Click OK to save changes and close the window.

Writing Formulas

Formulas can contain any combination of components and range greatly in complexity. Before attempting to write your first masterpiece of coded wizardry, take a moment to work through some simple examples that demonstrate the basic concepts involved.

Getting Started with Simple Formulas

Working through some simple formula examples, starting with the most basic and gradually expanding is a great way to quickly learn the basics. All the examples in this section assume the formula is being typed into the Sandbox::Example Calculation field in the Learn FileMaker database, so you can see the results on the Sandbox Form layout.

Constant-Only Formula

A *constant-only* formula is a formula that evaluates to a literal value without performing any actual operations. It is basically a statement containing a single value that will evaluate to a result of the same value. These have a practical value as a way of storing static information such as a current tax rate or value(s) used as a relationship match field to connect to a subset of related records of a certain type. For now, it will serve as the most basic example to get us started writing a simple formula.

To create your first constant-only formula, enter the value below and select a calculation result data type of “text.”

"Hello World"

The dialog should now look like the one shown in Figure 12-3.

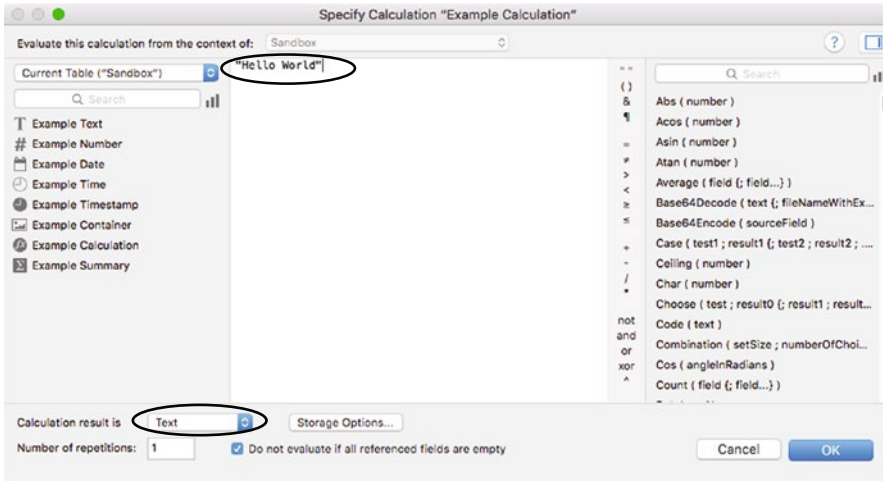


Figure 12-3. A simple “Hello World” calculation formula

Click the OK button to save the formula. The Example Calculation field on the layout should now display the text “Hello World” on every record. It is the same on each record because the formula returns a static value. For it to vary from one record to the next, the expressions must contain some variable components such as field values that change from one record to the next.

Creating Intentional Errors

Before moving forward, let’s pause to clarify the difference between a *syntax error* and a *result error*.

A *syntax error* is an error with the format or language of the code entered causing it to fail to compile. To illustrate this with the simple constant formula, remove the final quotation symbol from the formula:

"Hello World

When you then click the OK button, FileMaker should display the error dialog message shown in Figure 12-4. When a syntax error is detected, FileMaker will not allow you to save the formula until it is correctly formatted.



Figure 12-4. An example error message for formula syntax errors

By contrast, a *result error* is a resulting question mark that indicates a problem with the formulas settings or a component used within one or more of its operations.

To illustrate this, restore the formula's closing quote mark to eliminate the syntax error. Then change the calculation *result* to *date*, *time*, or *timestamp*. FileMaker will let you save the formula because the syntax is correct even though you have selected a result type inconsistent with the type of data (text) being returned by the calculation. When you look at the field on a record, the result will be displayed as a question mark.

Try each data type and see how the results vary.

```
"Hello World"
// result as text = Hello World
// result as number = Hello World
// result as date = ?
// result as time = ?
// result as timestamp = ?
// result as container = Hello World
```

Three of the data types will result in an error. However, even though the number and container results appears to display the correct information, the data in the field will not necessarily behave the way you might expect when searching, sorting, or using the information in other formulas. FileMaker will assume the data is the type you specify in the calculation result setting and this could cause more obvious problems elsewhere in both behavior and performance.

■ **Caution** Troubleshooting error results is a necessary skill as databases develop and begin to involve more and increasingly complex formulas.

Experimenting with Storage Options

As with entry fields, calculation fields have editable storage options that control how the result of a formula is stored. These are edited in a Storage Options dialog, shown in Figure 12-5.

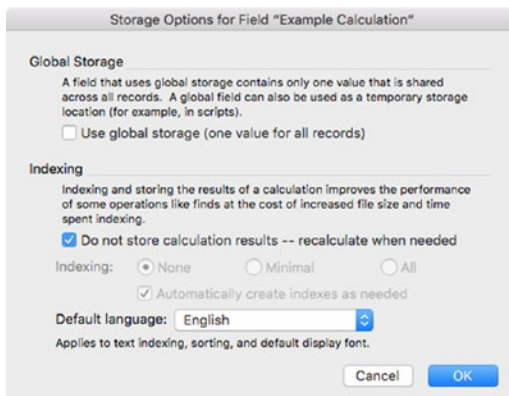


Figure 12-5. The Storage Options dialog with indexing disabled

The global storage and indexing features work the same as they do for Entry fields (Chapter 8). However, there is an additional option for indexing; `Do not store calculation results`. This feature is used to force a calculation to never automatically index results and instead to always recalculate a result anytime the field is accessed or displayed.

To illustrate the importance of this feature, restore the formula in the `Sandbox::Example Calculation` field to the original example — a result type of `Text` and the following formula:

```
"Hello World"
```

Save the formula and confirm that the field displays correctly on records viewed on the `Sandbox` layout. Once confirmed, open the formula again and type two forward slashes in front of the text to convert it into a comment. Now the formula should look like this:

```
// "Hello World"
```

Without making other changes, save the formula and again view it on the `Sandbox` layout. Don't be surprised if it still displays the value "Hello World" as if nothing changed. Select the `Record > New Record` menu and notice that the new record will correctly display a blank value in the field.

The reason for this is the *indexing* setting for the calculation field. While other field types use index settings primarily to pre-process data in a way that will speed up searches and help establish relational connections, a calculation field may not refresh its result when nothing else triggers it to reevaluate, especially when the index settings forbids it.

If you change and save a formula significantly, it *will* prompt every record to refresh the results. For example, change the formula for the `Sandbox::Example Calculation` field to a completely different text constant and when you save, every record will update to display the new value. However, as the example above demonstrates, FileMaker doesn't always force a simple calculation to update when its text constant is commented out or some other minor change is made. To make our simple example work correctly, follow these steps to turn indexing completely off for the calculation field:

1. Open the formula.
2. Click the `Storage Options` button.
3. Check the `Do not store calculation results` check box.
4. Click `OK` to close the `Storage Options` dialog.
5. Click `OK` to save the formula.

This simple example isn't practical and not representative of the kinds of challenges you will face with indexing. However, there will be many times when you need to adjust the index settings. It will take you some time to gain experience to know what will force an update or not. For example, a formula that uses built-in functions to return contextual values such as the number of records in the found set or the current record number within the found set will *require* the `Do not store calculation results` selected to update as you would expect when the user navigates through a found set. By contrast, formulas that include references to fields will update automatically when those field values are modified, even when indexing is allowed.

Inserting Components into Formulas

Within the calculation area, you can enter field references, functions, and other components by various methods including, typing, copying-pasting, dragging-dropping, or double-clicking on a field or function from the panels on either side of the dialog.

Using Auto-Complete

When typing the examples in this chapter so far, you may have noticed FileMaker presenting an *auto-complete suggestion interface* that continuously filters based on the word you are typing. In the example shown in Figure 12-6, after typing just the letter “E,” a list of available components is presented including a mixture of every built-in function, custom function, and field that contains the letter “E”.

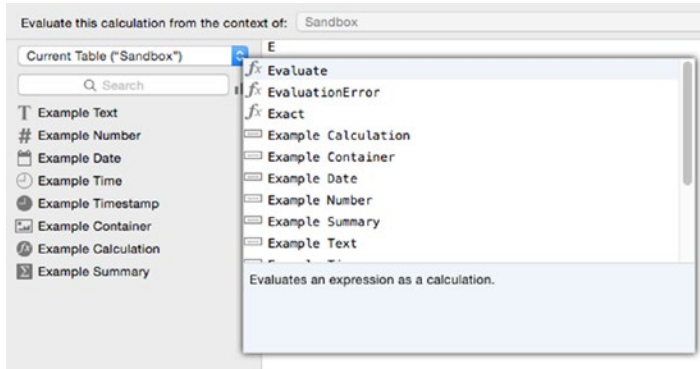


Figure 12-6. An example of the auto-complete function when typing into formulas

As you continue typing — “Ex,” “Exa,” “Exam” — you will see the list continuously filter the list down until you see only a list of our example fields.

When the auto-complete list appears, you have the option to ignore it and continue typing or to use it to automatically insert a value from the list. To do so, select the desired item and initiate its entry into your formula.

To select an item from the auto-complete list, you can either:

- Use the cursor to scroll the list and click directly on an item.
- Navigate up and down the list using the Up and Down arrow keys.
- Continue typing enough text until the desired item is at the top of the list and selected.

As your selection changes within the list of available components, the text in the shaded area at the bottom will update to display a brief description of the current selection. Function and field descriptions will appear here.

Once you have selected the desired component, you can enter it into the formula by either typing Enter or double-clicking on it in the list.

■ **Tip** For the fastest entry method using auto-complete, skip the cursor and arrow keys. Instead, type just enough of the component’s name until it appears at the top of the list and then type Enter.

Using the Field Selection Panel

The *field selection panel*, shown in Figure 12-7, is used to locate and quickly insert a reference to a local or related fields.

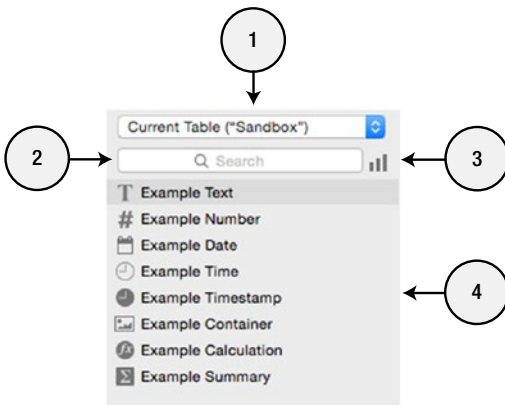


Figure 12-7. The field selection panel

The following controls are available to aid in the location and selection of a field:

1. *Table occurrence menu* — The pop-up menu at the top will default to the current context's table. For a calculation field, this means the table containing the field. When working with formulas for script steps or other interface components, this will be the table of the layout in the current window.
2. *Filter search field* — This text area is used to enter a search value that will narrow the list of fields.
3. *Field sort menu* — Click this icon to open a pop-up menu of sort options for the list of fields below. The available choices are the same as the sort options in the Fields tab of the Manage Database dialog:
 - a. Creation order
 - b. Field Name
 - c. Field Type
 - d. Custom Order
4. *Field list* — This list displays the fields for the selected table occurrence that match any search filter criteria entered, sorted in the order indicated by the sort menu.

Once a field is selected, double-click it to add it to the formula at the insertion point.

Handling Empty Fields

At the bottom of the Specify Calculation dialog is a check box labeled Don't evaluate option when fields empty. Enabling this box instructs FileMaker that the formula should not be evaluated when all the fields referenced within are empty. This is an easy way to eliminate a little unnecessary processing for instances where the result in such a case would be an empty value anyway. However, if the formula has a condition that will return a constant or other value when all the referenced fields are empty, be sure to *uncheck* this box.

Using the Function Selection Panel

Locating, selecting, and inserting a function through the Specify Calculation dialog interface is done using the *function selection panel*, shown in Figure 12-8. This panel contains a list of every built-in function, custom functions that you have added, and functions from any external plug-ins that are installed.

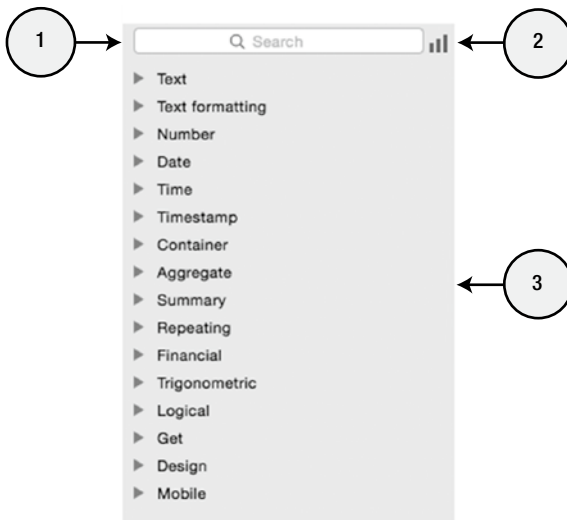


Figure 12-8. The function selection panel

The following controls are available to aid in the location and selection of a function:

1. *Filter search field* — This text area is used to enter a search value that will narrow the list of functions.
2. *Function option menu* — Click this icon to open a pop-up menu of options for the list of functions below. There are choices:
 - Show and organize All functions by name.
 - Show and organize All functions by type.
 - Show one selected category of functions sorted by name.
3. *Function list* — This list displays either all installed functions or those matching the sort/category menu selection.

Once the function is selected in the list, double-click it to add it to the formula at the insertion point. For a complete reference list of all available built-in functions, see Chapters 13-24.

Creating Repeating Calculation Fields

A calculation field can be defined to be repeating by entering a number in the Number of repetitions text area at the bottom of the Specify Calculation dialog. A repeating calculation works like a repeating entry field (Chapter 8). However, since returning a repeating calculated result is slightly more complex than simply allowing data entry in multiple repetitions of a field, there are a few different techniques and built-in functions that are crucial to consider when using this feature.

Using Repeating Fields in a Repeating Calculation

The most straightforward situation for a repeating calculation is to manipulate repeating fields where all fields involved are configured with the same number of repetitions. The example illustrated in Figure 12-9 involves two entry fields containing numbers that are added into a sum within a calculation field. Each field is defined with three repetitions.

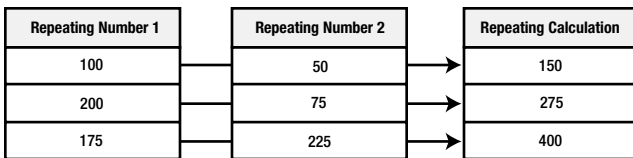


Figure 12-9. An illustration of a simple repeating calculation, manipulating values in other repeating fields

In this example, the calculation field’s formula would be defined to add the numbers together:

Repeating Number 1 + Repeating Number 2

This formula will be automatically applied to each set of repetitions, and the resulting value for each is placed into the corresponding repetition in the calculation field.

Using Non-Repeating Fields in a Repeating Calculation

When *non-repeating fields* are used in a *repeating calculation*, the formula must use the Extend function (Chapter 19) to extend the non-repeating field out to match the number of repetitions required to work in the repeating calculation. The example illustrated in Figure 12-10 involves one single-repetition entry field containing a single number, another entry field with three repetitions, and a calculation field with three repetitions.

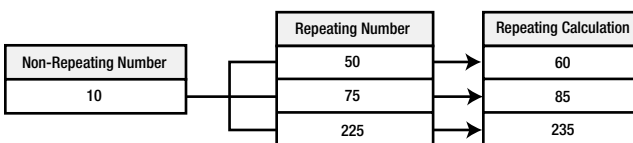


Figure 12-10. An illustration of a repeating calculation manipulating a mixture of repeating and non-repeating fields

For the formula to successfully add the value in the non-repeating field to each value in the repeating field and generate three results, one in each repetition, the calculation formula would be defined as follows:

```
Extend ( Non-Repeating Number ) + Repeating Number
```

This essentially duplicates the single value in the non-repeating field value into three separate virtual repetitions, each populated with the same value, as if it were a repeating the same number of times as the other field(s) involved in the calculation.

Other Repetition Functions

Other built-in functions that are useful in both *repeating* and *non-repeating* calculations include:

- `Get (ActiveRepetitionNumber)` — This function returns a number indicating which repetition of the current field has the cursor focus (Chapter 22).
- `Get (CalculationRepetitionNumber)` — This function returns a number indicating which repetition of the repeating field is currently being evaluated, allowing a calculation field formula to know which repetition of its own structure is currently being evaluated (Chapter 22).
- `Get Repetition` — This function returns the value from a specified repetition of a specified field (Chapter 19).

Using Space to Your Advantage

FileMaker enforces syntax rules to confirm the integrity of a formula. For example, literal text must be enclosed within quotes and all parentheses must be properly closed. However, extra spaces, tabs, and carriage returns, outside of quoted text, can be added throughout a formula to spread out the code and make it easier to read without affecting the syntax of the formula.

Throughout the examples in this book, the space-expanding techniques described here are used to make the code more readable in print. However, these practices can and should be followed when writing actual code to make it more readable and easier to edit.

Adding a Space Between Items

Adding a space between operators, function names, and delimiters will spread everything out horizontally and make the code much easier to read.

Each of the following statements exhibit some level of crowding, which makes them difficult to read and visually evaluate.

```
(150+50)/25
$FirstName&$Space&$LastName
Get(CurrentDate)
```

With space added, these will function the same, but are now easier to read.

```
( 150 + 50 ) / 25
$FirstName & $Space & $LastName
Get ( CurrentDate )
```

Expanding Complex Statements

Some functions quickly become difficult to follow once they reach a certain size. The `Case`, `Let`, and `Substitute` functions are especially prone to this. Adding horizontal space helps but it's not enough. When statements grow and include lengthy and complex expressions, consider using tabs and carriage returns to help spread them out onto multiple lines.

To illustrate this point, consider the following hypothetical formula that will accept a field or static value in place of the `Input` placeholder and return that value with the number of characters it contains in parentheses. For example, if `Input = "Hello World"` the result will be `"Hello World (11)"` with 11 being the count of characters in the input. To accomplish this, the formula uses a `Case` statement nested within a `Let` statement. With no extra spaces, it is difficult to read.

```
Let([data=Input];data&Case(data≠"";" (" & Length(data)&")))
```

Adding spaces between components helps a little, but even this short example is still somewhat visually hard to follow.

```
Let ( [ data = Input ] ; data & Case ( data ≠ "" ; " (" & Length ( data ) & ")" ) )
```

In addition to the spaces between components, by adding tabs and carriage returns, the formula is expanded in away that separates and highlights the inner portions of the two statements.

```
Let ( [
    data = Input
] ;
    data &
    Case (
        data ≠ "" ; " (" & Length ( data ) & ")"
    )
)
```

The benefit of this approach becomes even more apparent when the formula begins to grow. Just imagine the example below expressed as a continuous string *without* spaces, tabs, or returns.

```
Let ( [
    data1 = input1 ;
    data2 = input2 ;
    data3 = input3 ;
    data4 = input4 ;
    data5 = input5 ;
] ;
    Case (
        data1 ≠ "" ; data1 & " (" & Length ( data1 ) & ")" ;
        data2 ≠ "" ; data2 & " (" & Length ( data2 ) & ")" ;
        data3 ≠ "" ; data3 & " (" & Length ( data3 ) & ")" ;
        data4 ≠ "" ; data4 & " (" & Length ( data4 ) & ")" ;
        data5 ≠ "" ; data5 & " (" & Length ( data5 ) & ")" ;
    )
)
```

The Substitute function can benefit greatly from indentation and carriage returns when it contains numerous find-replace value pairs. This example shows a formula for a hypothetical statement where each of five find-replace value pairs is on its own line, making the overall statement easy to view and edit:

```
Substitute (
    input ;
    [ find1 ; replace1 ] ;
    [ find2 ; replace2 ] ;
    [ find3 ; replace3 ] ;
    [ find4 ; replace4 ] ;
    [ find5 ; replace5 ]
)
```

Managing Formulas That Exceed 30,000 Characters

While it is unlikely to happen, the more complex your formulas become the greater the odds are of bumping into FileMaker’s limit of 30,000 characters. As this happens, many will avoid adding additional spaces, tabs, and carriage returns to save space. However, there are techniques that can be employed to manage space *without* requiring code to become condensed and less readable. These include:

- Split up the formula into multiple variables or fields.
- Use the Let command to condense lengthy and repeated field references into short variables. For example, Contact::Contact Name First can be placed into a variable called n.first and used repeatedly, with each subsequent instance using up less space.
- Offload some of the formula to a custom function (Chapter 25).

Adding Calculations to the Learn FileMaker File

Maintaining the Sandbox table for calculation experimentation, let’s add a couple of simple calculation fields to the Company and Contact tables.

Company Contact Count

In the Company table, add a new calculation field called Company Contact Count and define the formula to get a count of the related contact records, following these steps:

1. On the Fields tab of the Manage Database dialog, select the Company table.
2. Enter the name for the new field, select Calculation as the type, and click the Create button.
3. Select the Company | Contact table occurrence from the pop-up menu in the field selection panel on the left side of the dialog.
4. Begin typing “Count” and select the Count function from the auto-complete list.

5. Double-click on the Record ID field from the Company | Contact table occurrence.
6. Make sure the Calculation result type is Number.
7. If the formula reads Count (Company | Contact::Record ID) click OK to close the Specify Calculation dialog.

Contact Address Label

In the Contact table, add a new calculation field called Contact Address Label and define the formula to contain the address formatted as a single text value. The formula will look like this:

```
Contact::Address Street & "¶" &  
Contact::Address City & ", " & Contact::Address State & " " & Contact::Address Zip & "¶" &  
Contact::Address Country
```

■ **Note** The examples provided in this chapter are relatively simple. More complex calculations will appear throughout the remaining chapters.

Summary

In this chapter, we discussed the basics of writing formulas.

CHAPTER 13



Text Functions

Text functions are used to analyze, convert, combine, extract, and create text strings.

Text Function Reference

FileMaker provides the following text functions:

- Char
- Code
- Exact
- Filter
- FilterValues
- Furigana
- GetAsCSS
- GetAsDate
- GetAsNumber
- GetAsSVG
- GetAsText
- GetAsTime
- GetAsTimestamp
- GetAsURLEncoded
- GetValue
- Hiragana
- KanaHankaku
- KanaZenkaku
- KanjiNumeral
- Katakana
- Left

- LeftValues
- LeftWords
- Length
- Lower
- Middle
- MiddleValues
- MiddleWords
- NumToJText
- PatternCount
- Position
- Proper
- Quote
- Replace
- Right
- RightValues
- RightWords
- RomanHankaku
- Roman Zenkaku
- Serial Increment
- SortValues
- Substitute
- Trim
- TrimAll
- UniqueValues
- Upper
- ValueCount
- WordCount

Char

The Char function converts a number into its equivalent Unicode text character.

Format	Char (number)
Parameters	number = a number between 1 and 99,999.
Result	The Unicode text character for number provided.
More Info	For a complete list of available Unicode numbers visit: http://www.ssec.wisc.edu/~tomw/java/unicode.html .

These examples show the function operating on different input:

```
Char ( 65 )    // result = "A"
Char ( 97 )    // result = "a"
Char ( 0 )     // result = "" (an empty string)
Char ( 9 )     // result = " " (a tab character)
```

Code

The Code function converts a text value into an equivalent Unicode number.

Format	Code (text)
Parameters	text = a text value.
Result	The number for the Unicode text character provided.

These examples show the function operating on different input:

```
Code ( "A" )   // result = 65
Code ( "a" )   // result = 97
Code ( "" )    // result = 0
```

Exact

The Exact function compares two values and determines if they are the same or not. The function is case sensitive and looks for *precisely* the same characters unlike a simple comparison with the `is equal` to operator.

Format	Exact (originalText ; comparisonText)
Parameters	originalText = any data type. comparisonText = any data type.
Result	A Boolean as follows: 1 = the two values are an exact match. 0 = the two values are not an exact match.

Here are some examples of comparing text constants:

```
Exact ( "A" ; "B" )           // result = 0
Exact ( "A" ; "A" )           // result = 1
Exact ( "Hello World" ; "World Hello" ) // result = 0
```

To see the how this function differs from the `is equal` to operator when comparing values of different cases:

```
"a" = "A"                       // result = 1
Exact ( "a" ; "A" )              // result = 0
```

Comparing text constants to field contents, these examples assumes a record in a Contact database with two fields: State containing a value of "NY" and "Age" containing a value of 21.

```
Exact ( Contact::State ; Age ) // result = 0
Exact ( Contact::State ; "PA" ) // result = 0
Exact ( Contact::State ; "NY" ) // result = 1
Exact ( Contact::Age ; 21 ) // result = 1
Exact ( Contact::State & Contact::Age ; "NY21" ) // result = 1
```

Examples of comparing other data types, including container fields:

```
Exact ( 150 ; 150 ) // result = 1
Exact ( "1/15/2017" ; $LastDate ) // result = 1
Exact ( Photo::Image1 ; Photo::Image2 ) // result = 0
```

Filter

The Filter function removes any unwanted characters from a text value.

Format	Filter (textToFilter ; filterText)
Parameters	textToFilter = a text value to be filtered. filterText = a string of the characters that are allowed to remain; any character not listed will be removed from textToFilter.
Result	The text from textToFilter without any unwanted characters not specified in filterText.

Here are some examples of removing any non-numeric characters:

```
Filter ( "Hello World" ; "1234567890" ) // result = ""
Filter ( "1 Hello 2 World 3" ; "1234567890" ) // result = "123"
Filter ( "Las Vegas, NV 89101" ; "1234567890" ) // result = "89101"
Filter ( "(212) 555-1234" ; "1234567890" ) // result = "2125551234"
```

Here is an example of removing anything besides a dollar amount:

```
Filter ( "The total purchase is $5,000.00" ; "1234567890$.,") // result = "$5,000"
```

FilterValues

The FilterValues function removes out any unwanted values from a list of values.

Format	FilterValues (textToFilter ; filterValues)
Parameters	textToFilter = a carriage return-delimited list of text values to be filtered. filterValues = a carriage return-delimited list of text values that are allowable; any values not listed will be removed from textToFilter.
Result	The text from textToFilter without any unwanted values not specified in filterValues.

Here are some examples of removing a list of constants:

```
FilterValues ( "One¶Two¶Three¶Four" ; "Two¶Four" ) // result = "Two¶Four"
FilterValues ( "NY¶IN¶OH¶PA" ; "PA¶NY" ) // result = "NY¶PA"
```

Furigana

The Furigana function converts Japanese text into one of five different alternatives. For more information on Japanese functions, see FileMaker's documentation.

GetAsCSS

The GetAsCSS function converts formatted text into the Cascading Style Sheet (CSS) format, thereby preserving the font, font size, font color and font style attributes.

Format	GetAsCSS (text)
Parameters	text = a text value.
Result	The text provided with all formatting information converted into CSS tags.

Converting the text from a field, this example assumes a record in a Contact table with a field named First Name that contains the name "John" where the font is "Arial," the font size is 18 pt, the font style is bold, and the font color is red:

```
GetAsCSS ( Contact::First Name )
// result = <span style="font-family: 'Arial';font-size: 18px; color: #FF2712;font-weight:bold;" >John</span>
```

GetAsDate

The GetAsDate function converts a text-based date into a date object to ensure proper results when compared to or sorted with other dates.

Format	GetAsDate (text)
Parameters	text = a date in a text string with a format consistent with the operating system date format or a number indicating the number of days that have passed since January 1, 0001; works up to 1460970 (December 31, 4000).
Result	The text provided formatted as a date object.

Dates can contain leading zeros or not, as shown in these examples:

```
GetAsDate ( "1/5/2017" ) // result = 1/5/2017
GetAsDate ( "01/05/2017" ) // result = 1/5/2017
```

Dates should always contain a four-digit year to ensure correct handling. However, this function will handle dates with two-digit years with pretty good results if the intended four-digit year is either within the next 30 years or the preceding 70 years from the current date. Outside of that range you may get unexpected results if you don't use four-digit years. These examples assume a current date of January 5, 2017:

```
GetAsDate ( "1/5/17" )           // result = 1/5/2017
GetAsDate ( "1/5/95" )           // result = 1/5/1995
GetAsDate ( "1/5/50" )           // result = 1/5/1950
```

When a number is provided, it is used to calculate the number of days that have passed since January 1, 0001. For example:

```
GetAsDate ( 736334 )             // result = 1/5/2017
```

GetAsNumber

The `GetAsNumber` function converts a text-based number value into a number object to ensure proper results when compared to or sorted with other numbers.

Format	<code>GetAsNumber (text)</code>
Parameters	text = a number in a text string.
Result	The text provided formatted as a number.

When providing numbers as text, they are converted back into numbers. For example:

```
GetAsNumber ( "1234" )           // result = 1234
GetAsNumber ( "015" )            // result = 15
GetAsNumber ( "13.75" )          // result = 13.75
```

Here are some examples of converting values that contain non-numeric characters:

```
GetAsNumber ( "$25.09" )         // result = 25.09
GetAsNumber ( "He ran 9.75 miles." ) // result = 9.75
```

In some cases, relying on automatic filtering of non-numeric characters might not provide in a desirable result. For example:

```
GetAsNumber ( "3 men ran 9.75 miles." ) // result = 39.75
```

The examples below assume a value of "03" is contained in a *text* field called *Qty*. They show the importance of converting text into numbers when comparing to other numeric values. Because of the leading zero on the text-based number in the field, the two values don't appear to be the same (in the first example) until the text is converted to a number (in the second example).

```
( 3 = Qty )                       // result = 0
( 3 = GetAsNumber ( Qty ) )       // result = 1
```

The examples below assume a value of “20” is contained in a *text* field called *Qty*. Since it is a text value, 20 will appear as a smaller value than 3 because the text is compared character by character rather than the entire value compared as a number. Once converted to text, as shown in the second example, it evaluates correctly.

```
( 3 > Qty )                // result = 1
( 3 > GetAsNumber ( Qty ) ) // result = 0
```

GetAsSVG

The `GetAsSVG` function converts text to the Scalable Vector Graphics (SVG) format, which supports more text formats than HTML so it may represent text more accurately in some cases.

Format	<code>GetAsSVG (text)</code>
Parameters	<code>text</code> = any text.
Result	The text provided with all formatting information converted into SVG tags.

The example below assumes a record in a `Contact` table with a field named `First Name` that contains the name “John” where the font is “Arial,” the font size is 18 pt, the font style is bold, and the font color is red.

```
GetAsSVG ( Contact::First Name )
// result =
<stylelist>
<style#0>"font-family: 'Arial';font-size: 18px;color: #FF2712;font-weight: bold;",
  begin: 1, end: 4</style>
</stylelist>
<data>
<span style="0">John</span>
</data>
```

GetAsText

The `GetAsText` function converts a number, date, time, or timestamp expression or any field containing any data type (including containers) into text.

Format	<code>GetAsText (data)</code>
Parameters	<code>data</code> = any text-based data (non-container).
Result	The data provided formatted as text.

No matter the type of data, it will be converted into a text string as seen in these examples:

```
GetAsText ( 58.75 )                // result = "58.75"
GetAsText ( 05:15:00 )            // result = "5:15:00"
GetAsText ( 6/30/2016 5:20:49 PM ) // result = "6/30/2016 5:20:49 PM"
```

A container field that contains a reference to an image file will return the metadata in the form of name, size, and path of the file as shown in this example:

```
GetAsText ( Contact::Image )
// result =
    size:191,175
    image:Mark Munro.jpg
    imagemac:/Macintosh HD/Users/admin/Desktop/Mark Munro.jpg
```

If the field contains an image rather than just a reference, the function returns the name of the image file as shown in this example:

```
GetAsText ( Contact::Image )    // result = Mark Munro.jpg
```

GetAsTime

The `GetAsTime` function converts a text-based time or timestamp value into a time object to ensure proper results when compared to or sorted with other times.

Format	<code>GetAsTime (text)</code>
Parameters	<code>text</code> = a time or timestamp in a text string with a format consistent with the operating system date.
Result	The text provided formatted as time.

Only text strings containing time values will be properly converted. For example:

```
GetAsTime ( "5:15:00" )    // result = 5:15:00
GetAsTime ( "Hello World" ) // result = ?
```

GetAsTimestamp

The `GetAsTimestamp` function converts a text-based time or timestamp value into a time object to ensure proper results when compared to or sorted with other times.

Format	<code>GetAsTimestamp (text)</code>
Parameters	<code>text</code> = a date, time or timestamp as a text string with a format consistent with the operating system date and time, or a number indicating the number of seconds that have passed since January 1, 0001.
Result	The text provided formatted as a timestamp.

When the text contains full timestamp information, it is converted into the proper data format as shown in this example:

```
GetAsTimestamp ( "1/5/2017 5:15:00" )    // result = 1/5/2017 5:15:00
```

If a date is provided without time information, the function returns a timestamp for midnight on the specified date, as shown here:

```
GetAsTimestamp ( "1/1/2017" ) // result = 1/1/2017 12:00 AM
```

If only a number is provided, the function returns timestamp for that number of seconds since 1/1/0001, as shown here:

```
GetAsTimestamp ( 100000 ) // result = 1/2/0001 3:46:40 AM
```

GetAsURLEncoded

The GetAsURLEncoded function encodes text for use in a Uniform Resource Locator (URL) as follows:

- All style information is removed from the text.
- The characters are converted to UTF-8 format.
- Any characters that are not a letter or digit, or are in the upper ASCII range are *percent encoded*, meaning they are converted to a percent symbol followed by the hexadecimal value of the character (%HH).

Format	GetAsURLEncoded (text)
Parameters	text = a text value.
Result	The text provided encoded for use in a URL.
More Info	For more information on URL encoding, visit: http://www.w3.org .

These examples show how some characters will be percent encoded:

```
GetAsURLEncoded ( "Hello World" ) // result = "Hello%20World"
GetAsURLEncoded ( "10% Surcharge" ) // result "10%25%20Surcharge"
```

GetValue

The GetValue function extracts a value from a list of values.

Format	GetValue (listOfValues ; valueNumber)
Parameters	listOfValues = a carriage return-delimited list of text values. valueNumber = a number specifying the desired value to extract from the list.
Result	The text of the specified value from the list provided.

Some examples showing the specified value will be returned:

```
GetValue ( "John¶Jane¶Jim¶Joe¶" ; 3 ) // result = "Jim"
GetValue ( "159¶245¶396¶721¶" ; 2 ) // result = 245
```

If the specified value is greater than the number of values in the list, an empty string will be returned, as shown in this example:

```
GetValue ( "John¶Jane¶Jim¶Joe¶" ; 18 ) // result = ""
```

Hiragana

The Hiragana function converts Katakana (Hankaku and Zenkaku) in text to Hiragana. For more information on Japanese functions, see FileMaker's documentation.

KanaHankaku

The KanaHankaku function converts Zenkaku Katakana in text to Hankaku Katakana. For more information on Japanese functions, see FileMaker's documentation.

KanaZenkaku

The KanaZenkaku function converts Hankaku Katakana in text to Zenkaku Katakana. For more information on Japanese functions, see FileMaker's documentation.

KanjiNumeral

The KanjiNumeral function converts Arabic numerals to Japanese Kanji numerals. For more information on Japanese functions, see FileMaker's documentation.

Katakana

The Katakana function converts Hiragana in text to Zenkaku Katakana. For more information on Japanese functions, see FileMaker's documentation.

Left

The Left function extracts a specified number of characters from the provided text starting from the first character on the left.

Format	Left (text ; numberOfCharacters)
Parameters	text = any text from which to extract characters. numberOfCharacters = a number indicating how many characters to extract.
Result	A text string containing the specified number of characters from the text provided.

Here is an example showing the number of characters specified being returned:

```
Left ( "Hello World" ; 8 ) // result = "Hello Wo"
```

When the number specified is greater than the number of characters in the text provided, the function will return the original value, as shown in this example:

```
Left ( "Hello World" ; 28 )      // result = "Hello World"
```

LeftValues

The `LeftValues` function extracts a specified number of values from a list of values starting from the first value.

Format	<code>LeftValues (text ; numberOfValues)</code>
Parameters	text = a carriage return-delimited list of text values. numberOfValues = a number indicating how many values to extract.
Result	The specified number of carriage return-delimited list of text values from the text provided.

Some examples showing the number of values specified extracted:

```
LeftValues ( "John¶Jane¶Jim¶Joe¶" ; 2 )      // result = "John¶Jane"
LeftValues ( "159¶245¶396¶721¶" ; 3 )      // result = "159¶245¶396"
```

When the number specified exceeds the available values, the entire text provided will be returned, as shown in this example:

```
LeftValues ( "John¶Jane¶Jim¶Joe¶" ; 10 )    // result = " John¶Jane¶Jim¶Joe¶"
```

LeftWords

The `LeftWords` function extracts a specified number of words from the text provided, starting from the first word on the left.

Format	<code>LeftWords (text ; numberOfWords)</code>
Parameters	text = a text value from which words will be extracted. numberOfWords = a number indicating how many words to extract.
Result	A text string containing the requested number of words from the text provided.

The specified number of words will be returned with any punctuation that falls within the range, as shown here:

```
LeftWords ( "Hello World. How are you?" ; 3 ) // result = "Hello World. How"
```

When punctuation falls at the end of the range, will *not* be included, as shown here:

```
LeftWords ( "Hello World. How are you?" ; 2 ) // result = "Hello World"
```

Length

The `Length` function counts the number of characters in a piece of text.

Format	<code>Length (text)</code>
Parameters	<code>text</code> = any text, number, date, time, timestamp or container value.
Result	A number indicating the count of characters in the text provided.

Any text will be counted as part of the length, as shown here:

```
Length ( "Hello World" )           // result = 11
```

This example shows that odd results can be expected when providing non-text values:

```
Length ( 1/15/2017 )              // result = 17
```

Lower

The `Lower` function transforms a text string into all lowercase.

Format	<code>Lower (text)</code>
Parameters	<code>text</code> = a text value.
Result	The text provided converted to all lowercase.

These examples show the function operating on different input:

```
Lower ( "Hello World" )           // result = "hello world"
Lower ( "THIS IS SCREAMING" )    // result = "this is screaming"
```

Middle

The `Middle` function extracts a specified number of characters from a specified position within a text string.

Format	<code>Middle (text ; start ; numberOfCharacters)</code>
Parameters	<code>text</code> = a text value from which to extract characters. <code>start</code> = a number indicating the character position from which to start the extraction. <code>numberOfCharacters</code> = a number indicating the quantity of characters to extract from the starting position.
Result	A text string containing only the requested characters from the specified location within the text provided.

These examples show the function operating on different input:

```
Middle ( "Hello World" ; 7 ; 5 )   // result = "World"
Middle ( 123456789 ; 4 ; 3 )      // result = 456
```


MiddleValues

The `MiddleValues` function extracts a specified number of values from a list of values starting from a specified value.

Format	<code>MiddleValues (text ; startingValue ; numberOfValues)</code>
Parameters	<p><code>text</code> = a text value containing a carriage return-delimited list of text values.</p> <p><code>startingValue</code> = a number indicating the value from which to start the extraction.</p> <p><code>numberOfValues</code> = a number indicating how many values to extract from the starting value.</p>
Result	The specified number of carriage return-delimited list of text values from the text provided.

Here is an example showing the extraction of two values from the middle of a list of values:

```
MiddleValues ( "John¶Jane¶Jim¶Joe¶" ; 2 ; 2 ) // result = "Jane¶Jim¶"
```

MiddleWords

The `MiddleWords` function extracts a specified number of words from the text provided, starting from a specified word.

Format	<code>MiddleWords (text ; startingWord ; numberOfWords)</code>
Parameters	<p><code>text</code> = a text value.</p> <p><code>startingWord</code> = a number indicating the word from which to start the extraction.</p> <p><code>numberOfWords</code> = a number indicating how many words to extract from the starting word.</p>
Result	A text string containing the requested number of words from the specified location within the text provided.

Here is an example of extracting two words from the middle of some text:

```
MiddleWords ( "Hello World. How are you?" ; 3 ; 2 ) // result = "How are"
```

NumToJText

The `NumToJText` function converts Arabic numerals to Japanese text. For more information on Japanese functions, see FileMaker's documentation.

PatternCount

The `PatternCount` function counts the number of times a specific pattern of text appears in the text provided.

Format	<code>PatternCount (text ; searchString)</code>
Parameters	<p><code>text</code> = a text value in which a pattern will be detected.</p> <p><code>searchString</code> = a text value that will be searched for in the text provided.</p>
Result	A number indicating the number of times the <code>searchString</code> appears within the text provided.

These examples show the function operating on different input:

```
PatternCount ( "Hello World. How is your world today?" ; "world" )
// result = 2
```

```
PatternCount ( Invoice::Tax Exempt ; "Yes" )
// result = 0 (if the Tax Exempt field does not contain the word "Yes")
```

Remember that `PatternCount` looks for matches *anywhere* in the text provided, including as part of a word or paragraph. When checking for the presence of a *value* in a list of carriage return-delimited values, it is important to avoid partial matches like those returned by the `PatternCount` function. For example, the following code will find *two* matches for “age” in the text provided. However, if we are looking for a match to “age” as a value — as a single paragraph — those results are faulty.

```
PatternCount ( "Guage¶Page" ; "age" ) // result = 2
```

To avoid this, one *may* use the cumbersome technique of including paragraph returns to detect a value match, however, this requires that both the source text and the pattern we are searching start and end with a return to catch the value at the beginning or end as well as the middle. So, one may be tempted to approach the problem like this:

```
PatternCount ( "¶Guage¶Page¶" ; "¶age¶" ) // result = 0
```

Alternatively, by combining several functions, this formula will safely detect only a full *value* match without the need to worry about leading or trailing carriage returns:

```
Not IsEmpty ( FilterValues ( "Guage¶Page" ; "age" ) ) // result = 0
```

Position

The `Position` function finds the starting character position of a specified occurrence of some text in the provided text.

Format	<code>Position (text ; searchString ; start ; occurrence)</code>
Parameters	<p><code>text</code> = a text value in which a string will be located.</p> <p><code>searchString</code> = a text value containing the text that will be searched for in the text provided.</p> <p><code>start</code> = a number indicating the character position, counting from the left, where the search will begin.</p> <p><code>occurrence</code> = a number indicating the occurrence of a match of the search string after the starting position that should be used as the result.</p>
Result	A number indicating the starting character location of the matched occurrence within the text provided.

These examples show the function operating on different input:

```
Position ( "Where is Waldo today?" ; "Waldo" ; 1 ; 1 )
// result = 10

Position ( "Where is Waldo today?" ; "Waldo" ; 1 ; 2 )
// result = 0

Position ( "Waldo is looking for Waldo?" ; "Waldo" ; 1 ; 2 )
// result = 22
```

Proper

The Proper function transforms the text provided into a “proper” case, with only the first letter of each word capitalized.

Format	Proper (text)
Parameters	text = a text value that will be transformed to proper case.
Result	The text provided converted to proper case.

These examples show the function operating on different input:

```
Proper ( "hello world" ) // result = "Hello World"
Proper ( "THIS IS SCREAMING" ) // result = "This Is Screaming"
```

Quote

The Quote function encloses a piece of text in quotation marks.

Format	Quote (text)
Parameters	text = a text value that should be wrapped in quotation marks.
Result	The text provided wrapped in quotation marks.

Here is an example of building a string with some quoted text in the middle:

```
"Say " & Quote ( "Hello World" ) & " now."
// result = Say "Hello World" now.
```

Replace

The `Replace` function replaces a specified number of characters at a certain position within a piece of text with some replacement text.

Format	<code>Replace (text ; start ; numberOfCharacters ; replacementText)</code>
Parameters	<p><code>text</code> = a text value in which some characters will be replaced.</p> <p><code>start</code> = a number indicating the character position where the replacement should begin.</p> <p><code>numberOfCharacters</code> = a number indicating the number of characters that should be replaced from the start position.</p> <p><code>replacementText</code> = a text value that will replace the original text; it does not need to be the same number of characters as that which it is replacing.</p>
Result	The text provided with the requested replacement of characters performed.

Here is an example of replacing some text in a string:

```
Replace ( "Hello Something" ; 7 ; 9 ; "World" ) // result = "Hello World"
```

Right

The `Right` function extracts a specified number of characters from the provided text starting from the last character on the right.

Format	<code>Right (text ; numberOfCharacters)</code>
Parameters	<p><code>text</code> = the text from which to extract characters.</p> <p><code>numberOfCharacters</code> = a number indicating how many characters to extract.</p>
Result	A text string containing the requested number of characters from the text provided.

Here is an example of extracting text from the right side of a string:

```
Right ( "Hello World" ; 8 ) // result = "lo World"
```

RightValues

The `RightValues` function extracts a specified number of values from a list of values starting from the last value.

Format	<code>RightValues (text ; numberOfValues)</code>
Parameters	<p><code>text</code> = a carriage return-delimited list of text values.</p> <p><code>numberOfValues</code> = a number indicating how many values to extract.</p>
Result	The specified number of carriage return-delimited list of text values from the text provided.

Here are some examples of extracting values from the right (bottom) of a list:

```
RightValues ( "John¶Jane¶Jim¶Joe¶" ; 2 ) // result = "Jim¶Joe"
RightValues ( "159¶245¶396¶721¶" ; 3 ) // result = "245¶396¶721"
```

RightWords

The `RightWords` function extracts a specified number of words from the text provided, starting from the last word on the right.

Format	<code>RightWords (text ; numberOfWords)</code>
Parameters	text = a text value from which words will be extracted. numberOfWords = a number indicating how many words to extract.
Result	A text string containing the requested number of words from the text provided.

Here is an example of extracting words from the right side of a string:

```
RightWords ( "Hello World. How are you?" ; 3 ) // result = "How are you"
```

RomanHankaku

The `RomanHankaku` function converts Zenkaku alphanumeric and symbols in text to Hankaku alphanumeric symbols. For more information on Japanese functions, see FileMaker's documentation.

RomanZenkaku

The `RomanZenkaku` function converts Hankaku alphanumeric and symbols in text to Zenkaku alphanumeric and symbols. For more information on Japanese functions, see FileMaker's documentation.

SerialIncrement

The `SerialIncrement` function detects the first numeric value from the right within a text string and increments it by a specified number.

Format	<code>SerialIncrement (text ; incrementBy)</code>
Parameters	text = a text value containing a number. incrementBy = a number indicating how much to increment the text by.
Result	The text provided with the final number incremented by one.

These examples show how values in values are incremented:

```
SerialIncrement ( "Hello " ; 1 ) // result = "Hello 1"
SerialIncrement ( "Hello 50" ; 1 ) // result = "Hello 51"
SerialIncrement ( "Hello50" ; 1 ) // result = "Hello51"
SerialIncrement ( "Hello 50.9" ; 1 ) // result = "Hello 50.10"
SerialIncrement ( 1/5/2017 ; 1 ) // result = 1/5/2018
```

SortValues

The SortValues function will rearrange the order of carriage return-delimited values.

Format	To sort values as text using the file's default locale, only a single parameter is required: SortValues (values) Optional parameters allow specification of data type and locale: SortValues (values ; datatype ; locale)
Parameters	values = a list of carriage return-delimited text values to be sorted. datatype = (optional) a number indicating the data type of the values. A positive number results in an ascending sort, a negative value results in a descending sort. locale = (optional) a text value indicating the locale that should control the sort.
Options for Datatype	The options for the datatype parameter are: 1 — sort as text (default when no datatype is specified) 2 — sort as numeric 3 — sort as date 4 — sort as time 5 — sort as timestamp
Options for Locale	The options for the locale parameter are: Arabic, Bengali, Bulgarian, Catalan, Catalog, Chinese, Chinese_Stroke, Croatian, Czech, Danish, Dutch, English, Estonian, Finnish, Finnish_Custom, French, German, German_Dictionary, Greek, Greek_Mixed, Gujarati, Hebrew, Hindi, Hugarian, Icelandic, Italian, Japanese, Kannada, Korean, Latvian, Lithuanian, Malayalam, Marathi, Norwegian, Panjabi, Persian, Polish, Portuguese, Romanian, Russian, Serbian, Slovak, Slovenian, Spanish, Spanish_Traditional, Swedish, Swedish_Custom, Tamil, Telugu, Thai, Turkish, Ukrainian, Unicode_Raw, Unicode_Standard and Vietnamese. These only need to be specified when the language of or sorting methodology for the values differs from that of the operating system upon which the database file was created.
Result	The carriage return-delimited list of values reordered based on the parameters specified.

This example sorts text values, using the file's default locale:

```
SortValues ( "New York¶Illinois¶Pennsylvania¶California" )
// result =
    California
    Illinois
    New York
    Pennsylvania
```

This example shows numbers sorting as text values when no datatype is specified:

```
SortValues ( "100¶10¶200¶20" )
// result =
    10
    100
    20
    200
```

This example shows numbers sorting as numeric values when a datatype is specified:

```
SortValues ( "100¶10¶200¶20" ; 2 )
// result =
    10
    20
    100
    200
```

To change the sort order from ascending to descending results, the datatype should be a negative value:

```
SortValues ( "100¶10¶200¶20" ; -2 )
// result =
    200
    100
    20
    10
```

Substitute

The Substitute function replaces one or more text strings anywhere it appears within a piece of text with a different string.

Format	<p>When substituting a single search-replace pair, the function should be called with three parameters:</p> <pre>Substitute (text ; searchString ; replacementString)</pre> <p>Optionally, the function accepts multiple search-replace pairs, in any quantity, each contained within square brackets and separated by a semi-colon:</p> <pre>Substitute (text ; [searchString1 ; replacementString1] ; [searchString2 ; replacementString2] ; [searchString3 ; replacementString3])</pre>
Parameters	<p>text = a text value representing the text to be manipulated. searchString = a text value representing the text to be located and replaced. replacementString = a text value representing the text to be used to replace the searchString.</p>
Result	<p>The text provided with the specified replacements tasks performed.</p>

These examples demonstrate single search-replace pairs:

```
Substitute ( "One Two Four" ; "Four" ; "Three" )
// result = "One Two Three"
Substitute ( "Hello World? It is good to see you?" ; "?" ; "!" )
// result = "Hello World! It is good to see you!"
```

This example demonstrates multiple search-replace pairs:

```
Substitute ( "One Two Four Six" ; [ "Four" ; "Three" ] ; [ "Six" ; "Four" ] )
= "One Two Three Four"
```

In this example, the text in the “Name” field will be cleared of carriage returns and tabs:

```
Substitute ( Table::Name ; [ "¶" ; "" ] ; [ " " ; "" ] )
```

This example shows a method of clearing extra space throughout a piece of text. It contains three substitutions: quadruple space, tripled space, and double space, each replaced with a single space in succession to ensure that most extra spaces is removed:

```
Substitute ( Table::Text Field ; [ " " ; " " ] ; [ " " ; " " ] ; [ " " ; " " ] )
```

Trim

The Trim function removes all leading and trailing spaces from a piece of text. This is useful to correct data entry errors or when importing from an external data source that contains a fixed number of characters per field.

Format	Trim (text)
Parameters	text = a text value that may contain extraneous spaces.
Result	The text provided without leading or trailing spaces.

Here is an example of trimming excessive spaces from a string:

```
Trim ( " Hello World " ) // result = "Hello World"
```

TrimAll

The TrimAll function manages spaces in a text string based on various criteria. Many of its options are useful only for non-Roman languages like Japanese and are not specified here. For more information on Japanese functions, see FileMaker’s documentation.

Format	TrimAll (text ; trimSpace ; trimType)
Parameters	text = a text value. trimSpace = a Boolean indicating whether to remove full-width spaces from non-Roman; leave this false for Roman languages. trimType = a number from 0-3 indicating how spaces should be handled.
Result	The text provided with the specified spaces adjusted.

This example will remove all spaces between Roman characters while always leaving one space between Roman words:

```
TrimAll ( " Hello World " ; 0 ; 0 ) // result = "Hello World"
```

This example will remove all spaces between Roman characters:

```
TrimAll ( " Hello World " ; 0 ; 3 ) // result = "HelloWorld"
```


UniqueValues

The UniqueValues function returns a carriage return-delimited list of values with duplicates removed.

Format	To manipulate values as text using the file's default locale, only a single parameter is required: UniqueValues (values) Optional parameters allow specification of data type and/or locale: UniqueValues (values ; datatype ; locale)
Parameters	values = a text value that provides a list of carriage return-delimited values to be processed. datatype = (optional) a number indicating the data type of the values. locale = (optional) a text expression indicating the locale that should control the sort.
Result	The original carriage return-delimited list of values without duplicates.

Here is an example of extracting a list of unique values from a list:

```
UniqueValues ( "15¶125¶10¶125¶10.0" ; 2 )
// result =
    15
    125
    10
```

■ **Note** See SortValues for a list of available values for the datatype and locale parameters.

Upper

The Upper function transforms the text provided into uppercase.

Format	Proper (text)
Parameters	text = a text value that will be transformed to uppercase.
Result	The text provided converted to all uppercase.

These examples show the function operating on different inputs:

```
Upper ( "Hello World" )           // result = "HELLO WORLD"
Upper ( "this is screaming" )    // result = "THIS IS SCREAMING"
```

ValueCount

The ValueCount function counts the number of values in a list of values.

Format	ValueCount (text)
Parameters	text = a carriage return-delimited list of text values.
Result	A number indicating the count of values detected within the text provided.

Here is an example of counting values in a list:

```
ValueCount ( "John¶Jane¶Jim¶Joe¶" ) // result = 4
```

WordCount

The `WordCount` function counts the number of words in a piece of text.

Format	<code>WordCount (text)</code>
Parameters	<code>text</code> = a text value.
Result	A number indicating the number of words detected within the text provided.

Here is an example of counting the number of words in a string:

```
WordCount ( "Hello World" ) // result = 2
```

Summary

In this chapter, we explored all the built-in functions for manipulating text strings.

CHAPTER 14



Text Functions (JSON)

JSON functions were introduced in FileMaker 16 as a subset of Text Functions. These are used to create and manipulate JSON objects that contain information stored in a structured text format.

Introducing JSON

JavaScript Object Notation (JSON) provides a lightweight data-interchange format that is both human readable and easy for computers to manipulate. This popular data format is widely used on websites that use Representational State Transfer (REST) web services. These functions improve the techniques used within FileMaker when manipulating data provided by external resources. However, they can also be enormously useful internally as a data exchange format between scripts and other functions.

For more technical information on JSON, visit: <http://www.json.org>.

Defining the JSON Format

The JSON format allows information to be stored as key/value pairs, where one component denotes the name or key of the other stored value. These are commonly referred to by various terms in different languages: object, record, struct, dictionary, hash table, keyed list, or associative array. Therefore, the terminology used in this section may not exactly match what you are accustomed to, depending on your experience with different languages.

A JSON *object* is a bracketed list of elements. A JSON *element* is a combination of an identifying key and associated value, like this:

```
{ "<Key>":<Value> }
```

Additional elements can be added with a comma delimiter

```
{ "<Key1>":<Value1>,<Key2>":<Value2> }
```

The *key* portion, sometimes called a *member*, is always a text string that names the associated value, acting kind of like a field name. The *value* portion contains an actual stored value that can be one of the following data types:

- JSONString — a text string.
- JSONNumber — a numeric value.
- JSONBoolean — a value that is either true or false.

- `JSONArray` — an ordered list of comma-separated values contained within square brackets. For example, `[1,2,3,4]`.
- `JSONObject` — a nested JSON object.
- `JSONNull` — a null value.
- `JSONRaw` — a value that will be determined by the JSON parser.

A JSON object identifying a person, may contain elements such as an identification number, first name, last name, and title, as shown here:

```
{"id":5103,"First":"John","Last":"Smith","Title":"Chief Technology Officer"}
```

The object can be typed into a more readable multi-line format, as shown here:

```
{
  "id":5103,
  "First":"John",
  "Last":"Smith",
  "Title":"Chief Technology Officer"
}
```

This example shows a JSON object containing two elements, `Product1` and `Product2`, each containing a nested JSON object.

```
{
  "Product1":
    {
      "id":1000,
      "name":"Widget 1",
      "price":39.99,
      "aisles":[3,8]
    },
  "Product2":
    {
      "id":1001,
      "name":"Widget 2",
      "price":59.99,
      "aisles":[2,4]
    }
}
```

This example shows a JSON object containing an array at the top level. Notice that the curly brackets are only used within the array if the value at a given position is a JSON Object.

```
[
  15,
  "Hello World",
  {"First":"John","Last":"Smith"}
]
```

■ **Tip** When writing JSON objects by hand, you can validate them using the tool at: <http://www.jsonlint.com>.

Referring to JSON Elements

As indicated by the `keyOrIndexOrPath`, we will see in the forthcoming JSON functions that an element in an object can be referred to by one of three different methods:

- Key
- Array index
- Path

Referring to an Element by Key

To refer to an element by *key*, use the name of the key. Using the following example object, referring to the `Title` element would be done by assigning `keyOrIndexOrPath` a value of `"Title"`:

```
{
  "id":5103,
  "First":"John",
  "Last":"Smith",
  "Title":"Chief Technology Officer"
}
```

Referring to an Element by Array Index

When referring to an element in an array at the root level of the object, use the index value. For example, to extract the second value below, assign `keyOrIndexOrPath` a value of `"[2]"`:

```
[
  15,
  "Hello World",
  {"First":"John", "Last":"Smith"}
]
```

When referring to a value in an array that is contained within another element rather than at the root level, you will need to either specify the path or extract the element's value first and then use the index position.

Referring to an Element by Path

The value stored in a JSON element can be another JSON object. To refer to an element within a nested object or to refer to an array item by index when the array is not at the root level of the object, you must specify a *path* to the element.

A path is specified by denoting each key in a period delimited list that is necessary to traverse the hierarchical structure down to the desired element. This pattern demonstrates how to navigate down three levels deep:

```
key1.key2.key3
```

An array, at any level, must denote a numeric index value instead of a named key. In this pattern, the second level would be an array containing several JSON objects and the path indicates the second of these is desired:

```
key1.[2].key3
```

Using this JSON object, let's work through some examples of using paths to refer to nested elements:

```
{
  "Product1":
    {
      "id":1000,
      "name":"Widget 1",
      "price":39.99,
      "aisles":[3,8]
    },
  "Product2":
    {
      "id":1001,
      "name":"Widget 2",
      "price":59.99,
      "aisles":[2,4]
    }
}
```

If the `keyOrIndexOrPath` parameter is "Product2" it will return the entire nested object stored under that key, as shown here:

```
Product2 =
{
  "id":1001,
  "name":"Widget 2",
  "price":59.99,
  "aisles":[2,4]
}
```

However, if the `keyOrIndexOrPath` parameter contains a path "Product2.name" it will only return the specified element from that product's object, as shown here:

```
Product2.name = Widget 2
```

To access a specific store aisle from the array of values stored in that element, the path would include the index to the desired array value. For example:

```
Product2.aisles.[2] = 4
```

JSON Function Reference

FileMaker provides the following JSON text functions:

- JSONDeleteElement
- JSONFormatElements
- JSONGetElement
- JSONListKeys
- JSONListValues
- JSONSetElement

JSONDeleteElement

The JSONDeleteElement function deletes a specified element from a JSON object.

Format	JSONDeleteElement (json ; keyOrIndexOrPath)
Parameters	json = a text value that contains a JSON object. keyOrIndexOrPath = a text value that specifies the JSON element that should be deleted.
Result	Returns the JSON object with the specified element removed.

This example assumes a field named Data that contains the following JSON object:

```
Sandbox::Example Text = {"First":"John","Last":"Smith","Title":"Chief Technology Officer"}
```

An element can be deleted from the object by element name, as shown here:

```
JSONDeleteElement (Sandbox::Example Text; "Title" )
// result = {"First":"John","Last":"Smith"}
```

JSONFormatElements

The JSONFormatElements function inserts extra space into a JSON object to render it in a more readable format.

Format	JSONFormatElements (json)
Parameters	json = a text value that contains a JSON array.
Result	The JSON object converted into a more user-friendly format.

Here is an example of the formatted results of this function:

```
JSONFormatElements ( {"First":"John","Last":"Smith","Title":"Chief Technology Officer"})
// result =
{
    "First" : "John",
    "Last"  : "Smith",
    "Title" : "Chief Technology Officer"
}
```

JSONGetElement

The JSONGetElement function extracts the value for a specific element from a JSON object.

Format	JSONGetElement (json ; keyOrIndexOrPath)
Parameters	json = a text value that contains a JSON array. keyOrIndexOrPath = the name or path to a desired element or an index to a desired array value.
Result	The specified value, extracted from the specified JSON object.

Here is an example showing the results of the function:

```
$JSON = {"First":"John","Last":"Smith","Title":"Chief Technology Officer"}
JSONGetElement ( $JSON ; "Title")
// result = "Chief Technology Officer"
```

JSONListKeys

The JSONListKeys function extracts a list of the keys of the specified element of a JSON object.

Format	JSONListKeys (json ; keyOrIndexOrPath)
Parameters	json = a text value that contains a JSON array. keyOrIndexOrPath = the name or path to a desired element or an index to a desired array value. Leave as an empty string to list the keys at the root level of the object.
Result	A carriage return-delimited list of the root-level keys for the specified element.

Here is an example showing the resulting list of keys:

```
JSONListKeys ( {"First":"John","Last":"Smith","Title":"Chief Technology Officer"} )
// result =
First
Last
Title
```


JSONListValues

The JSONListValues function extracts a list of the keys of the specified element of a JSON object.

Format	JSONListValues (json ; keyOrIndexOrPath)
Parameters	json = a text value that contains a JSON array. keyOrIndexOrPath = the name or path to a desired element or an index to a desired array value. Leave as an empty string to list the keys at the root level of the object.
Result	A carriage return-delimited list of the root-level values for the specified element.

Here is an example showing the resulting list of values:

```
JSONListValues ( {"First":"John","Last":"Smith","Title":"Chief Technology Officer"} )
// result =
    John
    Smith
    Chief Technology Officer
```

JSONSetElement

The JSONSetElement function adds or modifies the value for a specific element from a JSON object.

Format	To modify a single element: JSONSetElement (json ; keyOrIndexOrPath ; value ; type) To modify multiple elements, add square brackets around the last three parameters, which can repeat with a semicolon delimiter: JSONSetElement (json ; [key1 ; value1 ; type1] ; [key2 ; value2 ; type2] ; [key3 ; value3 ; type3])
Parameters	json = a text value that contains a JSON array. keyOrIndexOrPath = the name or path to a desired element or an index to a desired array value. If the element doesn't exist, it will be created. value = a text value that should become the new value for the specified element. type = the name of the data type of the new element.
Result	The JSON object with the new or modified element.

To create a new JSON object:

```
JSONSetElement ( "" ; "id" ; 1000 ; JSONNumber)
// result = {"id":1000}
```

To create a new JSON object with multiple elements:

```
JSONSetElement ( "" ;
    [ "id" ; 1000 ; JSONNumber ] ;
    [ "first" ; "John" ; JSONString ] ;
    [ "last" ; "Smith" ; JSONString ] ;
    [ "title" ; "Chief Technology Officer" ; JSONString ]
)
// result = {"first":"John","id":1000,"last":"Smith","title":"Chief Technology Officer"}
```

To modify an element in an existing JSON object:

```
JSONSetElement ( {"first":"John","last":"Smith"} ; "first" ; "Jane" ; JSONString )
// result = {"first":"Jane","last":"Smith"}
```

Summary

In this chapter, we introduced JSON objects and explored all the built-in functions for creating, parsing, and manipulating them.

CHAPTER 15



Text Formatting Functions

Text formatting functions are used to modify the font, style, size, and color of text strings, giving developers control over how text appears in fields on layouts *and* how it is stored in fields.

These functions are used to apply formatting to the text contained *within* a field. When a field object is placed on a layout, formatting can be applied that controls how field content appears for every record when displayed on a layout. This is completely independent of the actual data stored in the field. When displayed, layout format settings are superimposed on the field's text content unless that content has format settings applied within the field. In that case, the field settings override the layout settings for the portion of text that has format applied.

All of these functions only affect the formatting of text within the field and not the actual text content.

Text Formatting Function Reference

FileMaker provides the following built-in text formatting functions:

- RGB
- TextColor
- TextColorRemove
- TextFont
- TextFontRemove
- TextSize
- TextSizeRemove
- TextStyleAdd
- TextStyleRemove

When formatting is applied to text, it will be retained only if the data type returned is text and the result is placed into a field. Similarly, when all of a certain type of formatting is removed — fonts, size, color style — and the text is placed into a field, it will then display using the field's layout format settings.

Some actions performed with formatted text may result in it being converted into plain, unformatted text. For example, formatting is not retained when text is added to the body of an email using the `Send Email` script step.

RGB

The `RGB` function is used to generate an integer that represents a specific color by combining red, green, and blue values.

Format	<code>RGB (red ; green ; blue)</code>
Parameters	<p><code>red</code> = a number between 0 and 255 representing the amount of red. <code>green</code> = a number between 0 and 255 representing the amount of green. <code>blue</code> = a number between 0 and 255 representing the amount of blue.</p>
Result	<p>A number between 0 and 16777215 that is derived by evaluating the following formula using the parameters provided:</p> $\text{red} * 256^2 + \text{green} * 256 + \text{blue}$

These examples show the function operating on different input:

```
RGB ( 255 ; 0 ; 0 )      // result = 16711680
RGB ( 0 ; 0 ; 0 )      // result = 0
```

TextColor

The `TextColor` function applies a specified color to a text string.

Format	<code>TextColor (text ; RGB)</code>
Parameters	<p><code>text</code> = a text value to which a font color should be applied. <code>RGB</code> = an integer from 0 to 16777215 indicating the desired color for the text provided. Can be generated by a nested call to the <code>RGB</code> function.</p>
Result	The text provided with the specified color applied.

Here is an example of the function:

```
TextColor ( "Mark Munro" ; RGB ( 255 ; 0 ; 0 ) )
// result (formatted in red) = "Mark Munro"
```

TextColorRemove

The `TextColorRemove` function removes all font colors or one specific color from a text string depending on the instructions it receives.

Format	To remove all color settings from the entire text, only a single parameter is required: <code>TextColorRemove (text)</code> To remove one color used anywhere in the text, add a second optional parameter that specifies that color: <code>TextColorRemove (text ; RGB)</code>
Parameters	<code>text</code> = a text value that contains text formatted in one or more colors. <code>RGB</code> = (optional) an integer from 0 to 16777215 indicating the desired color for the text provided. Can be generated by a nested call to the <code>RGB</code> function.
Result	The text provided with all or the specified color settings removed.

This example pulls the text out of a field named “Notes” in a Contacts table and removes all color settings:

```
TextColorRemove ( Contact::Notes )
```

The following example uses several functions to apply different colors to some sample text and then uses `TextColorRemove` to remove only the green color. If the resulting text is placed into a field, the other colors would remain while the green text returns to black or whatever layout color setting is applied to the field object.

```
Let ( [
    textRed = TextColor ( "Hello World Red. " ; RGB ( 255 ; 0 ; 0 ) ) ;
    textGreen = TextColor ( "Hello World Blue. " ; RGB ( 0 ; 255 ; 0 ) ) ;
    textBlue = TextColor ( "Hello World Green. " ; RGB ( 0 ; 0 ; 255 ) ) ;
    textMulti = textRed & textGreen & textBlue
] ;
    TextColorRemove (textMulti ; RGB ( 0 ; 255 ; 0 ) )
)
```

TextFont

The `TextFont` function changes the font of a text string.

Format	<code>TextFont (text ; fontName)</code>
Parameters	<code>text</code> = a text value from which font styling should be applied. <code>fontName</code> = a text value containing the name of the desired font.
Result	The text provided with the specified font applied.

Here are some examples of applying a font to some text:

```
TextFont ( "Hello World" ; "Arial Narrow" )
TextFont ( "Hello World" ; "Courier New" )
```

TextFontRemove

The TextFontRemove function removes all fonts or one specific font from a text string.

Format	To remove all font settings from the entire text, only a single parameter is required: TextFontRemove (text) To remove one font used anywhere in the text, add a second optional parameter that specifies that font name: TextFontRemove (text ; fontToRemove)
Parameters	text = a text value from which specific font styling should be removed. fontToRemove = a text value containing the name of the desired font.
Result	The text provided with either all or the specified font settings removed.

This example will remove all font settings from any text within a Notes field in a Contacts table:

```
TextFontRemove ( Contact::Notes )
```

This example will remove any application of Courier New from anywhere within the text in the field:

```
TextFontRemove ( Contact::Notes ; "Courier New" )
```

TextFormatRemove

The TextFormatRemove function removes all text formatting from a text string including fonts, styles, font sizes, and font colors.

Format	TextFormatRemove (text)
Parameters	text = a text value from which text formatting should be completely removed.
Result	The text provided as a plain, unformatted string.

These examples show the use of the function to remove all formatting from a text string and from text extracted from a field:

```
TextFormatRemove ( "Hello World" )
TextFormatRemove ( Contact::Notes )
```

TextSize

The TextSize function changes the font size of a text string.

Format	TextSize (text ; fontSize)
Parameters	text = a text value to which a font size should be applied. fontSize = a font size expressed as an integer.
Result	The text provided with the desired font size applied.

These examples show the application of a font size to a text string and to text extracted from a field:

```
textSize ( "Hello World" ; 18 )
textSize ( Contact::Notes ; 12 )
```

TextSizeRemove

The TextSizeRemove function removes all font size settings or one specific size from a text string.

Format	To remove all sizes from the entire text, only a single parameter is required: TextSizeRemove (text) To remove one size used anywhere in the text, add a second optional parameter that specifies that one size: TextSizeRemove (text ; sizeToRemove)
Parameters	text = a text value from which text size formatting should be removed. sizeToRemove = a font size expressed as an integer.
Result	The text provided with all or the specified font sizes removed.

This example will remove all font sizes from any text within a “Notes” field in a “Contacts” table:

```
TextSizeRemove ( Contact::Notes )
```

This example will remove any application of 18 pt. font size from anywhere within the text:

```
TextFontRemove ( Contact::Notes ; 18 )
```

TextStyleAdd

The TextStyleAdd function changes the font style of a text string.

Format	TextStyleAdd (text ; styles)
Parameters	text = a text value to which text styles should be applied. styles = one or more style, names delimited with a plus symbol (+).
Options for Styles	Plain, Bold, Italic, Underline, HighlightYellow, Condense, Extend, Strikethrough, SmallCaps, Superscript, Subscript, Uppercase, Lowercase, Titlecase, WordUnderline, DoubleUnderline.
Result	The text provided with the desired font style(s) applied.

These examples show the application of one or more styles to a text string or to text extracted from a field:

```
textSize ( "Hello World" ; Plain )
textSize ( "Hello World" ; Italic+Underline )
textSize ( Contact::Notes ; Strikethrough )
```

TextStyleRemove

The TextStyleRemove function removes all font style settings or one or more specific styles from a text string depending on the instructions it receives.

Format	To remove all styles from the entire text, only a single parameter is required: TextStyleRemove (text) To remove one or more specific styles used anywhere in the text, add a second optional parameter that specifies which styles: TextStyleRemove (text ; styles)
Parameters	text = a text value from which text styles should be removed. styles = one or more style names delimited with a plus symbol (+).
Result	The text provided with all or the specified font style(s) removed.

This example will remove *all font styles* from the text extracted from a Notes field in a Contacts table:

```
TextStyleRemove ( Contact::Notes )
```

This example will remove any application of *specific styles* from anywhere within text extracted from a field:

```
TextStyleRemove ( Contact::Notes ; bold )
TextStyleRemove ( Contact::Notes ; italic+underline )
```

Summary

In this chapter, we explored the built-in functions for formatting text.

CHAPTER 16



Number Functions

Number functions are used to manipulate numbers.

Number Function Reference

FileMaker provides the following built-in number functions:

- Abs
- Ceiling
- Combination
- Div
- Exp
- Factorial
- Floor
- Int
- Lg
- Ln
- Log
- Mod
- Random
- Round
- SetPrecision
- Sign
- Sqrt
- Truncate

Abs

The Abs function calculates the absolute value of a number, which is a non-negative value representing its distance from zero.

Format	Abs (number)
Parameters	number = any number.
Result	A number representing the absolute value of the number provided.

These examples show the function operating on a positive number, negative number, math expression, a text string containing a number, and a number extracted from a field:

```
Abs ( 8.75 )           // result = 8.75
Abs ( -5 )            // result = 5
Abs ( 4 - 8 )         // result = 4
Abs ( "-9" )          // result = 9
Abs ( Invoice::Balance Due ) // result = 95
```

Ceiling

The Ceiling function rounds a number *up* to the next integer.

Format	Ceiling (number)
Parameters	number = any number.
Result	The number provided rounded up to the next integer.

Here are some examples of the function operating on a positive and negative number:

```
Ceiling ( 1.4 )           // result = 2
Ceiling ( -2.5 )          // result = -2
Ceiling ( 2.00000001 )    // result = 3
```

Combination

The Combination function calculates the number of combinations possible based on a specific set size and number of choices, which can be useful in statistics and other mathematics.

Format	Combination (setSize ; numberOfChoices)
Parameters	setSize = any positive number. numberOfChoices = a number indicating the number of choices.
Result	A number indicating the possible combinations of choices given the parameters specified.

Given a set size of three values (a, b, c) and making choices of two values, the number of possible unique combinations is *three*; ab, ac, bc. No matter the set size or number of choices, the function will calculate the possible combinations, as shown in these examples:

```
Combination ( 3 ; 2 )    // result = 3
Combination ( 4 ; 2 )    // result = 6
Combination ( 8 ; 3 )    // result = 56
```

This example demonstrates an erroneous attempt to calculate the impossible combinations of *six* values with a set of only *four* values:

```
Combination ( 4 ; 6 )    // result = ?
```

Div

The Div function calculates the next lowest integer from the result of a number being divided by a divisor.

Format	Div (number ; divisor)
Parameters	number = any number. divisor = a number to be used as a divisor.
Result	An integer number.

These examples show the function operating on different input:

```
Div ( 10 ; 3 )          // result = 3
Div ( 6.25 ; 2.75 )    // result = 2
```

Exp

The Exp function calculates the mathematical constant e (approximately equal to 2.7182818) raised to the power of a specified number.

Format	Exp (number)
Parameters	number = any number.
Result	The value of the constant raised to the specified power.

These examples show the function operating on different input:

```
Exp ( 1 )              // result = 2.71828182
Exp ( 3 )              // result = 20.0855369231876677
```

Factorial

The Factorial function calculates a non-negative integer ($n!$), which is the product of multiplying all positive integers less than or equal to a specified number (n), stopping at 1, or at an optional number of factors.

Format When not specifying a limit to the number of factors, a single parameter is required:

Factorial (number)

To limit to the number of factors, a second, optional parameter is added:

Factorial (number ; numberOfFactors)

Parameters number = a positive integer which will be used as n .
 numberOfFactors = a positive integer that indicates how many factors should be included.

Result A number representing the factorial of the number provided.

These examples show the function operating on different input:

```
Factorial ( 3 )            // result = 6
Factorial ( 4 )            // result = 24
Factorial ( 4 ; 2 )        // result = 12
```

Floor

The Floor function rounds a number down to the next integer.

Format Floor (number)

Parameters number = any number.

Result The number provided rounded down to the next integer.

These examples show the function operating on different input:

```
Floor ( 1.4 )     // result = 1
Floor ( -2.5 )    // result = -3
```

Int

The Int function returns the integer part of a number without rounding by dropping any digits to the right of the decimal.

Format Int (number)

Parameters number = any number.

Result An integer of the number provided.

These examples show the function operating on different input:

```
Int ( 34.2653 ) // result = 34
Int ( -2.85 )   // result = -2
Int ( 10 / 3 )  // result = 3
```

Lg

The Lg function calculates the base-2 logarithm of a number, which is the power to which the number must be raised to obtain the value desired.

Format	Lg (number)
Parameters	number = a positive number.
Result	A number.

These examples show the function operating on different input:

```
Lg ( 4 )        // result = 2
Lg ( -2 )       // result = ? (Error)
```

Ln

The Ln function calculates the base-e (natural) logarithm of a number, which is the power to which the mathematical constant e (2.71828182) would have to be raised to equal the number specified.

Format	Ln (number)
Parameters	number = any number.
Result	A number.

These examples show the function operating on different input:

```
Ln ( 4 )        // result = 1.3862943611198906
Ln ( 4 )        // result = 1.3862943611198906
Ln ( 10 / 5 )   // result = .6931471805599453
```

Log

The Log function calculates the base-10 (common) logarithm of a number.

Format	Log (number)
Parameters	number = a positive number.
Result	A number.

These examples show the function operating on different input:

```
Log ( 50 )      // result = 1.6989700043360188
Log ( 100 )    // result = 2
Log ( -75 )    // result = ? (Error)
```

Mod

The Mod function calculates the remainder from the result of a number being divided by a divisor. This is useful when converting units of measure, such as converting seconds to minutes, minutes to hours, or days to years.

Format	Mod (number ; divisor)
Parameters	number = any number. divisor = a number to be used as the divisor.
Result	A number.

These examples show the function operating on different input:

```
Mod ( 100 ; 60 )      // result = 40
Mod ( 410 ; 365 )    // result = 45
```

This example shows the function used with the Int function to convert 310 minutes into a text string that expresses that as hours and minutes:

```
Int ( 310 / 60 ) & " hours, " & Mod ( 310 ; 60 ) & " minutes"
// result = 5 hours, 10 minutes
```

Random

The Random function calculates a pseudo-random number between 0 and .99999999999999999999.

Format	Random
Parameters	Not applicable
Result	A random number.

These examples show the function providing a different value each time it is called:

```
Random          // result = .69521348632189605699
Random          // result = .49928041600104466902
Random          // result = .16828354164749970145
```

These examples show how to use the function to generate a random integer between 0 and 10:

```
Int ( 10 * Random ) // result = 8
Int ( 10 * Random ) // result = 3
Int ( 10 * Random ) // result = 6
```

Round

The Round function rounds a number to specified number of decimal places.

Format	Round (number ; precision)
Parameters	number = a number that will be rounded.
Result	The number provided rounded as specified.

These examples show the function operating on different input:

```
Round ( 10.1564 ; 2 ) // result = 10.16
Round ( 10 / 3 ; 0 ) // result = 3
```

SetPrecision

The SetPrecision function evaluates a math expression with a specified precision between 16 and 400.

Format	SetPrecision (expression ; precision)
Parameters	expression = any number or numeric expression. precision = a number between 16 and 400 indicating the desired precision.
Result	The result of the equation provided, expressed to the specified precision.

These examples show the function operating on different input:

```
SetPrecision ( 10 / 3 ; 16 ) // result = 3.3333333333333333
SetPrecision ( 150.756234915268458475965 ; 0 ) // result = 150.7562349152684585
```

Sign

The Sign function determines if a given number is negative or positive.

Format	Sign (number)
Parameters	number = any number.
Result	A number indicating the sign of the number provided as follows: -1 = the number provided is negative 1 = the number provided is positive 0 = the number provided is zero

These examples show the function operating on different input:

```
Sign ( 15 ) // result = 1
Sign ( 3 - 20 ) // result = -1
```

Sqrt

The Sqrt function calculates the square root of a number.

Format	Sqrt (number)
Parameters	number = a positive number.
Result	A number that is the square root of the number provided.

These examples show the function operating on different input:

```
Sqrt ( 4 )           // result = 2
Sqrt ( 10 )         // result = 3.1622776601683793
```

Truncate

The Truncate function shortens a number to a specified number of decimal places without rounding.

Format	Truncate (number ; precision)
Parameters	number = a number. precision = a numeric expression or field indicating the desired number of decimal places.
Result	The number provided with the number of decimal places adjusted as specified.

These examples show the function operating on different input:

```
Truncate ( 10.246913 ; 2 ) // result = 10.24
```

Summary

In this chapter, we explored all the built-in functions for manipulating numeric values.

CHAPTER 17



Date, Time, and Timestamp Functions

These three categories provide functions that manipulate date and time values.

Date Function Reference

Date functions are used to create and manipulate dates. FileMaker provides the following built-in date functions:

- Date
- Day
- Day Name
- DayNameJ
- DayOfWeek
- DayOfYear
- Month
- MonthName
- MonthNameJ
- WeekOfYear
- WeekOfYearFiscal
- Year
- YearName

Date

The Date function creates a date object from separate numeric month, day, and year values.

Format	Date (month ; day ; year)
Parameters	month = a number indicating a month. day = a number indicating a day of the month. year = a number indicating a year.
Result	A date object constructed from the values provided.

This basic example constructs a date object from three numeric values:

```
Date ( 1 ; 15 ; 2017 )           // result = 1/15/2017
```

This example demonstrates that any of the parameters can be expressions that will be evaluated prior to the construction of the date:

```
Date ( 1 ; 15 ; 2027 - 10 )     // result = 1/15/2017
```

The function will automatically wrap to a new month, day or year if the month or day provided falls out of range. For example, a month value of 13 will cause the function to automatically return a date for January of the following year:

```
Date ( 13 ; 15 ; 2017 )       // result = 1/15/2018
```

Day

The Day function extracts the day number of the month for a given date.

Format	Day (date)
Parameters	date = a date.
Result	A number.

These examples show the function operating on different input:

```
Day ( Date ( 1 ; 15 ; 2017 ) ) // result = 15
Day ( "2/26/2017" )           // result = 26
```

DayName

The DayName function calculates the name of the weekday for a given date.

Format	DayName (date)
Parameters	date = a date.
Result	A text string containing the name of the weekday for the date provided.

These examples show the function operating on different input:

```
DayName ( Date ( 1 ; 15 ; 2017 ) )    // result = Sunday
DayName ( "6/27/1758" )             // result = Tuesday
```

DayNameJ

The DayNameJ function performs the same function as DayName but returns text in Japanese. For more information on Japanese functions, see FileMaker's documentation.

DayOfWeek

The DayOfWeek function calculates a number representing the day of the week for a given date.

Format	DayOfWeek (date)
Parameters	date = a date.
Result	A number from 1 to 7 indicating where the date provided falls on the calendar week from Sunday to Monday.

These examples show the function operating on different input:

```
DayOfWeek ( Date ( 1 ; 15 ; 2017 ) ) // result = 1
DayOfWeek ( "6/27/1758" )           // result = 3
```

DayOfYear

The DayOfYear function calculates a number representing the day of the year for a given date.

Format	DayOfYear (date)
Parameters	date = a date.
Result	A number from 1 to 365 indicating where the date provided falls within a year.

These examples show the function operating on different input:

```
DayOfYear ( Date ( 1 ; 15 ; 2017 ) ) // result = 15
DayOfYear ( "6/27/1758" )           // result = 178
```

Month

The Month function calculates the number of the month for a given date.

Format	Month (date)
Parameters	date = a date.
Result	A number from 1 to 12 indicating the month number of the date provided.

These examples show the function operating on different input:

```
Month ( Date ( 1 ; 15 ; 2017 ) ) // result = 1
Month ( "6/27/1758" ) // result = 6
```

MonthName

The MonthName function calculates the name of the month for a given date.

Format	MonthName (date)
Parameters	date = a date.
Result	A text string containing the name of the month of the date provided.

These examples show the function operating on different input:

```
MonthName ( Date ( 1 ; 15 ; 2017 ) ) // result = "January"
MonthName ( "6/27/1758" ) // result = "June"
```

MonthNameJ

The MonthNameJ function performs the same function as MonthName but returns text in Japanese. For more information on Japanese functions, see FileMaker's documentation.

WeekOfYear

The WeekOfYear function calculates a number representing the week of the year for a given date.

Format	WeekOfYear (date)
Parameters	date = a date.
Result	A number from 1 to 52 indicating the week number within a year of the date provided.

These examples show the function operating on different input:

```
WeekOfYear ( Date ( 1 ; 15 ; 2017 ) ) // result = 3
WeekOfYear ( "6/27/1758" ) // result = 26
```

WeekOfYearFiscal

The `WeekOfYearFiscal` function calculates a number representing the week of a year for a given date based on a specified starting date for a workweek. This is sometimes useful in accounting to calculate if a year has an extra pay period because a week is split across the calendar year boundary.

Format	<code>WeekOfYearFiscal (date ; startingDay)</code>
Parameters	<p><code>date</code> = a date.</p> <p><code>startingDay</code> = a number indicating the start of a workweek: 1 = Sunday, 2 = Monday, 3 = Tuesday, etc.</p>
Result	A number indicating the week of the year for the date provided based on the start of the workweek specified.

These examples show how January 2, 2009, which falls on a Friday, can be either the first week of 2009 or the fifty-third week of 2008 depending on the day considered to be the start of the week:

```
WeekOfYearFiscal ( "1/2/2009" ; 1 ) // result = 53
WeekOfYearFiscal ( "1/2/2009" ; 2 ) // result = 1
WeekOfYearFiscal ( "1/2/2009" ; 3 ) // result = 1
WeekOfYearFiscal ( "1/2/2009" ; 4 ) // result = 1
WeekOfYearFiscal ( "1/2/2009" ; 5 ) // result = 1
WeekOfYearFiscal ( "1/2/2009" ; 6 ) // result = 1
WeekOfYearFiscal ( "1/2/2009" ; 7 ) // result = 53
```

Year

The `Year` function returns the year for a given date.

Format	<code>Year (date)</code>
Parameters	<code>date</code> = a date.
Result	A number that is the year for the date provided.

These examples show the function operating on different input:

```
Year ( Date ( 1 ; 15 ; 2017 ) ) // result = 2017
Year ( "6/27/1758" ) // result = 1758
```

YearName

The `YearName` function calculates the Japanese year name of a given date. For more information on Japanese functions, see FileMaker's documentation.

Time Function Reference

Time functions are used to create and manipulate time values. FileMaker provides the following built-in time functions:

- Hour
- Minute
- Second
- Time

Hour

The `Hour` function extracts the number of hours of a given time.

Format	<code>Hour (time)</code>
Parameters	<code>time</code> = a time.
Result	The hour number of the time provided.

These examples show the function operating on different input:

```
Hour ( "09:15:55 AM" )           // result = 9
Hour ( "4/20/2017 03:30:00 PM" ) // result = 15
```

Minute

The `Minute` function extracts the number of minutes from a given time.

Format	<code>Minute (time)</code>
Parameters	<code>time</code> = a time.
Result	The number of minutes from the time provided.

These examples show the function operating on different input:

```
Minute ( "09:15:55 AM" )         // result = 15
Minute ( "4/20/2017 03:30:00 PM" ) // result = 30
```

Seconds

The Seconds function extracts the number of seconds from a given time.

Format	Seconds (time)
Parameters	time = a time.
Result	The number of seconds from the time provided.

These examples show the function operating on different input:

```
Seconds ( "09:15:55 AM" )      // result = 55
Seconds ( "4/20/2017 03:30:00 PM" ) // result = 0
```

Time

The Time function creates a time object from separate numeric hours, minutes, and seconds values.

Format	Time (hours ; minutes ; seconds)
Parameters	hours = a number representing hours. minutes = a number representing minutes. seconds = a number representing seconds.
Result	A time object constructed from the values provided.

These examples show the function operating on different input:

```
Time ( 9 ; 15 ; 55 ) // result = 9:15:55
Time ( 2 ; 8 ; 19 ) // result = 2:8:19
```

Timestamp Function Reference

Timestamp functions are used to create and manipulate timestamp data, which is a combination of a date and time value in a single string. FileMaker provides the following built-in timestamp function:

- Timestamp

Timestamp

The Timestamp function creates a timestamp based on a given date and time.

Format	Timestamp (date ; time)
Parameters	date = a date. time = a time.
Result	A timestamp from the date and time provided.

These examples show that timestamps will automatically add the appropriate AM/PM suffix to the time portion:

```
Timestamp ( "1/15/2017" ; "9:15:55" )  
// result = 1/15/2017 9:15:55 AM
```

```
Timestamp ( Date ( 5 ; 10 ; 1990 ) ; Time ( 10 ; 30 ; 00 ) )  
// result = 5/10/1990 10:30:00 AM
```

If the hours are out of range, as in military time, the function will automatically convert to civilian time, also with the appropriate AM/PM suffix:

```
Timestamp ( "1/15/2017" ; "15:15:55" )  
// result = 1/15/2017 3:15:55 PM
```

Summary

In this chapter, we explored all the built-in functions for manipulating date, time, and timestamp values.

CHAPTER 18



Container Functions

Container functions are used to analyze, manipulate, and report about data in container fields.

Container Function Reference

FileMaker provides the following built-in container functions:

- Base64Decode
- Base64Encode
- Base64EncodeRFC
- GetContainerAttribute
- GetHeight
- GetThumbnail
- GetWidth
- VerifyContainer

Base64Decode

The Base64Decode function converts Base64 encoded text into its original content: either text or container data. Since Base64 format does not preserve the file name or extension, these can be included as an optional parameter.

Base64 is a group of encoding schemes for converting binary content to ASCII strings. Generally, it is used to encode graphics, files, and applications for transmission using methods designed to deal with textual data. In FileMaker, these types of data are stored in container fields.

Format	Without specifying a file name and extension, generic values will be automatically created. In this case, only the text being decoded need be included: Base64Decode (text) To specify a file name with an extension for the file being created, add a second parameter: Base64Decode (text ; fileNameWithExtension)
Parameters	text = a text value encoded as Base64 that will be decoded. fileNameWithExtension = (optional) a text value that contains a file name and extension for the file being created from the Base64 text.
Result	A text string suitable for placement in a text or container field.
More Info	For more information about Base64 visit: https://www.base64decode.org .

This example shows the encoded contents of a field named EncodedData being converted into an image with a name of sample.jpg:

```
Base64Decode ( EncodedData ; "sample.jpg" )
// result = an image ready for placement within a container field
```

Base64Encode

The Base64Encode function converts text or container data into Base64 formatted text.

Format	Base64Encode (sourceField)
Parameters	sourceField = a text value that will be encoded.
Result	A text string encoded in Base64 format.

These examples show the function operating on different input:

```
Base64Encode ( Image::PreviewImage )
// result = the image as a string of text characters in Base64 format
```

```
Base64Encode ( "Hello World" )
// result = aHR0cHM6Ly95b3V0dS5iZS9kUXc0dz1XZ1hjUQ==
```

Base64EncodeRFC

The Base64EncodeRFC function converts text or container data into the specified Base64 formatted text.

A Request for Comment (RFC) is a series of documents that emerged from the research efforts that created the ARPANET and later the Internet. When the Internet Engineering Task Force (IETF) was formed in the mid-1980s, RFCs were the primary vehicle for publishing IETF standards, many pertaining to the Internet protocols.

Format	Base64EncodeRFC (RFCNumber ; data)
Parameters	RFCNumber = a number indicating the RFC standard number to use. data = a text value that will be encoded.
Options for RFCNumber	The function supports the following RFC standards: 1421 — 64-character line length with CRLF (carriage return, line feed) line endings. 2045 — 72-character line length and CRLF line endings. 3548 — No line breaks. 4648 — No line breaks. This is the default RFC standard that will be used if the value provided is empty or an unrecognized value. 4880 — 76-character line length, CRLF line ends with a 24-bit CRC value appended.
Result	A text string encoded in Base64 format using the RFC standard specified.
More Info	For more technical information, visit: https://www.ietf.org/rfc.html .

This example shows the function encoding some text using the 3548 RFC:

```
Base64EncodeRFC ( 3548 ; "Hello World" )
// result = SGVsbG8gV29ybGQ=
```

CryptAuthCode

The `CryptAuthCode` function returns a binary keyed-hash message authentication code (HMAC) of text or container data using a specified cryptographic hash algorithm.

Format	CryptAuthCode (data ; algorithm ; key)
Parameters	data = a text value containing data that should be encrypted. algorithm = the name of a supported cryptographic algorithm that should be used to generate the HMAC. key = a text value that will be used as the key to encrypt the data.
Options for Algorithm	The following cryptographic algorithms are supported by this function: MD5 MDC2 SHA SHA1 SHA224 SHA256 SHA384 SHA512
Result	An encrypted text file that can be placed into a container field.

When this example is used as a calculation field resulting in container data or when the result is placed into a container field by script, the result will be an encrypted text file named `hmac.data`:

```
CryptAuthCode ( "Hello World" ; "MD5" ; "S3cr3at3Password" )
```

CryptEncrypt

The `CryptEncrypt` function encrypts text or container data into an encrypted file.

To convert the key parameter into a cryptographic key, the function uses the Password-Based Key Derivation Function 2 (PBKDF2) algorithm, which is part of RSA Laboratories' Public-Key Cryptography Standards (PKCS) series that was published as RFC 2898 by the Internet Engineering Task Force.

The cryptographic key is then used to encrypt the data using the Advanced Encryption Standard Galois/Counter Mode (AES-GCM) 128-bit algorithm. The resulting encrypted file includes an encrypted 256-bit Secure Hash Algorithm (SHA-256) digest of the data that is used to validate the data during encryption.

■ **Caution** The `Crypt` functions are *not* supported in runtime solutions (Chapter 41, “Developer Utilities”) and will return an error.

Format	<code>CryptEncrypt (data ; key)</code>
Parameters	<p><code>data</code> = a text value that should be encrypted.</p> <p><code>key</code> = a text value containing text that will be used as the key to encrypt the data.</p>
Result	An encrypted text file that can be placed into a container field.

When this example is used as a calculation field resulting in container data or when the result is placed into a container field by script, the result will be an encrypted text file named `encrypted.data`:

```
CryptEncrypt ( "Hello World" ; "S3cr3at3Password" )
```

CryptEncryptBase64

The `CryptEncryptBase64` function encrypts text or container data into Base64 text.

Format	<code>CryptEncryptBase64 (data ; key)</code>
Parameters	<p><code>data</code> = a text value that will be encrypted.</p> <p><code>key</code> = a text value containing text that will be used as the key to encrypt the data.</p>
Result	The data provided, encrypted into a Base64 text string.

Example

```
CryptEncryptBase64 ( "Hello World" ; "S3cr3at3Password" )
// result = qqGBz8icxONPAYGiIIRSW/JfOHIVApOmyZ8OD5vxWxPsiYxv14HBjUL/Ar8MXcjOpiENVeunbDcc+X4
+m3ZNwFH3/uws8580HPmtASBs9ZK
```

CryptDecrypt

The `CryptDecrypt` function converts an encrypted data file from a container back into its original content.

Format	<code>CryptDecrypt (container ; key)</code>
Parameters	<p><code>container</code> = data representing an encrypted file resides.</p> <p><code>key</code> = a text value containing text that will be used as the key to decrypt the data.</p>
Result	The original text or container content restored to its original unencrypted form.

This example assumes a field containing the encrypted `.data` file generated from the `CryptEncrypt` example:

```
CryptDecrypt ( Sandbox::Example Container ; "S3cr3at3Password" )
// result = "Hello World"
```

CryptDecryptBase64

The `CryptDecryptBase64` function converts text encrypted in Base64 format back into its original content.

Format	<code>CryptDecryptBase64 (data ; key)</code>
Parameters	<p><code>data</code> = data that was previously encrypted with into Base64 format.</p> <p><code>key</code> = a text value containing text that will be used as the key to decrypt the data.</p>
Result	The original text or container content restored to its original form.

This example assumes that the `Example Calculation` field contains the encrypted text that resulted from the `CryptEncryptBase64` example:

```
CryptDecrypt ( Sandbox::Example Calculation ; "S3cr3at3Password" )
// result = "Hello World"
```

CryptDigest

The `CryptDigest` function returns a binary hash value of text or container data using a specified cryptographic hash algorithm.

Format	<code>CryptDigest (data ; algorithm)</code>
Parameters	<p><code>data</code> = a text value that will be encrypted.</p> <p><code>algorithm</code> = the name of a supported cryptographic algorithm that should be used to generate the HMAC (see <code>CryptAuthCode</code> for list of supported algorithms).</p>
Result	A text file that can be placed into a container field, containing the encrypted digest of the original data.

When this example is used as a calculation field resulting in container data or when the result is placed into a container field by script, the result will be an encrypted text file named `digest.data`:

```
CryptDigest ( "Hello World" ; "MD5" ; "S3cr3at3Password" )
```

GetContainerAttribute

The `GetContainerAttribute` function returns an attribute from the file metadata of a container field.

Format	<code>GetContainerAttribute (sourceField ; attributeName)</code>
Parameters	<code>sourceField</code> = a container field reference. <code>attributeName</code> = the name of an attribute or group.
Result	Various, depending on the information requested and the file type stored in the container.

AttributeName Options

The `attributeName` parameter can contain the name of an individual attribute or a group of attributes.

■ **Caution** When attributes are retrieved in groups they always include a label with the value, while individual attributes only return the value. Some attributes will return either a number that indicates a value or a number with the text explanation of the value in parentheses. For more details on the values returned, experiment with groups or consult FileMaker's help documentation.

Groups of Attributes

The following groups of attributes can be retrieved:

- `all` = returns every attribute applicable to the file in the container
- `general` = returns every attribute applicable to all files types
- `audio` = returns every attribute applicable to audio files
- `image` = returns every attribute applicable to images
- `photo` = returns every attribute applicable to photographs
- `barcode` = returns every attribute applicable to barcodes
- `signature` = returns every attribute applicable to signatures

General Attributes

The following attributes are in the general group and are applicable to every file type:

- `filename` = the name of the file
- `MD5` = a cryptographic hash function result for the file, which can be used to identify duplicates with different file names
- `storageType` = the method of storage for the file; `Embedded`, `External (Secure)`, `External (Open)`, `File Reference`, `Text`
- `fileSize` = the size of the file in bytes
- `internalSize` = the amount of space used inside the database by the container field (in bytes)
- `externalSize` = the amount of space that is used outside the database by the container field (in bytes)
- `externalFiles` = a list of external files in the container field

Image Attributes

The following attributes are in the image group and are applicable to images:

- `width`
- `height`
- `dpiWidth`
- `dpiHeight`
- `transparency`

Photo Attributes

The following attributes are in the photo group and are applicable to `jpeg` and `tiff` files only:

- `orientation` = the orientation of the photo
- `created` = the earliest timestamp available for the photo
- `modified` = the latest timestamp available for the photo; an empty string indicates the photo has not been modified
- `latitude` = the latitude of the location where the photo was taken
- `longitude` = the longitude of the location where the photo was taken
- `make` = the manufacture of the camera used to take the photo
- `model` = the model of the camera used to take the photo

Audio Attributes

The following attributes are in the audio group and are applicable to mp3 and m4a files only:

- title
- artist
- year
- track
- genre
- composer
- cover Art
- duration
- bitRate

Barcode Attributes

The following attributes are in the barcode group and are applicable to barcode files only:

- barcodeText
- barcodeType

Signature Attributes

The following attributes are in the signature group and are applicable to signatures only:

- signed = the timestamp when the signature was captured

Examples

These examples retrieve an individual attribute from the general group:

```
GetContainerAttribute (Example::ContainerField; "filename" )
// result = Img_1036.jpg
```

```
GetContainerAttribute (Example::ContainerField; "storageType" )
// result = Embedded
```

These examples retrieve an individual attribute from the photo group:

```
GetContainerAttribute (Example::ContainerField; "width" )
// result = 2448
```

```
GetContainerAttribute (Example::ContainerField; "dpiwidth" )
// result = 72
```

```
GetContainerAttribute (Example::ContainerField; "orientation" )
// result = 6 (Rotated 90 degrees counterclockwise)
```


This example retrieves the entire photos group:

```
GetContainerAttribute (Example::ContainerField; "photo" )
// result =
    Orientation: 1 (Normal)
    Created: 10/6/2009 6:55:56 AM
    Modified:
    Latitude:
    Longitude:
    Make: Canon
    Model: Canon EOS 5D
```

This example retrieves the all attributes from an audio file:

```
GetContainerAttribute (Example::ContainerField; "all" )
// result =
    all: [General]
    Filename: 04 Since I've Been Loving You.m4a
    Storage Type: Embedded
    MD5: 5992BBD29308F52967E747B2EFFE9AA4
    File Size: 7182880
    Internal Size: 7182941
    External Size: 0
    External Files: 0

    [Audio]
    Title: Since I've Been Loving You
    Artist: Led Zeppelin
    Album: Led Zeppelin III
    Year: 1970
    Track: 4/10
    Genre: Rock
    Composer:
    Cover Art:
    Duration: 0:07:23
    Bit Rate: 128 kbps
```

GetHeight

The `GetHeight` function returns the height in pixels of an image in a container.

Format	<code>GetHeight (sourceField)</code>
Parameters	<code>sourceField</code> = a container field reference.
Result	A number indicating the height of the image provided in pixels.

This example shows the function returning the height of an image stored in a container field:

```
GetHeight ( Example::ContainerField ) // result = 3264
```

GetThumbnail

The `GetThumbnail` function creates data representing a thumbnail image of the content in a container field to a specified height and width.

Format	<code>GetThumbnail (sourceField ; fitToWidth ; fitToHeight)</code>
Parameters	<p><code>sourceField</code> = a container field reference.</p> <p><code>fitToWidth</code> = a number, smaller than the original image, specifying the desired width for the thumbnail.</p> <p><code>fitToHeight</code> = a number, smaller than the original image specifying the desired height for the thumbnail.</p>
Result	Data representing a thumbnail of an image that can be placed into a container field. When both a height and width are specified, the proportion of the image will be automatically maintained favoring the height.

This example shows the function scaling an image from a container field. Notice that the `fitToWidth` measurement is ignored to preserve the proportions of the image, favoring the `fitToHeight` specification.

```
GetThumbnail ( Example::ContainerField ; 2000 ; 500 )
// result = a thumbnail of the image scaled down to 375 x 500
```

This example shows the function scaling an image to half the height and width, which are both dynamically calculated using the `GetWidth` and `GetHeight` functions:

```
GetThumbnail (
  Example::ContainerField ;
  GetWidth (Example::ContainerField) / 2 ;
  GetHeight (Example::ContainerField) / 2
)
// result = a thumbnail of the image scaled to 50%
```

GetWidth

The `GetWidth` function returns the width in pixels of an image in a container.

Format	<code>GetWidth (sourceField)</code>
Parameters	<code>sourceField</code> = a container field reference.
Result	A number indicating the width of the image provided in pixels.

This example shows the function calculating the width of an image stored in a container field:

```
GetWidth ( Example::ContainerField ) // result = 2448
```

HexEncode

The HexEncode function will encode text or container data into hexadecimal (base-16) format.

Format	HexEncode (data)
Parameters	data = a text value that will be encoded.
Result	A text value containing the original data encoded into hexadecimal format.

This example shows the function converting a text string into hexadecimal format:

```
HexEncode ( "Hello World" ) // result = 48656C6C6F20576F726C64
```

HexDecode

The HexDecode function will decode text from hexadecimal (base-16) format back into regular text or container data.

Format	When converting text encoded as hexadecimal, only the text being decoded need be included: Base64Decode (text) To specify a file name with an extension when decoding a file for insertion into a container being, add a second parameter: Base64Decode (text ; fileNameWithExtension)
Parameters	data = a text value that was previously encoded into hexadecimal format. fileNameWithExtension = (optional) a text value that contains a file name and extension for the file being decoded.
Result	Text or container data.

This example shows the function converting a hexadecimal string back into its original decoded format:

```
HexDecode ( "48656C6C6F20576F726C64" ) // result = "Hello World"
```

TextEncode

The TextEncode function encodes text, converting it into a text file that can be placed into a container field and then exported.

Format	TextEncode (text ; encoding ; lineEndings)
Parameters	text = a text value that will be encoded. encoding = the name of a character encoding scheme to use. lineEndings = a numeric value indicating the line endings to use.
Options for encoding	The following options can be used as the encoding parameter: utf-8 — for Unicode encoding iso-8859-1 — for Latin 1 encoding windows-1251 — for Cyrillic encoding shift_jis — for Japanese encoding windows-1252 — for Windows encoding gb18030 — for Simplified Chinese encoding euc-kr — for Korean encoding big5 — for Traditional Chinese encoding macintosh — for Mac Roman encoding
Options for lineEndings	The following options can be used as the lineEndings parameter: 1 — leave line endings unchanged 2 — use carriage return (legacy Macintosh standard) 3 — use line feed (Unix/Linux/MacOS standard) 4 — use carriage return followed by a line feed (Windows standard)
Result	The text encoded as a text file for placement into a container field. The name of the file will be the encoding option with an extension of .txt.

When this example is used as a calculation field resulting in container data or when the result is placed into a container field by script, the result will be a text file named macintosh.txt:

```
TextEncode ( "Hello World" ; "macintosh" ; 3 )
```

TextDecode

The TextDecode function decodes a text file in a container field, converting it into a plain text that can be placed into a field.

Format	TextDecode (container ; encoding)
Parameters	container = a container field reference. encoding = the name of a character encoding scheme to use (see TextEncode for list of available options).
Result	The text contents of the file in the specified container field.

When a text file containing the phrase “Hello World” is stored in a container field named Example Container:

```
TextDecode ( Sandbox::Example Container ; "macintosh" ) // result = "Hello World"
```

VerifyContainer

The VerifyContainer function determines if the data in a container field was changed or deleted outside of FileMaker.

Format	VerifyContainer (sourceField)
Parameters	sourceField = a container field reference.
Result	A value indicating the status of the container field’s contents: 1 = no changes or deletions have occurred in the original file outside of FileMaker 0 = the external file was modified or deleted ? = the field is not a container field

This example shows the function confirming that the data container field has not been modified externally.

```
VerifyContainer ( Example::ContainerField ) // result = 1
```

Summary

In this chapter, we explored all the built-in functions for working with container data.

CHAPTER 19



Aggregate, Summary, and Repeating Functions

There are three categories of functions that handle the manipulation of multiple values.

Aggregate Function Reference

Aggregate functions are used to summarize fields and to perform statistical computations on numbers, dates, and times.

FileMaker provides the following built-in aggregate functions:

- Average
- Count
- List
- Max
- Min
- Steve
- StDevP
- Sum
- Variance
- VarianceP

These functions can accept a single repeating field or a list of non-repeating, repeating, or related fields. Due to the similarity in behavior and the number of possibly different combinations of input, all the examples use the same data as shown in Figure 19-1.

The screenshot shows a window titled "Aggregate Functions" with a subtitle "Aggregate Function Example Data". It contains two tables:

LOCAL EXAMPLE FIELDS

NonRepeating1	1	Repeating1	4	Repeating2	9	Repeating3	14
NonRepeating2	2		5		10		15
NonRepeating3	3		6		11		16
			7		12		17
			8		13		18

RELATED EXAMPLE FIELDS

NonRepeating			Repeating									
1	2	3	1					2				
19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53	54	55	56	57

Figure 19-1. A set of fields, local and related, with simple values that are used for all the examples in this section

Average

The Average function calculates the average of all values in one or more fields.

Format	A single repeating or related numeric field can be specified with one parameter: Average (field) Multiple numeric fields (repeating or not) or numeric values can be used, with a semicolon delimiter: Average (field1 ; field2 ; field3 ; etc.)
Parameters	field = numeric values.
Result	A number representing the average of the values specified, varying based on the combination of literal values, non-repeating fields, repeating fields, or related fields included in the parameter(s).

Average Examples

The examples in this section assume the data shown in Figure 19-1.

Using the Average Function with Local Fields

A list of non-repeating local fields will return the average based on all the values contained within the fields provided:

```
Average ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = 2
```

A *single repeating local field* will return the average based on all the values contained within that field:

```
Average ( Example::Repeating1 )
// result = 6
```

A *list of repeating local fields* used in a *non-repeating calculation* will return the average based on the first repetition from each field provided:

```
Average ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 9
```

A *list of repeating local fields* used in a *repeating calculation* returns the average for each repetition of the result based on the corresponding repetition from each field provided:

```
Average ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 9
// repetition 2 = 10
// repetition 3 = 11
// repetition 4 = 12
// repetition 5 = 13
```

Using the Average Function with Related Fields

A *single related, non-repeating field* will return the average based on all the values from that field for each record related to the local record performing the calculation:

```
Average ( Related::NonRepeating1 )
// result = 32
```

A *list of related, non-repeating fields* returns the average based on the values from the fields provided for the first related record:

```
Average ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = 20
```

A *single related, repeating field* will return the average based on the values from every repetition from every record related to the local record performing the calculation:

```
Average ( Related::Repeating1 )
// result = 37
```

A *list of related, repeating fields* used in a *non-repeating calculation* will return the average based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
Average ( Related::Repeating1 ; Related::Repeating2 )
// result = 24.5
```


A list of *related, repeating fields* used in a *repeating calculation* will return the average based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
Average ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 24.5
// repetition 2 = 25.5
// repetition 3 = 26.5
// repetition 4 = 27.5
// repetition 5 = 28.5
```

Count

The Count function counts the number of values in one or more fields.

Format	A single repeating or related field can be specified with one parameter: Count (field) Multiple fields (repeating or not) or values can be used, with a semicolon delimiter: Count (field1 ; field2 ; field3 ; etc.)
Parameters	field = any text values.
Result	A number: the total count of the values provided, varying based on the combination of literal values, non-repeating fields, repeating fields, or related fields included in the parameter(s).

Count Examples

The examples in this section assume the data shown in Figure 19-1.

■ **Caution** Although all the examples below count lists of *numeric values*, remember that the Count function can count *any* text values. For example, the result of counting "8¶10¶2" and "Mon¶Tues¶Wed" are the same: 3.

Using the Count Function with Local Fields

A list of *non-repeating local fields* will return the total non-blank value count based on all the values contained within the fields provided:

```
Count ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = 3
```

A *single repeating local field* will return the total non-blank value count based on all the values contained within that field:

```
Count ( Example::Repeating1 )
// result = 5
```

A list of *repeating local fields* used in a *non-repeating calculation* will return the total non-blank value count based on the first repetition from each field provided:

```
Count ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 3
```

A list of *repeating local fields* used in a *repeating calculation* return the total non-blank value count for each repetition of the result based on the corresponding repetition from each field provided:

```
Count ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 3
// repetition 2 = 3
// repetition 3 = 3
// repetition 4 = 3
// repetition 5 = 3
```

Using the Count Function with Related Fields

A *single related, non-repeating field* will return the total non-blank value count based on all the values from that field for each record related to the local record performing the calculation:

```
Count ( Related::NonRepeating1 )
// result = 3
```

A list of *related, non-repeating fields* returns the total non-blank value count based on the values from the fields provided for the first related record:

```
Count ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = 3
```

A *single related, repeating field* will return the total non-blank value count based on the values from every repetition from every record related to the local record performing the calculation:

```
Count ( Related::Repeating1 )
// result = 15
```

A list of *related, repeating fields* used in a *non-repeating calculation* will return the total non-blank value count based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
Count ( Related::Repeating1 ; Related::Repeating2 )
// result = 2
```

A list of *related, repeating fields* used in a *repeating calculation* will return the total non-blank value count based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
Count ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 2
// repetition 2 = 2
// repetition 3 = 2
// repetition 4 = 2
// repetition 5 = 2
```

List

The `List` function generates a carriage return-delimited list of values for one or more fields.

Format	When listing values from a single repeating field or a single related field, only one parameter is required: <code>List (field)</code> Optionally, multiple fields can be added with semicolon delimiters: <code>List (field1 ; field2 ; field3 ; etc)</code>
Parameters	<code>field</code> = a field reference.
Result	A carriage return-delimited text-based list of values from the specified field(s), varying based on the combination of non-repeating, repeating, or related fields included in the parameter(s).

List Examples

The examples in this section assume the data shown in Figure 19-1.

Using the List Function with Local Fields

A *list of non-repeating local fields* will return a carriage return-delimited list based on all the values contained within the fields provided:

```
List ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = 1¶2¶3
```

A *single repeating local field* will return a carriage return-delimited list based on all the values contained within that field:

```
List ( Example::Repeating1 )
// result = 4¶5¶6¶7¶8
```

A *list of repeating local fields used in a non-repeating calculation* will return a carriage return-delimited list based on the first repetition from each field provided:

```
List ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 4¶9¶14
```

A *list of repeating local fields used in a repeating calculation* return a carriage return-delimited list for each repetition of the result based on the corresponding repetition from each field provided:

```
List ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 4¶9¶14
// repetition 2 = 5¶10¶15
// repetition 3 = 6¶11¶16
// repetition 4 = 7¶12¶17
// repetition 5 = 8¶13¶18
```

Using the List Function with Related Fields

A *single related, non-repeating field* will return a carriage return-delimited list based on all the values from that field for each record related to the local record performing the calculation:

```
List ( Related::NonRepeating1 )
// result = 19¶32¶45
```

A *list of related, non-repeating fields* return a carriage return-delimited list based on the values from the fields provided for the first related record:

```
List ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = 19¶20¶21
```

A *single related, repeating field* will return a carriage return-delimited list based on the values from every repetition from every record related to the local record performing the calculation:

```
List ( Related::Repeating1 )
// result = 22¶23¶24¶25¶26¶35¶36¶37¶38¶39¶48¶49¶50¶51¶52
```

A *list of related, repeating fields* used in a *non-repeating calculation* will return a carriage return-delimited list based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
List ( Related::Repeating1 ; Related::Repeating2 )
// result = 22¶27
```

A *list of related, repeating fields* used in a *repeating calculation* will return a carriage return-delimited list based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
List ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 22¶27
// repetition 2 = 23¶28
// repetition 3 = 24¶29
// repetition 4 = 25¶30
// repetition 5 = 26¶31
```

Max

The Max function extracts the highest value from a list of values.

Format	A single repeating or related numeric field can be specified with one parameter: Max (field) Multiple numeric fields (repeating or not) or numeric values can be used, with a semicolon delimiter: Max (field1 ; field2 ; field3 ; etc)
Parameters	field = numeric values.
Result	A number representing the highest value from the values provided, varying based on the combination of separate numeric values, non-repeating fields, repeating fields, or related fields included in the parameter(s).

Max Examples

The examples in this section assume the data shown in Figure 19-1.

Using the Max Function with Local Fields

A *list of non-repeating local fields* will return the highest value based on all the values contained within the fields provided:

```
Max ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = 3
```

A *single repeating local field* will return the highest value based on all the values contained within that field:

```
Max ( Example::Repeating1 )
// result = 8
```

A *list of repeating local fields* used in a *non-repeating calculation* will return the highest value based on the first repetition from each field provided:

```
Max ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 14
```

A *list of repeating local fields* used in a *repeating calculation* returns the highest value for each repetition of the result based on the corresponding repetition from each field provided:

```
Max ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 14
// repetition 2 = 15
// repetition 3 = 16
// repetition 4 = 17
// repetition 5 = 18
```

Using the Max Function with Related Fields

A *single related, non-repeating field* will return the highest value based on all the values from that field for each record related to the local record performing the calculation:

```
Max ( Related::NonRepeating1 )
// result = 45
```

A *list of related, non-repeating fields* return the highest value based on the values from the fields provided for the first related record:

```
Max ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = 21
```

A *single related, repeating field* will return the highest value based on the values from every repetition from every record related to the local record performing the calculation:

```
Max ( Related::Repeating1 )
// result = 52
```

A list of *related, repeating fields* used in a *non-repeating calculation* will return the highest value based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
Max ( Related::Repeating1 ; Related::Repeating2 )
// result = 27
```

A list of *related, repeating fields* used in a *repeating calculation* will return the highest value based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
Max ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 27
// repetition 2 = 28
// repetition 3 = 29
// repetition 4 = 30
// repetition 5 = 31
```

Min

The Max function extracts the lowest value from a list of values.

Format	A single repeating or related numeric field can be specified with one parameter: Min (field) Multiple numeric fields (repeating or not) or numeric values can be used, with a semicolon delimiter: Min (field1 ; field2 ; field3 ; etc)
Parameters	field = numeric values.
Result	A number representing the lowest value from the values provided, varying based on the combination of separate numeric values, non-repeating fields, repeating fields, or related fields included in the parameter(s).

Min Examples

The examples in this section assume the data shown in Figure 19-1.

Using the Min Function with Local Fields

A list of *non-repeating local fields* will return the lowest value based on all the values contained within the fields provided:

```
Min ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = 1
```

A *single repeating local field* will return the lowest value based on all the values contained within that field:

```
Min ( Example::Repeating1 )
// result = 4
```

A list of repeating local fields used in a non-repeating calculation will return the lowest value based on the first repetition from each field provided:

```
Min ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 4
```

A list of repeating local fields used in a repeating calculation return the lowest value for each repetition of the result based on the corresponding repetition from each field provided:

```
Min ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 4
// repetition 2 = 5
// repetition 3 = 6
// repetition 4 = 7
// repetition 5 = 8
```

Using the Min Function with Related Fields

A single related, non-repeating field will return the lowest value based on all the values from that field for each record related to the local record performing the calculation:

```
Min ( Related::NonRepeating1 )
// result = 19
```

A list of related, non-repeating fields returns the lowest value based on the values from the fields provided for the first related record:

```
Min ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = 19
```

A single related, repeating field will return the lowest value based on the values from every repetition from every record related to the local record performing the calculation:

```
Min ( Related::Repeating1 )
// result = 22
```

A list of related, repeating fields used in a non-repeating calculation will return the lowest value based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
Min ( Related::Repeating1 ; Related::Repeating2 )
// result = 22
```

A list of related, repeating fields used in a repeating calculation will return the lowest value based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
Min ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 22
// repetition 2 = 23
// repetition 3 = 24
// repetition 4 = 25
// repetition 5 = 26
```

StDev

The StDev function calculates the standard deviation of a list of values. Standard deviation is a measure of how spread out a set of numbers are from their mean, which is calculated by taking the square root of the average of the squared deviations of the values from their average value.

Format	A single repeating or related numeric field can be specified with one parameter: StDev (field) Multiple numeric fields (repeating or not) or numeric values can be used, with a semicolon delimiter: StDev (field1 ; field2 ; field3 ; etc)
Parameters	field = numeric values.
Result	A number representing the standard deviation from the values provided, varying based on the combination of separate numeric values, non-repeating fields, repeating fields, or related fields included in the parameter(s).
More Info	For background information on standard deviation visit: https://en.wikipedia.org/wiki/Standard_deviation .

StDev Examples

The examples in this section assume the data shown in Figure 19-1.

Using the StDev Function with Local Fields

A list of *non-repeating local fields* will return the standard deviation based on all the values contained within the fields provided:

```
StDev ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = 1
```

A *single repeating local field* will return the standard deviation based on all the values contained within that field:

```
StDev ( Example::Repeating1 )
// result = 1.5811388300841897
```

A list of *repeating local fields* used in a *non-repeating calculation* will return the standard deviation based on the first repetition from each field provided:

```
StDev ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 5
```


A *list of repeating local fields* used in a *repeating calculation* return the standard deviation for each repetition of the result based on the corresponding repetition from each field provided:

```
StDev ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 5
// repetition 2 = 5
// repetition 3 = 5
// repetition 4 = 5
// repetition 5 = 5
```

Using the StDev Function with Related Fields

A *single related, non-repeating field* will return the standard deviation based on all the values from that field for each record related to the local record performing the calculation:

```
StDev ( Related::NonRepeating1 )
// result = 13
```

A *list of related, non-repeating fields* returns the standard deviation based on the values from the fields provided for the first related record:

```
StDev ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = 1
```

A *single related, repeating field* will return the standard deviation based on the values from every repetition from every record related to the local record performing the calculation:

```
StDev ( Related::Repeating1 )
// result = 11.0840941378690418
```

A *list of related, repeating fields* used in a *non-repeating calculation* will return the standard deviation based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
StDev ( Related::Repeating1 ; Related::Repeating2 )
// result = 3.5355339059327376
```

A *list of related, repeating fields* used in a *repeating calculation* will return the standard deviation based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
StDev ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 3.5355339059327376
// repetition 2 = 3.5355339059327376
// repetition 3 = 3.5355339059327376
// repetition 4 = 3.5355339059327376
// repetition 5 = 3.5355339059327376
```

StDevP

The StDevP function calculates the standard deviation of a population represented by a list of values.

Format	A single repeating or related numeric field can be specified with one parameter: StDevP (field) Multiple numeric fields (repeating or not) or numeric values can be used, with a semicolon delimiter: StDevP (field1 ; field2 ; field3 ; etc)
Parameters	field = numeric values.
Result	A number representing the standard deviation of a population from the values provided, varying based on the combination of separate numeric values, non-repeating fields, repeating fields, or related fields included in the parameter(s).

StDevP Examples

The examples in this section assume the data shown in Figure 19-1.

Using the StDevP Function with Local Fields

A list of *non-repeating local fields* will return the standard deviation of a population based on all the values contained within the fields provided:

```
StDevP ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = .816496580927726
```

A *single repeating local field* will return the standard deviation of a population based on all the values contained within that field:

```
StDevP ( Example::Repeating1 )
// result = 1.414213562373095
```

A list of *repeating local fields* used in a *non-repeating calculation* will return the standard deviation of a population based on the first repetition from each field provided:

```
StDevP ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 4.0824829046386302
```

A list of *repeating local fields* used in a *repeating calculation* return the standard deviation of a population for each repetition of the result based on the corresponding repetition from each field provided:

```
StDevP ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 4.0824829046386302
// repetition 2 = 4.0824829046386302
// repetition 3 = 4.0824829046386302
// repetition 4 = 4.0824829046386302
// repetition 5 = 4.0824829046386302
```

Using the StDevP Function with Related Fields

A *single related, non-repeating field* will return the standard deviation of a population based on all the values from that field for each record related to the local record performing the calculation:

```
StDevP ( Related::NonRepeating1 )
// result = 10.614455520604384
```

A *list of related, non-repeating fields* return the standard deviation of a population based on the values from the fields provided for the first related record:

```
StDevP ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = .816496580927726
```

A *single related, repeating field* will return the standard deviation of a population based on the values from every repetition from every record related to the local record performing the calculation:

```
StDevP ( Related::Repeating1 )
// result = 10.7082522694726737
```

A *list of related, repeating fields* used in a *non-repeating calculation* will return the standard deviation of a population based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
StDevP ( Related::Repeating1 ; Related::Repeating2 )
// result = 2.5
```

A *list of related, repeating fields* used in a *repeating calculation* will return the standard deviation of a population based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
StDevP ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 2.5
// repetition 2 = 2.5
// repetition 3 = 2.5
// repetition 4 = 2.5
// repetition 5 = 2.5
```

Sum

The Sum function adds a series of numbers into a total.

Format	A single repeating or related numeric field can be specified with one parameter: Sum (field) Multiple numeric fields (repeating or not) or numeric values can be used, with a semicolon delimiter: Sum (field1 ; field2 ; field3 ; etc)
Parameters	field = numeric values.
Result	A number representing the total of the values provided, varying based on the combination of separate numeric values, non-repeating fields, repeating fields, or related fields included in the parameter(s).

Sum Examples

The examples in this section assume the data shown in Figure 19-1.

Using the Sum Function with Local Fields

A *list of non-repeating local fields* will return the sum total based on all the values contained within the fields provided:

```
Sum ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = 6
```

A *single repeating local field* will return the sum total based on all the values contained within that field:

```
Sum ( Example::Repeating1 )
// result = 30
```

A *list of repeating local fields* used in a *non-repeating calculation* will return the sum total based on the first repetition from each field provided:

```
Sum ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 27
```

A *list of repeating local fields* used in a *repeating calculation* returns the sum total for each repetition of the result based on the corresponding repetition from each field provided:

```
Sum ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 27
// repetition 2 = 30
// repetition 3 = 33
// repetition 4 = 36
// repetition 5 = 39
```

Using the Sum Function with Related Fields

A *single related, non-repeating field* will return the sum total based on all the values from that field for each record related to the local record performing the calculation:

```
Sum ( Related::NonRepeating1 )
// result = 96
```

A *list of related, non-repeating fields* returns the sum total based on the values from the fields provided for the first related record:

```
Sum ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = 60
```

A *single related, repeating field* will return the sum total based on the values from every repetition from every record related to the local record performing the calculation:

```
Sum ( Related::Repeating1 )
// result = 555
```

A *list of related, repeating fields* used in a *non-repeating calculation* will return the sum total based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
Sum ( Related::Repeating1 ; Related::Repeating2 )
// result = 49
```

A *list of related, repeating fields* used in a *repeating calculation* will return the sum total based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
Sum ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 49
// repetition 2 = 51
// repetition 3 = 53
// repetition 4 = 55
// repetition 5 = 57
```

Variance

The Variance function calculates the variance of a series of numbers. Variance is a measure of how varied a distribution of numbers is from its mean.

Format	A single repeating or related numeric field can be specified with one parameter: Variance (field) Multiple numeric fields (repeating or not) or numeric values can be used, with a semicolon delimiter: Variance (field1 ; field2 ; field3 ; etc)
Parameters	field = numeric values.
Result	A number representing the variance of the values provided, varying based on the combination of separate numeric values, non-repeating fields, repeating fields, or related fields included in the parameter(s).
More Info	For more general information about variance, visit: https://en.wikipedia.org/wiki/Variance .

Variance Examples

The examples in this section assume the data shown in Figure 19-1.

Using the Variance Function with Local Fields

A *list of non-repeating local fields* will return the variance based on all the values contained within the fields provided:

```
Variance ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = 1
```

A *single repeating local field* will return the variance based on all the values contained within that field:

```
Variance ( Example::Repeating1 )
// result = 2.5
```

A *list of repeating local fields* used in a *non-repeating calculation* will return the variance based on the first repetition from each field provided:

```
Variance ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 25
```

A *list of repeating local fields* used in a *repeating calculation* returns the variance for each repetition of the result based on the corresponding repetition from each field provided:

```
Variance ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 25
// repetition 2 = 25
// repetition 3 = 25
// repetition 4 = 25
// repetition 5 = 25
```

Using the Variance Function with Related Fields

A *single related, non-repeating field* will return the variance based on all the values from that field for each record related to the local record performing the calculation:

```
Variance ( Related::NonRepeating1 )
// result = 169
```

A *list of related, non-repeating fields* returns the variance based on the values from the fields provided for the first related record:

```
Variance ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = 1
```

A *single related, repeating field* will return the variance based on the values from every repetition from every record related to the local record performing the calculation:

```
Variance ( Related::Repeating1 )
// result = 122.8571428571428571
```

A *list of related, repeating fields* used in a *non-repeating calculation* will return the variance based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
Variance ( Related::Repeating1 ; Related::Repeating2 )
// result = 12.5
```

A list of *related, repeating fields* used in a *repeating calculation* will return the variance based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
Variance ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 12.5
// repetition 2 = 12.5
// repetition 3 = 12.5
// repetition 4 = 12.5
// repetition 5 = 12.5
```

VarianceP

The VarianceP function calculates the variance of a population distribution of a series of numbers.

Format	A single repeating or related numeric field can be specified with one parameter: VarianceP (field) Multiple numeric fields (repeating or not) or numeric values can be used, with a semicolon delimiter: VarianceP (field1 ; field2 ; field3 ; etc)
Parameters	field = numeric values.
Result	A number representing the variance of a population distribution of the values provided, varying based on the combination of separate numeric values, non-repeating fields, repeating fields, or related fields included in the parameter(s).

VarianceP Examples

The examples in this section assume the data shown in Figure 19-1.

Using the VarianceP Function with Local Fields

A list of *non-repeating local fields* will return the variance of a population based on all the values contained within the fields provided:

```
VarianceP ( Example::NonRepeating1 ; Example::NonRepeating2 ; Example::NonRepeating3 )
// result = .6666666666666667
```

A *single repeating local field* will return the variance of a population based on all the values contained within that field:

```
VarianceP ( Example::Repeating1 )
// result = 2
```

A list of *repeating local fields* used in a *non-repeating calculation* will return the variance of a population based on the first repetition from each field provided:

```
VarianceP ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// result = 16.666666666666667
```

A *list of repeating local fields* used in a *repeating calculation* returns the variance of a population for each repetition of the result based on the corresponding repetition from each field provided:

```
VarianceP ( Example::Repeating1 ; Example::Repeating2 ; Example::Repeating3 )
// repetition 1 = 16.666666666666667
// repetition 2 = 16.666666666666667
// repetition 3 = 16.666666666666667
// repetition 4 = 16.666666666666667
// repetition 5 = 16.666666666666667
```

Using the VarianceP Function with Related Fields

A *single related, non-repeating field* will return the variance of a population based on all the values from that field for each record related to the local record performing the calculation:

```
VarianceP ( Related::NonRepeating1 )
// result = 112.666666666666667
```

A *list of related, non-repeating fields* returns the variance of a population based on the values from the fields provided for the first related record:

```
VarianceP ( Related::NonRepeating1 ; Related::NonRepeating2 ; Related::NonRepeating3 )
// result = .666666666666667
```

A *single related, repeating field* will return the variance of a population based on the values from every repetition from every record related to the local record performing the calculation:

```
VarianceP ( Related::Repeating1 )
// result = 114.666666666666667
```

A *list of related, repeating fields* used in a *non-repeating calculation* will return the variance of a population based on the values from the first repetition of those fields from the first record related to the local record performing the calculation:

```
VarianceP ( Related::Repeating1 ; Related::Repeating2 )
// result = 6.25
```

A *list of related, repeating fields* used in a *repeating calculation* will return the variance of a population based on the values from the corresponding repetition from each field provided from the first record related to the local record performing the calculation:

```
VarianceP ( Related::Repeating1 ; Related::Repeating2 )
// repetition 1 = 6.25
// repetition 2 = 6.25
// repetition 3 = 6.25
// repetition 4 = 6.25
// repetition 5 = 6.25
```


Summary Functions

Summary functions are used to calculate a summary of values for a field in all records in a found set or to summarize values for records into different subgroups. FileMaker provides one built-in summary function:

- `GetSummary`

GetSummary

The `GetSummary` function returns the value of a summary field for the current found set of records. This allows a calculation formula to perform a similar function to what a summary field can display on a layout (Chapter 8, “Summary Fields”).

Format	<code>GetSummary (summaryField ; breakfield)</code>
Parameters	<code>summaryField</code> = a reference to a summary field. <code>breakField</code> = a reference to a field, which will break a large group of fields down into smaller subgroups if the set of records are sorted by that field.
Result	A number date, time, or timestamp depending on the summary field provided. If the <code>breakField</code> is the same as the <code>summaryField</code> , the summary value will be for all records in the found set. If the fields are different and the found set is sorted by the <code>breakField</code> , the summary will be of records with a matching value.

Assume a table with the fields, records and values, sorted shown in Table 19-1. The `Example Summary` field is defined to be a total summary of the `Example Value` field.

Table 19-1. *Example Records*

Record ID	Example Value (number)	Example City (text)	Example Summary
1	10	Atlanta	50
2	10	Atlanta	50
3	10	New York	50
4	10	New York	50
5	10	New York	50

On a layout, the summary field displays the total for all records unless it is placed within a sub-summary layout part (Chapter 28, “Defining Summary Parts”). Since summary fields only summarize on layouts and are not accessible to calculations, you must create a new calculation field named `Example Summary Calculation` and a formula using the `GetSummary` function in order to access the summary for further manipulation.

In this first example, both parameters are pointing to the `Example Summary` field. The result is the total for all records in the found set, like what would appear on a layout.

```
GetSummary ( Example Summary ; Example Summary ) // result = 50
```

However, in the following example, the `breakField` parameter points to the `Example City` field. The result will now vary from one record to the next. Because it will be sub-summarizing the result based on the `Example City` field, a result of 20 will appear on the first two records and a value of 30 on the last three.

```
GetSummary ( Example Summary ; Example City)
```

Repeating Function Reference

Repeating functions are used to perform calculations involving repeating fields. FileMaker provides the following built-in repeating functions:

- `Extend`
- `GetRepetition`
- `Last`

Extend

The `Extend` function allows a repeating and a non-repeating field to be used together within a repeating calculation. The non-repeating field is “extended” so that it can be used or compared with each repetition in a repeating field as if it were a repeating field (Chapter 8, “Repeating”).

Format	<code>Extend (non-repeatingField)</code>
Parameters	<code>non-repeatingField</code> = a reference to a non-repeating field or an expression that returns a reference to a non-repeating field.
Result	The contents of the field for use in a calculation with repeating fields.

This example assumes an `Invoice` database where a record’s `SubTotalCost` field contains 100.00. This single amount must be combined with the cost of five different shipping methods to arrive at five different grand totals that are offered to a customer. Each shipping cost is stored in one cell of a repeating field named `ShippingCost` with values of 5, 28, 15, 40, and 13. The formula and results below are for a `GrandTotal` calculation field that has five repetitions. Since the `SubTotalCost` field is extended, its single value will be successfully added to all five values in `ShippingCost` to arrive at five results.

```
Extend ( Invoice::SubTotalCost ) + Invoice::ShippingCost
// repetition 1 = 105.00
// repetition 2 = 128.00
// repetition 3 = 115.00
// repetition 4 = 140.00
// repetition 5 = 113.00
```

GetRepetition

The `GetRepetition` function returns the contents of a single repetition of a repeating field.

Format	<code>GetRepetition (repeatingField ; number)</code>
Parameters	<code>repeatingField</code> = a reference to a repeating field. <code>number</code> = a number indicating the specific repetition desired.
Result	The contents of the specified repetition of the field.

These examples show the different results pulled from different repetitions of the same field. It assumes an Invoice database has a `Line Item Qty` field with five repetitions containing the following values: 5, 28, 15, 40, and 13:

```
GetRepetition ( Invoice::Line Item Qty ; 1 ) // result = 5
GetRepetition ( Invoice::Line Item Qty ; 2 ) // result = 28
GetRepetition ( Invoice::Line Item Qty ; 3 ) // result = 15
GetRepetition ( Invoice::Line Item Qty ; 4 ) // result = 40
GetRepetition ( Invoice::Line Item Qty ; 5 ) // result = 13
```

Last

The `Last` function returns the last non-blank value from a repeating field.

Format	<code>Last (repeatingField)</code>
Parameters	<code>repeatingField</code> = a reference to a repeating field.
Result	The value from the last repetition that contains a value from the specified field.

This example shows the result when an invoice database has a `Line Item Qty` field with 5 repetitions containing the following three values: 5, 28, and 15:

```
Last ( Invoice::Line Item Qty ) // result = 15
```

Summary

In this chapter, we explored all the built-in functions for performing aggregate, summary, and repeating calculations.

CHAPTER 20



Financial and Trigonometric Functions

These two categories of functions handle financial and advanced mathematical calculations.

Financial Function Reference

Financial functions are used to calculate financial results for investment and interest payments. FileMaker provides the following built-in financial functions:

- FV
- NPV
- PMT
- PV

FV

The FV function returns the *future value* of an investment, based on a fixed interest rate and payment amount for a specified period of months.

Format	FV (payment ; interestRate ; periods)
Parameters	payment = a number indicating the amount of payment to be made each period. interestRate = a number representing the interest rate per period. periods = a number indicating the total periods over which the investment will be repaid.
Result	A number calculated as the total amount to be paid over the entire load period.
More Info	For more general information about future value, visit: https://en.wikipedia.org/wiki/Future_value .

This example shows the total payment on a loan of \$1,000 that will be paid back over 10 months with an interest rate of 2.75%, rounded to two decimal places:

```
Round ( FV ( 100 ; .0275 ; 10 ) ; 2 ) // result = 1133.28
```

NPV

The NPV function returns the *net present value* of a series of unequal payments that are made at regular intervals at a fixed rate of interest.

Format	NPV (payment ; interestRate)
Parameters	payment = a reference to a repeating field. interestRate = a number indicating the interest rate charged at regular intervals.
Result	A number indicating the total amount of profit to be made from this transaction.
More Info	For more general information about net present value, visit: https://en.wikipedia.org/wiki/Net_present_value .

This example assumes that the repeating field, Payments contains -5000 (the loan amount), 1000, 1200, 500, 500, 3000.

```
NPV ( Example::Payments ; .05 ) // result = 223.49
```

PMT

The PMT function returns the payment required by a loan amount and interest rate for a given term.

Format	PMT (principal ; interestRate ; term
Parameters	principal = a number indicating the amount of the loan principal. interestRate = a number indicating in the rate of interest. term = a number indicating the loan term in months.
Result	A number indicating the amount of monthly payment to pay off the loan plus interest.

A 30-month loan for \$10,000 at an interest rate of 5% would require a payment calculated as follows:

```
Round ( PMT ( 10000 ; .05 ; 30 ) ; 2 ) // result = 650.51
```

If the interest is an annual rate, divide it by 12, as shown in this example:

```
Round ( PMT ( 10000 ; ( .25 / 12 ) ; 30 ) ; 2 ) // result = 451.63
```

PV

The PV function returns the present value, or present discounted value, of a loan, which is an expectation of the value of income as of the current date.

Format	PV (payment ; interestRate ; periods)
Parameters	payment = a number indicating the amount of a payment that is made per period. interestRate = a number indicating the interest rate per period. periods = a number indicating the number of total periods.
Result	A number representing the present value of the loan.
More Info	For more general information on present value, visit: https://en.wikipedia.org/wiki/Present_value .

This example shows a calculation of the present value of a loan with 30 payments of \$350 each at a 5% interest rate, rounded to two digits:

```
Round ( PV ( 350 ; .05 ; 30 ) ; 2 ) // result = 5380.36
```

Trigonometric Function Reference

Trigonometric functions are used to perform geometric calculations. FileMaker provides the following built-in trigonometric functions:

- Acos
- Asin
- Atan
- Cos
- Degrees
- Pi
- Radians
- Sin
- Tan

ACOS

The Acos function returns the arccosine, or inverse cosine, of a number.

Format	Acos (number)
Parameters	number = a number ranging from -1 to 1.
Result	A number representing an angle in radians from 0 (zero) to Pi.

This example shows the function calculating the arccosine of .75:

```
Acos ( .75 ) // result = .7227342478134157
```

Asin

The `Asin` function returns the arcsine, or inverse sine, of a number.

Format	<code>Asin (number)</code>
Parameters	<code>number</code> = a number ranging from -1 to 1.
Result	A number representing an angle in radians from $\text{Pi}/2$ to $\text{Pi}/2$.

This example shows the function calculating the arcsine of .75:

```
Asin ( .75 ) // result = .848062078981481
```

Atan

The `Atan` function returns the arc tangent, or inverse tangent, of a number.

Format	<code>Atan (number)</code>
Parameters	<code>number</code> = a number.
Result	A number representing the angle, in radians, whose tangent is equal to the number provided.

This example shows the function calculating the arc tangent of .5:

```
Atan ( .5 ) // result = .4636476090008061
```

Cos

The `Cos` function returns the cosine of an angle.

Format	<code>Cos (angleInRadians)</code>
Parameters	<code>angleInRadians</code> = a number expressed in radians.
Result	The cosine of the angle provided.

This example shows the function calculating the cosine of .95:

```
Cos ( .95 ) // result = .5816830894638836
```

Degrees

The `Degrees` function converts an angle to degrees.

Format	<code>Degrees (angleInRadians)</code>
Parameters	<code>angleInRadians</code> = a number expressed in radians.
Result	A number representing the radians provided expressed in degrees.

This example shows the function converting an angle of .95 into degrees:

```
Degrees ( .95 ) // result = 54.4309905374282048
```

Pi

The Pi function returns the value of the constant Pi.

Format	Pi
Parameters	Not applicable
Result	The constant Pi to 16 decimal places.

This example shows the function returning the Pi constant:

```
Pi // result = 3.1415926535897932
```

Radians

The Radians function converts an angle expressed in degrees into radians.

Format	Radians (angleInDegrees)
Parameters	angleInDegrees = a number expressed in degrees.
Result	A number representing the degrees provided expressed in radians.

This example shows the function converting a 90-degree angle into radians:

```
Radians ( 90 ) // result = 1.5707963267948966
```

Sin

The Sin function returns the sine of an angle expressed in radians.

Format	Sin (angleInRadians)
Parameters	angleInRadians = a number expressing an angle in radians.
Result	The sine of the number provided.

This example shows the function calculating the sine of an angle:

```
Sin ( .75 ) // result = .6816387600233342
```


Tan

The Tan function returns the tangent of an angle expressed in radians.

Format	Tan (angleInRadians)
Parameters	angleInRadians = a number expressing an angle in radians.
Result	The tangent of the number provided.

This example shows the function calculating the tangent of an angle:

```
Tan ( 60 ) // result = .3200403893795629
```

Summary

In this chapter, we explored all the built-in functions for performing financial and trigonometric calculations.

CHAPTER 21



Logical Functions

Logical functions are used to test conditions to produce variable results and perform other evaluative functions.

Logical Function Reference

FileMaker provides the following built-in logical functions:

- Case
- Choose
- Evaluate
- EvaluationError
- ExecuteSQL
- GetAsBoolean
- GetAVPlayerAttribute
- GetField
- GetFieldName
- GetLayoutObjectAttribute
- GetNthRecord
- If
- IsEmpty
- IsValid
- IsValidExpression
- Let
- Lookup
- LookupNext
- Self

Case

The Case function evaluates one or more expressions, in order, and returns a result corresponding to the first expression to return a true result. If none of the expressions are true, an optional default result will be returned. Case is somewhat like an if-then function but can handle multiple statements with less verbiage.

Format With a single test-result pair and no default result, only two parameters are required:
 Case (test1 ; result1)
 Add additional test-result pairs as needed:
 Case (test1 ; result1 ; test2 ; result2)
 A default result can be included, which will be used if none of the other test expressions produce a result:
 Case (test1 ; result1 ; test2 ; result2 ; defaultResult)
 As these statements grow, consider adding carriage returns to reformat them vertically, putting each pair on its own line and making the statement easier to read:
 Case (
 test1 ; result1 ;
 test2 ; result2 ;
 test3 ; result3 ;
 test4 ; result4 ;
 defaultResult
)

Parameters test = a text or numeric expression that evaluates to a Boolean.
 result = a literal result or expression that generates a result that will correspond to the paired test.
 defaultResult = (optional) a result that will be used if none of the provided tests are true.

Result A result provided that corresponds to the true test in whatever data type is generated.

This simple example uses the number of days from a field named "Elapsed" to determine how overdue a payment has become.

```
Case (  
  Elapsed > 90 ; "Severe" ;  
  Elapsed > 60 ; "Delinquent" ;  
  Elapsed > 30 ; "Past Due" ;  
  "On Time"  
)
```

The result of the calculation will vary depending on the number in that field.

```
// when Elapsed = 20, the result = "On Time"  
// when Elapsed = 32, the result = "Past Due"  
// when Elapsed = 64, the result = "Delinquent"
```

Choose

The Choose function evaluates a single test condition to generate an integer that is used to select from a list of one or more provided results. The results are zero-based, so the first result is returned if the test condition evaluates to a 0, the second result for a 1, and so on.

Format	<p>If there is only one possible result, only two parameters are required: Choose (test ; result0)</p> <p>Additional results can be added with a semicolon delimiter: Choose (test ; result0 ; result1 ; result2 ; etc.)</p> <p>As these statements grow, consider adding carriage returns to reformat them vertically, putting each pair of conditions on its own line and making the statement easier to read: Choose (test1 ; result0 ; result1 ; result2 ; result3)</p>
Parameters	<p>test = a number of expression that evaluates to an integer. result = a literal result or expression.</p>
Result	The result located at the position indicated by the integer generated by the test expression.

This simple example uses a value from a field named "Status" to determine how overdue a payment has become.

```
Choose ( Status ; "On Time" ; "Past Due" ; "Delinquent" ; "Severe" )
```

The result of the above calculation will vary depending on the number contained in the Status field, as shown here:

```
// when Status = 0, the result = "On Time"
// when Status = 1, the result = "Past Due"
// when Status = 2, the result = "Delinquent"
// when Status = 3, the result = "Severe"
```

Evaluate

The Evaluate function converts any text-based expression into a calculation and then executes it to generate a result. Creative use of this function gives developers a powerful ability to create variable functionality in the form of dynamic calculations that can be built automatically by the database and executed as needed.

Format	<p>Without the optional field parameter(s): Evaluate (expression)</p> <p>When used as the formula for a calculation field, use the optional list of fields to control when the function is triggered. A change to any of the fields listed will cause the function to evaluate the expression. They should be added as a second parameter as a list of semicolon delimited field references that are enclosed in square brackets: Evaluate (expression ; [field1 ; field2 ; etc])</p>
Parameters	<p>expression = a text value. field = (optional) a list of field references that should trigger the expression to evaluate when a change is made to any of them.</p>
Result	The result of the evaluated expression.

This shows a simple math equation being evaluated. Notice that the formula is a text string and not a compiled equation.

```
Evaluate ( "5 + 5" ) // result = 10
```

This example shows a formula pulled from a field named CalcFormula that will execute anytime the First Name or Last Name fields are modified.

```
Evaluate ( Contacts::CalcFormula ; [ Contacts::First Name ; Contacts::Last Name ] )
```

One use of this function is to create dynamic variables with a variable name based on a text value pulled from a specific record. In this example, an invoice contains a company code, which is used as the name of a new variable that will store the amount due:

```
Evaluate (
  "Let [ $" & Invoice::CompanyCode & " = " &
  Quote ( Invoice::AmountDue ) & " ] ;
  " & Quote ( "" ) & " )"
)
```

The above example demonstrates just how quickly even a simple use of this function can become confusing. To avoid this, write a compiled formula as an example of what functions the dynamic one will perform and then work backwards, converting it into a text string and evaluating that to see if the result is the formula you started with. Then, wrap it in an Evaluate function and make generic any hard-coded values it may contain and it will be ready for use. The text expression above is dynamically generating the code below (with spaces added for clarity), inserting a field value for the variable name and the amount, and then evaluating it.

```
Let [
  $$WriteTrackMedia = "$15.00"
] ;
""
)
```

EvaluationError

The EvaluationError function returns an code if an error is generated by an Evaluate function call. A script can use this function to first test an expression and report errors more intelligently to a user.

Format	EvaluationError (expression)
Parameters	expression = a text expression or text field containing a call to the Evaluate function.
Result	a number representing the error that occurred when evaluating the expression or a 0 (zero) if there are no errors.

This example attempts to evaluate a text-based Get (Current Date) function that is missing the closing parenthesis and will, therefore, generate an error:

```
EvaluationError ( Evaluate ( "Get ( CurrentDate " ) )
// result = 1207
```

■ **Note** Error 1207 indicates an error of “Unbalanced Parenthesis.” For a complete list of errors, visit http://www.filemaker.com/help/14/fmp/en/html/error_codes.html

When the formula in the text string is corrected, the function confirms it by returning zero:

```
EvaluationError ( Evaluate ( "Get ( CurrentDate )" ) )
// result = 0
```

ExecuteSQL

The ExecuteSQL function performs an SQL query statement in a FileMaker database. This function is discussed in detail in Chapter 26.

GetAsBoolean

The GetAsBoolean function converts an expression into a Boolean based on it evaluating to true or false.

Format	GetAsBoolean (data)
Parameters	data = an expression or field containing an expression.
Result	A Boolean: 1 = the expression provided evaluated to true 0 = the expression provided evaluated to false

These examples show the function operating on different input:

```
GetAsBoolean ( 10 = 10 )      // result = 1
GetAsBoolean ( 10 > 20 )     // result = 0
```

GetAVPlayerAttribute

The GetAVPlayerAttribute function returns one or more attributes for the audio, video, or image file being played within a container field on FileMaker Go.

Format	GetAVPlayerAttribute (attributeName)			
Parameters	attributeName = a text string or text expression resulting in the name of the desired attribute.			
Options for attributeName	The attributeName parameter can contain any of the following values:			
	all	statOffset	result	volume
	sourceType	endOffset	hideControls	pictureInPicture
	source	duration	disableInteraction	externalPlayback
	playbacksState	triggerEvent	disableExternalControls	imageSourceType
	presentation	triggerEventDetail	pauseInBackground	imageSource
	position	sequence	zoom	imageDuration
Result	A text string with the attribute(s) requested.			

Requesting all attributes returns a nice, readable list of every attribute in a human-readable format:

```
GetAVPlayerAttribute ( "all" )
// result =
  [AVPlayer Attributes]

  Source Type: None
  Source:
  Playback State: Stopped
  Presentation: Embedded
  Position: 0.000000 seconds
  Start Offset: 0.000000 seconds
  End Offset: 0.000000 seconds
  Duration: 0.000000 seconds
  Trigger Event: Internal
  Trigger Event Detail: Unknown
  Sequence: None
  Result: Success
  Hide Controls: 0 (No)
  Disable Interaction: 0 (No)
  Disable External Controls: 0 (No)
  Pause In Background: 0 (No)

  Image Source Type: None
  Image Source:
  Image Duration: 0.000000 seconds
```

However, when requesting a single attribute, it returns a code indicating the result.

```
GetAVPlayerAttribute ( "playbackState" ) // result = 0
```

FileMaker's help documentation includes a full list of these codes and their readable translation. However, another option is to request all attributes and then parse out the information desired using several other functions, as shown in this example:

```
Let ( [
  values = GetAVPlayerAttribute ( "all" );
  attribute = "Playback State" ;
  start = Position ( values ; attribute ; 1 ; 1 ) ;
  data = Right ( values ; Length ( values ) - start ) ;
  data = Substitute ( data ; ": " ; "¶" ) ;
  data = GetValue ( data ; 2 )
] ;
  data
)
```

GetField

The `GetField` function returns the contents of a field based on the name of the field in a text string rather than a dynamic reference.

Format `GetField (fieldName)`
Parameters `fieldName` = a text expression that contains the name of field.
Result Whatever data is contained in the field provided.

The parameter can be either the name of the field or a text-based reference to the field. Use the latter for a field from a related table.

```
GetField ( "Full Name" )                    // result = Mark Munro
GetField ( "People::Full Name" )         // result = Mark Munro
```

■ **Caution** When using this function, it is advisable to not use hard-coded text for field names since they will not be updated when field names change. Instead use the `GetFieldName` function to convert a dynamic field reference into a text-based field reference and then pass that to the `GetField` function.

If the parameter is a field reference and that field's contents are the name of a field, the function will return the contents of the field named within the field specified. For example, if there is a field named `ReferringField` that contains the value `People::Full Name`, the result of this example will be the same as above:

```
GetField ( People::ReferringField )       // result = Mark Munro
```

However, if a field reference points to a field that contains anything other than the name of a field, the result will be an error.

```
GetField ( People::Full Name )           // result = ?
```

GetFieldName

The `GetFieldName` function returns the name of a field reference.

Format `GetFieldName (field)`
Parameters `field` = a field reference or expression that results in one.
Result A text string containing the fully qualified name of the field.

This example shows the function converting a dynamic field reference into a text-based field reference:

```
GetFieldName ( People::Full Name )       // result = "People::Full Name"
```

■ **Caution** In most cases, when a field reference is provided to a function, the function is using the reference to access and manipulate the field *value*. However, in this case, the function ignores the field *value* and instead focuses on the field *reference* itself, returning the reference as a text string.

GetLayoutObjectAttribute

The `GetLayoutObjectAttribute` function will return an attribute of a named layout object from the current layout.

Format	<p>Without optional parameters:</p> <pre>GetLayoutObjectAttribute (objectName ; attributeName)</pre> <p>Optionally, when retrieving an attribute of a repeating field, the repetition number can be added. If the parameter is excluded or set to a value of 0, the first repetition will be used for applicable attributes:</p> <pre>GetLayoutObjectAttribute (objectName ; attributeName ; repetitionNumber)</pre> <p>Optionally, when retrieving an attribute of a portal row, the number can be added. If the parameter is excluded or set to a value of 0, the first row will be used for applicable attributes:</p> <pre>GetLayoutObjectAttribute (objectName ; attributeName ; portalRowNumber)</pre>
Parameters	<p><code>objectName</code> = the name of the layout object (Chapter 31, "Naming Objects").</p> <p><code>attributeName</code> = the name of the desired attribute (see list below).</p> <p><code>repetitionNumber</code> = (optional) an integer indicating the repetition number for the object for which to retrieve the attribute.</p> <p><code>portalRowNumber</code> = (optional) an integer indicating the portal row from which to retrieve the attribute.</p>
Result	A text string containing the value of the specified attribute.

AttributeName Options

The `attributeName` parameter can contain any of the following attributes:

- `objectType`
- `hasFocus` = returns a Boolean indicating if the object has focus
- `containsFocus` = returns a Boolean indicating if the object contains the object that has focus
- `isFrontPanel` = (tab or slide panel only) returns a Boolean indicating if the object is in front
- `isActive` = returns a Boolean indicating if the object is active or not
- `isObjectHidden` = returns a Boolean indicating if the object is currently hidden or not
- `bounds` = returns a list of four space-separated numeric values that indicate the position of the object on the layout, in the following order: top, left, bottom, right, rotation degrees; based on the current object size and position that may change as a window is resized
- `left` = returns a number representing the left edge of the object on the layout; based on the current object position that may change as a window is resized
- `right` = returns a number representing right-left edge of the object on the layout; based on the current object position that may change as a window is resized
- `top` = returns a number representing the top edge of the object on the layout; based on the current object position that may change as a window is resized

- `bottom` = returns a number representing the bottom edge of the object on the layout; based on the current object position that may change as a window is resized
- `width` = returns the width of the object; based on the current object size that may change as a window is resized
- `height` = returns the height of the object; based on the current object size that may change as a window is resized
- `rotation` = returns a number representing the rotation of the object (degrees)
- `startPoint` = returns two space-separated numeric values that indicate the start position of a line or the upper-left corner of another object type in the following order: top, left; based on the current object position that may change as a window is resized
- `endPoint` = returns two space-separated numeric values that indicate the end position of a line or the lower-right corner of other object types, in the following order: bottom, right; based on the current object position that may change as a window is resized
- `source` = returns a text string containing the source information of the object, varying by type listed in Table 21-1
- `content` = returns a text string containing the content of object, varying by type listed in Table 21-1
- `enclosingObject` = returns the name of the object that encloses the object specified
- `containedObjects` = returns a list of named objects contained by the object specified

Table 21-1. The result of the source and content attribute based on object type

Object Type	Source returns	Content returns
Button & Popover	—	The text of the button
Chart	An XML description of the chart object	A bitmap representation of the chart object
Field	A fully qualified field name with table	The field contents formatted based on the object's settings
Graphic	Information about the image such as filename	Information about the image such as filename
Portals	The name of the related table	N/A
Text Object	The text	The text
Web viewers	The URL currently displayed	The HTML code of the current web page displayed

These examples show the function operating with different input:

```
GetLayoutObjectAttribute ( "image" ; "bounds" )           // result = 532 625 955 953 0
GetLayoutObjectAttribute ( "image" ; "objectType" )     // result = field
```

■ **Caution** The bounds, left, right, top, and bottom measurements returned by the `GetLayoutObjectAttribute` function are based on the layout object's position relative to the top-left corner of the primary display's *desktop area* and not the object's position within a *layout*. That means that as a window is moved around the screen, these values will *change*. This behavior is consistent with past versions of FileMaker Pro for macOS but is a change for Windows.

GetNthRecord

The `GetNthRecord` function returns the contents of a field from a specified record within the found set regardless of the current record.

Format	<code>GetNthRecord (fieldname ; recordNumber)</code>
Parameters	<p><code>fieldName</code> = a reference to a field.</p> <p><code>recordNumber</code> = a number of the desired record.</p>
Result	The contents of the field for the record specified.

This example would return the contents of the `Full Name` field from the fifth record in the found set from the `People` table:

```
GetNthRecord ( People::Full Name ; 5 )
```

If

The `If` function conditionally returns a value depending on the outcome of a text expression. Although the function can be nested to handle more than two results, the `Case` function is recommended.

Format	<p>When one conditional result is required, only two parameters must be included: a test expression and a result to be returned if the test is true:</p> <pre>If (test ; resultOne)</pre> <p>Where the above will return a null or blank result if the <code>test</code> is false, an optional second result parameter can be included, which will be returned instead if the test is false:</p> <pre>If (test ; resultOne ; resultTwo)</pre>
Parameters	<p><code>test</code> = a numeric value or logical expression that evaluates to a Boolean.</p> <p><code>resultOne</code> = a value will be returned if the test expression is true.</p> <p><code>resultTwo</code> = (optional) a value that will be returned if the test expression is false.</p>
Result	The appropriate result based on the value returned from the expression.

This simple example will determine if the date of an invoice is in the past or the future. If the value in the `Invoice` record's `Date` field is less than or equal to today's date, the test expression will return true and the function will return the word `Past`. If the date is greater than today's date, the test expression will return false and the function will return the word `Future`.

```
If ( Invoice::Date ≤ Get ( CurrentDate ) ; "Past" ; "Future" )
```

IsEmpty

The IsEmpty function determines if a value is empty.

Format	IsEmpty (field)
Parameters	field = a field reference or expression.
Result	A Boolean: 1 = true; the parameter evaluates to an empty string 0 = false; the parameter evaluates to a text value; anything other than an empty string.

These examples show the functions operating with different input:

```
IsEmpty ( "" )           // result = 1
IsEmpty ( "Hello World" ) // result = 0
```

IsValid

The IsValid function determines if another field's value is valid.

Format	IsValid (field)
Parameters	field = a field reference or expression.
Result	A Boolean indicating if the field provided is valid or not. A false result may indicate either: —A field's contents are a different data type than the field is defined to contain; a calculation returning a text string when it is defined to return a number. —A related field or entire table could not be found or has been deleted.

If the Invoice record's Date Due calculation field is defined to contain a date and instead evaluates to a number, it will return 0 (zero) indicating an invalid result:

```
IsValid ( Invoice::Date Due ) // result = 0
```

IsValidExpression

The IsValidExpression function determines if an expression's syntax is correct and will execute successfully. This can be helpful to confirm the validity of some dynamically compiled code that will be sent to the Evaluate function.

Format	IsValidExpression (expression)
Parameters	expression = an expression.
Result	A Boolean: 1 = true; the expression is valid 0 = false; the expression is not valid.

This example will return a false result because the `Get (CurrentDate)` function in the text string is missing a closing parenthesis and will not evaluate to a valid result:

```
IsValidExpression ( "Get ( CurrentDate " ) // result = 0
```

Once the error is corrected, the result will be true:

```
IsValidExpression ( "Get ( CurrentDate )" ) // result = 1
```

Let

The `Let` function forms a statement in which one or more temporary variables can be named and initialized and then used during the execution of a nested calculation formula. The function allows for the compression of lengthy field names and expressions into shorter variables, which can then be used multiple times within a complex formula. It also allows the un-nesting of complex formulas: separating shorter expressions and putting them into variables one after the other and then using them together in a compressed formula.

Format When setting a single variable, two parameters are required: first, a combination parameter of a variable, equal sign operator and expression; and second, a calculation formula:

```
Let ( var1 = expression ; calculation )
```

When multiple variables are required, they are added within a set of square brackets and a semicolon delimiter (shown here in a preferred, optional multi-line format to improve clarity):

```
Let ( [
    var1 = expression1 ;
    var2 = expression2 ;
    var3 = expression3
] ;
    calculation
)
```

Parameters `var` = the name of a calculation local variable, script local variable, or database local variable.
`expression` = a field reference, constant, or expression, the result of which will be placed into the corresponding variable.
`calculation` = a formula that has access to all the variables defined above.

Result The result of the calculation.

This is a simple example of a single variable being initialized as a text string and then returned as the result:

```
Let ( textString = "World" ; "Hello " & textString )) // result = "Hello World"
```

The following example has only one variable, but the calculation is more realistic. First, the value in the `Number of Days Overdue` field is placed into a `daysOver` variable. The calculation portion of the statement uses the `Case` function to create a text string with content that varies if `daysOver` is greater than 30 or not. Notice how the field is used twice in the calculation by means of the shorter `daysOver` variable name.

```
Let (
  daysOver = Invoice::Number of Days Overdue
;
  Case (daysOver > 30 ; "Past Due (" & daysOver & " days)" ; "Current" )
)
```

The following example demonstrates how a Let statement can be used to set global variables. This example performs the same function and returns the same result as the previous one. The only difference is that it puts the value from the Customer ID field into a global variable named \$\$LastCustomerID and it increments the value in another global variable \$\$CustomerCount by 1.

```
Let ( [
  $$LastCustomerID = Invoice::Customer ID ;
  $$CustomerCount = $$CustomerCount + 1 ;
  daysOver = Invoice::Number of Days Overdue
] ;
  Case (daysOver > 30 ; "Past Due (" & daysOver & " days)" ; "Current" )
)
```

Expressions with the Let statement can get complex. Remember that this function can be nested within other functions and other functions can be nested within its variable expressions and calculations.

Lookup

The Lookup function attempts to return a value from a related field through a specific relationship.

Format	Without an optional parameter: Lookup (sourceField) Optionally, a second parameter can be included to define a result when the function fails to find a related match: Lookup (sourceField ; failExpression)
Parameters	sourceField = a field reference to a related field from which a value should be returned. failExpression = (optional) an expression indicating the result when no related value is found.
Result	The contents of the specified field or the optional failExpression.

These examples assume two tables — Products (shown in Table 21-2) and Invoice Line Items — and a relationship named Invoice Line | Product that connects them by the Product ID fields in each.

Table 21-2. An example Products table

Product ID	Product Name
1	Widget A
2	Widget B
3	Widget C

Table 21-3. An example of an empty Invoice Line Items table.

Product ID	Invoice Line Product Name
------------	---------------------------

If the Invoice Line Product Name was a calculation that will look up the product name when the user types in a Product ID, the formula would look like this with the results shown depending on the values entered:

```
Lookup ( Invoice Line | Product: Product Name )
// entering "1" result = "Widget A"
// entering "3" result = "Widget C"
// entering "4" result = ""
```

By providing an optional failExpression, the result of a missing Product ID would no longer be an empty string:

```
Lookup ( Invoice Line | Product::Product Name ; "Error: Product ID Not Found" )
// entering "1" result = "Widget A"
// entering "3" result = "Widget C"
// entering "4" result = "Error: Product ID Not Found"
```

■ **Caution** This function is a somewhat outdated method for accessing related information. When defining fields, the *auto-enter calculation* feature is preferable to lookups and any formula can reach through a relationship and access-related values with ease.

LookupNext

Like Lookup, the LookupNext function attempts to return a value from a related field through a specific relationship. However, it has a fail option that will automatically retrieve the next lower or higher value when no exact match is found.

Format	LookupNext (sourceField ; lower/higherFlag)
Parameters	sourceField = a field reference to a related field from which a value should be returned. lower/higherFlag = the keyword lower or higher indicating which value should be used if no exact match is found.
Result	The contents of the specified field from a record with an exact match through the relationship or the next higher or lower value depending on the keyword provided.

Using the same example from the Lookup function, the failure to find a match for a Product ID of 4 will now retrieve the next lower value: Widget C, which has a Product ID of 3.

```
Lookup ( Invoice Line | Product::Product Name ; lower )
// entering "4" result = "Widget C"
```

Self

The `Self`-function returns the content of an object without naming or referring to it explicitly. This allows a single calculation to be applied to many objects without modification since the reference to the object is the function. The function can be used in layout calculations for conditional formatting, hiding layout objects, and tooltips. It is also useful in the field definition as an auto-enter calculation for other field types.

Format	Self
Parameters	Not applicable
Result	The contents of the object that are associated with the calculation using the function.

A field's *auto-enter formula*, can clean up user-entered data with a formula like the example below, which removes all text formatting, non-alphabetic characters, and extra spaces. If the user entered " M a r k - M u n r o 1/5/2017 @ %\$ ", their entry would instantly be cleaned up to "Mark Munro" helping to ensure neat and consistent data entry.

```
Let ( [
    entry = TextFormatRemove ( Self ) ;
    entry = Filter ( entry ; "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz " ) ;
    entry = TrimAll ( entry ; 0 ; 0 )
] ;
    entry
)
```

Summary

In this chapter, we explored all the built-in functions for performing logical calculations.

CHAPTER 22



Get Functions

Get functions provide a single piece of information about the computer's system, the user's environment, the current database context, and the status of various processes or current settings. These are useful when creating conditional formulas or scripts that return different results or perform a different task based on some aspect of the current context or situation. They each require a *single unchanging* parameter — a keyword that indicates the desired information.

Get (<StaticParameter>)

Get Function Reference

FileMaker provides the following built-in get functions:

- Get (AccountExtendedPrivileges)
- Get (AccountName)
- Get (AccountPrivilegeSetName)
- Get (ActiveFieldContents)
- Get (ActiveFieldName)
- Get (ActiveFieldTableName)
- Get (ActiveLayoutObjectName)
- Get (ActiveModifierKeys)
- Get (ActivePortalRowNumber)
- Get (ActiveRepetitionNumber)
- Get (ActiveSelectionSize)
- Get (ActiveSelectionStart)
- Get (AllowAbortState)
- Get (AllowFormattingBarState)
- Get (ApplicationArchitecture)
- Get (ApplicationLanguage)
- Get (ApplicationVersion)
- Get (CalculationRepetitionNumber)

- Get (ConnectionAttributes)
- Get (ConnectionState)
- Get (CurrentDate)
- Get (CurrentExtendedPrivileges)
- Get (CurrentHostTimestamp)
- Get (CurrentPrivilegeSetName)
- Get (CurrentTime)
- Get (CurrentTimestamp)
- Get (CurrentTimeUTCMilliseconds)
- Get (CustomMenuSetName)
- Get (DesktopPath)
- Get (Device)
- Get (DocumentsPath)
- Get (DocumentsPathListing)
- Get (EncryptionState)
- Get (ErrorCaptureState)
- Get (FileMakerPath)
- Get (FileName)
- Get (FilePath)
- Get (FileSize)
- Get (FoundCount)
- Get (HighContrastColor)
- Get (HighContrastState)
- Get (HostApplicationVersion)
- Get (HostIPAddress)
- Get (HostName)
- Get (InstalledFMPlugins)
- Get (LastError)
- Get (LastExternalErrorDetail)
- Get (LastMessageChoice)
- Get (LayoutAccess)
- Get (LayoutCount)
- Get (LayoutNumber)
- Get (LayoutTableName)

- Get (ViewState)
- Get (MenubarState)
- Get (ModifiedFields)
- Get (MultiUserState)
- Get (NetworkProtocol)
- Get (NetworkType)
- Get (PageNumber)
- Get (PersistentID)
- Get (PreferencesPath)
- Get (PrinterName)
- Get (QuickFindText)
- Get (PrivilegeSetName)
- Get (RecordAccess)
- Get (RecordID)
- Get (RecordModificationCount)
- Get (RecordNumber)
- Get (RecordOpenCount)
- Get (RecordOpenState)
- Get (RequestCount)
- Get (RequestOmitState)
- Get (ScreenDepth)
- Get (ScreenHeight)
- Get (ScreenScaleFactor)
- Get (ScreenWidth)
- Get (ScriptAnimationState)
- Get (ScriptName)
- Get (ScriptParameter)
- Get (ScriptResult)
- Get (SortState)
- Get (StatusAreaState)
- Get (SystemDrive)
- Get (SystemIPAddress)
- Get (SystemLanguage)
- Get (SystemNICAddress)

- `Get (SystemPlatform)`
- `Get (SystemVersion)`
- `Get (TemporaryPath)`
- `Get (TextRulerVisible)`
- `Get (TotalRecordCount)`
- `Get (TouchKeyboardState)`
- `Get (TriggerCurrentPanel)`
- `Get (TriggerExternalEvent)`
- `Get (TriggerGestureInfo)`
- `Get (TriggerKeystroke)`
- `Get (TriggerModifierKeys)`
- `Get (TriggerTargetPanel)`
- `Get (UserCount)`
- `Get (UserName)`
- `Get (UseSystemFormatsState)`
- `Get (UUID)`
- `Get (WindowContentHeight)`
- `Get (WindowContentWidth)`
- `Get (WindowDesktopHeight)`
- `Get (WindowDesktopWidth)`
- `Get (WindowHeight)`
- `Get (WindowLeft)`
- `Get (WindowMode)`
- `Get (WindowName)`
- `Get (WindowOrientation)`
- `Get (WindowStyle)`
- `Get (WindowTop)`
- `Get (WindowVisible)`
- `Get (WindowWidth)`
- `Get (WindowZoomLevel)`

Get (AccountExtendedPrivileges)

This function returns extended privileges assigned to the account used to authenticate the connection to the database file.

Format `Get (AccountExtendedPrivileges)`
Result A text string containing the name of extended privilege for the current user.

This example shows a typical result from this function:

```
Get ( AccountExtendedPrivileges )
= "fmapp"
```

Get (AccountGroupName)

This function returns the account group name when the account is using an external server or OAuth authentication method.

Format `Get (AccountGroupName)`
Result A text string containing the name of the account group for the current user.

This example shows a typical result from this function:

```
Get ( AccountGroupName )        // result = "dbmarketing"
```

Get (AccountName)

This function returns the account name used to authenticate the connection to the database file.

Format `Get (AccountName)`
Result A text string containing the name of the current user account.

This example shows a typical result from this function:

```
Get ( AccountName )    // result = "Joanne Miller"
```

Get (AccountPrivilegeSetName)

This function returns the name of the privilege set for the account used to authenticate the connection to the database file.

Format `Get (AccountPrivilegeSetName)`
Result A text string containing the privilege set for the current user's account.

This example shows a typical result from this function:

```
Get ( AccountPrivilegeSetName )
= "[Full Access]"
```

Get (ActiveFieldContents)

This function returns the contents of the field that has current focus in the front window.

Format Get (ActiveFieldContents)

Result The contents of the field or an empty string if no field has focus.

This example assumes the user is viewing a record in a table named `Contacts` for a person named “John Smith” and has the cursor in a field named `Full Name`:

```
Get ( ActiveFieldContents )    // result = Mark Munro
```

Get (ActiveFieldName)

This function returns the field that has current focus in the front window.

Format Get (ActiveFieldName)

Result A text string with the name of the field that has current focus (empty string if no field is in focus).

This example assumes the user is viewing a record in a table named `Contacts` for a person named “John Smith” and has the cursor in a field named `Full Name`:

```
Get ( ActiveFieldName )    // result = Full Name
```

Get (ActiveFieldTableName)

This function returns the table to which the field with focus belongs.

Format Get (ActiveFieldTableName)

Result A text string containing the name of the table for the current field. If the current field is from a related table rather than the current table, the result will be the name of that related table. Don’t assume the active field table is the same as the active layout table. If no field has focus, it will return an empty string.

This example assumes the user is viewing a record in a table named `Contacts` for a person named “John Smith” and has the cursor in a field named `Full Name`:

```
Get ( ActiveFieldTableName )    // result = "Contacts"
```

Get (ActiveLayoutObjectName)

This function returns the name current active layout object: fields, buttons, portals, tabs, web viewers (Chapter 31, “Naming Objects”).

Format `Get (ActiveLayoutObjectName)`

Result A text string containing the object name of the currently active layout object or an empty string if no object is active or the active object does not have a name assigned.

This example shows the result from the function if active focus is on a portal with an assigned name of "Invoice Portal":

```
Get ( ActiveLayoutObjectName ) // result = "Invoice Portal"
```

Get (ActiveModifierKeys)

This function returns a value indicating which modifier key(s) the user is currently pressing.

Format `Get (ActiveModifierKeys)`

Result A number that is calculated by adding together the following, depending on which key(s) is being held down by the user:

```
Shift key = +1
Caps Lock key = +2
Ctrl key (Windows) or Control key (macOS) = +4
Alt key (Windows) or Option key (macOS) = +8
Command key (macOS) = +16
```

This example indicates that the user is holding down the Control and Command keys on macOS.

```
Get ( ActiveModifierKeys ) // result = 20
```

This example uses several other functions to build a carriage return-delimited list of the names of every modifier key held down at the time it is evaluated.

```
Let ( [
    keys = Get ( ActiveModifierKeys ) ;
    list = Case ( Mod ( keys; 32 ) ≥ 16 ; "Command¶" ; "" ) ;
    list = list & Case ( Mod ( keys; 16 ) ≥ 8 ; "Option¶" ; "" ) ;
    list = list & Case ( Mod ( keys; 8 ) ≥ 4 ; "Control¶" ; "" ) ;
    list = list & Case ( Mod ( keys; 4 ) ≥ 2 ; "Caps Lock¶" ; "" ) ;
    list = list & Case ( Mod ( keys; 2 ) ≥ 1 ; "Shift¶" ; "" )
] ;
list
)
```

Get (ActivePortalRowNumber)

This function returns which portal row is currently selected.

Format Get (ActivePortalRowNumber)

Result A number indicating the portal row that has focus or contains the field with focus. If no portal row is selected it will return 0.

This example shows a typical result from this function:

```
Get ( ActivePortalRowNumber ) // result = 5
```

Get (ActiveRepetitionNumber)

This function returns the active field repetition of the field, which has focus in the front window.

Format Get (ActiveRepetitionNumber)

Result A number indicating which repetition has current focus. It will return an empty string if no field is selected.

This example shows a typical result from this function:

```
Get ( ActiveRepetitionNumber ) // result = 3
```

Get (ActiveSelectionSize)

This function returns the length in characters of selected text within the active field.

Format Get (ActiveSelectionSize)

Result A number indicating the number of characters that makes up the selection.

This example shows a typical result from this function:

```
Get ( ActiveSelectionSize ) // result = 13
```

Get (ActiveSelectionStart)

This function determines the character position at which a selection of text within a field begins in the front window.

Format Get (ActiveSelectionStart)

Result A number indicating where a selection of text begins within a field. If no selection exists or no field has current focus, it will return an empty string.

This example shows the function returning the number of characters that make up the selection:

```
Get ( ActiveSelectionSize ) // result = 5
```

This example uses several functions to extract the text of just the current selection from a field:

```
Let ( [
    count = Get ( ActiveSelectionSize ) ;
    start = Get ( ActiveSelectionStart ) ;
    text = Get ( ActiveFieldContents )
] ;
    Middle ( text ; start ; count )
)
```

Get (AllowAbortState)

This function determines if the user has the option to abort a script or not.

Format Get (AllowAbortState)

Result A Boolean indicating if the user can abort the current script or not:
 1 = true; the user can abort the current script
 0 = false; the user can't abort the current script

This example shows a typical result from this function:

```
Get ( AllowAbortState ) // result = 1
```

Get (AllowFormattingBarState)

This function determines if the formatting bar is able to be made visible or not.

Format Get (AllowFormattingBarState)

Result A Boolean indicating if the formatting bar can be made visible:
 1 = true
 0 = false

This example shows a typical result from this function:

```
Get ( AllowFormattingBarState ) // result = 1
```

Get (ApplicationArchitecture)

This function return the architecture of the version of the application used to open the database file, which can be useful if installing a plug-in that has processor-specific versions.

Format `Get (ApplicationArchitecture)`

Result A text string containing the architecture of the FileMaker Pro application. The possible values are:

- `i386` — indicates a 32-bit version of FileMaker Pro.
- `x86_64` — indicates a 64-bit version of FileMaker Pro, FileMaker Server, FileMaker WebDirect, and Custom Web Publishing.
- `arm7` — indicates FileMaker Go running on an ARMv7-based device.
- `arm7s` — indicates FileMaker Go running on an ARMv7s-based device.
- `arm64` — FileMaker Go running on a 64-bit ARM-based device.

This example shows a typical result from this function:

```
Get ( ApplicationArchitecture )           // result = "x86_64"
```

Get (ApplicationLanguage)

This function returns the current language version of the application used to open the database file.

Format `Get (ApplicationLanguage)`

Result An English text string containing the name of the language version of the FileMaker Pro application or of the web browser for FileMaker WebDirect used to open the database file.

This example shows a typical result from this function:

```
Get ( ApplicationLanguage )             // result = "English"
```

Get (ApplicationVersion)

This function returns the product name and version for the current version of FileMaker Server used to open the database file.

Format `Get (ApplicationVersion)`

Result A text string containing an application identification string and full version number for the FileMaker Pro product running on the host computer or an empty string if the database is running locally instead of accessed across a network from a host.

This example shows all the possible results:

```
Get ( ApplicationVersion )
// = "Pro 16.0.1"
// = "ProAdvanced 16.0.1"
// = "Runtime 16.0.1"
// = "Web Publishing Engine 16.0.1"
// = "xDBC 16.0.1"
// = "Server 16.0.1"
// = "Go 16.0.1"
// = "Go_iPad 16.0.1"
```

Get (CalculationRepetitionNumber)

This function returns the repetition of a repeating calculation field being evaluated. This is useful when each repetition of a calculation field must pull a value from a corresponding repeating field or perform slightly different functions depending on the repetition.

Format Get (CalculationRepetitionNumber)

Result The number of the repetition of a calculation field that is currently being evaluated.

This example assumes a Start Date field with a value of 1/5/2017. A second calculation field with five repetitions will display the dates of weekly reviews. Using this formula, each repetition will calculate a different number of days to add based on an identification of its own position using this function.

```
Extend ( Example::Start Date ) + ( 7 * Get ( CalculationRepetitionNumber ) )
// repetition 1 = 1/12/2017
// repetition 2 = 1/19/2017
// repetition 3 = 1/26/2017
// repetition 4 = 2/2/2017
// repetition 5 = 2/9/2017
```

Get (ConnectionAttributes)

This function returns details about the connection to a host computer, assuring the connection is secure.

Format Get (ConnectionAttributes)

Result The host server's name and SSL certificate issuer.

This example shows a typical result from this function:

```
Get ( ConnectionAttributes )
// result =
   [ Peer Certificate ]
   commonName: office_server
   CA issuers: ABC SSL Inc.
```

Get (ConnectionState)

This function returns the security state of the connection to a database accessed across a network from a host computer.

Format Get (ConnectionState)

Result A number indicating the connection state. The possible values are:

- 0 = indicates no network connection (file is used locally)
 - 1 = indicates a non-encrypted connection that is not encrypted (no SSL certificate)
 - 2 = indicates an unverifiable encrypted connection (SSL certificate that can't be verified)
 - 3 = indicates a verifiable encrypted connection (SSL certificate that is verified)
-

This example shows a typical result from this function:

```
Get ( ConnectionState )           // result = 1
```

Get (CurrentDate)

This function returns the current date.

Format Get (CurrentDate)

Result A date text string containing the current date, formatted based on the operating system preferences of the computer evaluating the function; generally, the user's computer unless the Perform Script On Server command was used in a script.

This example shows a typical result from this function:

```
Get ( CurrentDate )           // result = 1/15/2017
```

Get (CurrentExtendedPrivileges)

This function returns the name of the extended privileges enabled based on the user's method of logging into the database file.

Format Get (CurrentExtendedPrivileges)

Result The name of the active extended privileges.

This example shows a typical result from this function:

```
Get ( CurrentExtendedPrivileges )           // result = "fmapp"
```

Get (CurrentHostTimestamp)

This function returns the current date and time of the host computer, which is useful if the client is located in a different time zone than the server. Using this function in an unstored calculation that is visible on a list layout may severely degrade the performance of your database since it will attempt to update the value for every record in the found set each time the layout view is updated.

Format Get (CurrentHostTimestamp)

Result A timestamp text string containing the current date and time, formatted based on the operating system preferences of the host computer.

This example shows a typical result from this function:

```
Get ( CurrentHostTimestamp )           // result = "1/15/2017 1:53:09 PM"
```

Get (CurrentPrivilegeSetName)

This function returns the name of the privilege set based on the user's method of logging into the database file.

Format Get (CurrentPrivilegeSetName)

Result The name of the active privilege set.

This example shows a typical result from this function:

```
Get ( CurrentPrivilegeSetName )            // result = "[Full Access]"
```

Get (CurrentTime)

This function returns the current time.

Format Get (CurrentTime)

Result A time text string containing the current time, formatted based on the operating system preferences of the computer evaluating the function; generally the user's computer unless the Perform Script On Server command was used in a script.

This example shows a typical result from this function:

```
Get ( CurrentTime )            // result = "2:05:10 PM"
```

Get (CurrentTimestamp)

This function returns the current date and time.

Format Get (CurrentTimestamp)

Result A timestamp text string containing the current date and time, formatted based on the operating system preferences of the computer evaluating the function; generally, the user's computer unless the Perform Script On Server command was used in a script.

This example shows a typical result from this function:

```
Get ( CurrentTimestamp )            // result = "1/15/2017 2:05:10 PM"
```

Get (CurrentTimeUTCMilliseconds)

This function returns the current time in Coordinated Universal Time (UTC) to the nearest millisecond without regard to the current time zone. UTC was previously known as Greenwich Mean Time (GMT).

Format	<code>Get (CurrentTimeUTCmilliseconds)</code>
Result	A number representing the current time of the computer running the script in milliseconds without regard to the user's current time zone.
More Info	For more information on UTC, visit: https://en.wikipedia.org/wiki/Coordinated_Universal_Time .

This example shows a typical result from this function:

```
Get ( CurrentTimeUTCmilliseconds ) // result = 63603934624024
```

To calculate the time for a specific time zone, just factor in the UTC time zone adjustment for the region. For example, New York City is five hours behind UTC during Eastern Standard Time, so this formula will return the time adjusted for that region:

```
Let ( [
    timeUTC = Get ( CurrentTimeUTCmilliseconds ) ;
    timeNY = Round ( ( timeUTC + ( -5 * 3600000 ) ) / 1000 ; 0 )
] ;
    GetAsTimestamp ( timeNY )
)
```

Get (CustomMenuSetName)

This function returns the name of the active custom menu set (Chapter 33).

Format	<code>Get (CustomMenuSetName)</code>
Result	A text string containing the name of the active menu set. If the Standard FileMaker Menus are currently active or if the function evaluates when running FileMaker WebDirect, it will return an empty string.

This example shows a typical result from this function:

```
Get ( CustomMenuSetName ) // result = "Project Custom Menu Set"
```

Get (DesktopPath)

This function returns the path to the user's local desktop folder.

Format	<code>Get (DesktopPath)</code>
Result	A text string containing the path.

This example shows a typical result from this function:

```
Get ( DesktopPath )
// macOS result = /Macintosh HD/Users/admin/Desktop/
// Windows result = /C:/Users/Admin/Desktop/
```

Get (Device)

This function returns the type of computer or device upon which the user is running a FileMaker Product.

Format Get (Device)

Result A number indicating the device type. The possible values are:

- 0 = an unknown device
 - 1 = a macOS computer
 - 2 = a Windows computer
 - 3 = an iPad
 - 4 = an iPhone or iPod touch
 - 5 = an Android device
-

This example shows a typical result from this function:

```
Get ( Device ) // result = 1
```

Since the function is zero based, the best method to convert the result into the device name is to use the Choose function.

```
Choose ( Get ( Device ) ;
    "Unknown" ;
    "macOS" ;
    "Windows" ;
    "iPad" ;
    "iPhone" ;
    "Android" ;
    "" )
```

It is also possible to do this with the GetValue function; however, you will need to add 1 to the result to adjust it from zero-based to one-based, as shown in this example:

```
GetValue ( "Unknown¶macOS¶Windows¶iPad¶iPhone¶Android" ; Get ( Device ) + 1 )
// result = macOS
```

Get (DocumentsPath)

This function returns the path to the documents folder. When the function is evaluated from a client version of the software, it returns their local Documents folder. When evaluated by a FileMaker Server, it returns the /FileMaker Server/Data/Documents/ folder.

Format Get (DocumentsPath)

Result A text string containing the path.

This example shows a typical result from this function:

```
Get ( DocumentsPath )
// Mac result = /Macintosh HD/Users/admin/Documents/
// Windows result = /C:/Users/admin/Documents/
```

Get (DocumentsPathListing)

This function returns a list of every item within the user's local documents folder, including all nested items.

Format Get (DocumentsPathListing)

Result A carriage return-delimited list of paths.

This example shows a typical result from this function:

```
Get ( DocumentsPathListing )
// result =
    /Macintosh HD/Users/admin/Documents/Expense Reports/
    /Macintosh HD/Users/admin/Documents/Expense Reports/Q1.xlsx
    /Macintosh HD/Users/admin/Documents/To Do List.txt
```

Get (EncryptionState)

This function returns a database file's current encryption state.

Format Get (EncryptionState)

Result A Boolean indicating the encryption state:

0 = the file is not encrypted

1 = the file is encrypted

If the file is encrypted, the Shared ID will be included after the 1, delimited with a carriage return.

This example shows the result when the file *is not* encrypted:

```
Get ( EncryptionState )
// result =
    0
```

This example shows a result when the file *is* encrypted:

```
Get ( EncryptionState )
// result =
    1
    51836
```


Get (ErrorCaptureState)

This function returns the state of error capture in scripts.

Format Get (ErrorCaptureState)

Result A Boolean indicating if the Set Error Capture script step has been turned on or off (Chapter 35, “Managing Errors”):

0 = error capture is off

1 = error capture is on

This example shows a typical result from this function:

```
Get ( ErrorCaptureState )    // result = 1
```

Get (FileMakerPath)

This function returns the path to the local FileMaker Pro application running the database file.

Format Get (FileMakerPath)

Result A text string containing the path to the application.

This example shows a typical result from this function:

```
Get ( FileMakerPath )
// macOS result = /Macintosh HD/Applications/FileMaker Pro 16/
// Windows result = /C:/Program Files/FileMaker/FileMaker Pro 16/
```

Get (FileName)

This function returns the name of the current database file.

Format Get (FileName)

Result A text string containing the name of the file without extension.

This example shows the result when the function is performed within a database file named Contacts.fmp12:

```
Get ( FileName )    // result = "Contacts"
```

Get (FilePath)

This function returns the path to the current database file.

Format Get (FilePath)

Result A text string containing the path to the file.

If the database is accessed from a folder directory, the result will be a path to the file:

```
Get ( FilePath )
// macOS result = /Macintosh HD/Users/admin/Documents/Contacts.fmp12
// Windows result = /C:/Users/admin/Documents/Contacts.fmp12
```

If the database is accessed from across the network to a host computer and the function evaluates on the local client computer, the result will be in the form of a `fmurl`:

```
// Hosted database result = fmnet:/10.0.1.51/Contacts.fmp12
```

Get (FileSize)

This function returns the file size of the current database.

Format `Get (FileSize)`

Result A number indicating the file size in bytes.

This example shows a typical result from this function:

```
Get ( FileSize )
= 352256
```

To convert the value to megabytes, divide the result and round it:

```
Round ( Get ( FileSize ) / 1024^2 ; 2 )
= .34
```

Get (FoundCount)

This function returns the number of records in the found set of the current window.

Format `Get (FoundCount)`

Result A number indicating the count of records in the found set.

This example shows a typical result from this function:

```
Get ( FoundCount )    // result = 47
```

Get (HighContrastColor)

This function returns the current high-contrast, default color scheme name if an option is enabled and selected in Windows Ease of Access Center.

Format `Get (HighContrastColor)`

Result A text string containing the name of the high-contrast color option selected or an empty string if this option is not enabled, or if used on a macOS computer.

This example shows a typical result from this function:

```
Get ( HighContrastColor )    // result = High Contrast White
```

Get (HighContrastState)

This function determines if High Contrast is available and active in Windows Ease of Access Center.

Format Get (HighContrastState)

Result A Boolean indicating if the feature is available and active:
 0 = indicates either the contrast option is not available, inactive or the current computer is a macOS computer
 1 = indicates the contrast option is available and inactive

This example shows a typical result from this function:

```
Get ( HighContrastState )    // result = 1
```

Get (HostApplicationVersion)

This function returns the product name and version for the current database file's host from which the file is being accessed across a network. The host might be a copy of FileMaker Pro, FileMaker Pro Advanced, or FileMaker Server. If the file is running locally (unhosted), the function will return an empty value.

Format Get (HostApplicationVersion)

Result A text string containing an application specifier and full version number of the FileMaker Pro product running on the host computer or an empty string if the database is running locally instead of accessed across a network from a host.

This example shows the three possible results:

```
Get ( HostApplicationVersion )
// = Pro 16.0.1
// = ProAdvanced 16.0.1
// = Server 16.0.1
```

Get (HostIPAddress)

This function returns the IP Address for the current database file's host FileMaker Server. The result will be an empty string if the current database is not hosted.

Format Get (HostIPAddress)

Result A text string containing the address of the host computer or an empty string if the database is running locally instead of accessed across a network from a host.

This example shows a typical result from this function:

```
Get ( HostIPAddress ) // result = 10.0.1.50
```

Get (HostName)

This function returns the name of the current database file's host FileMaker Server as displayed in the macOS System Preferences or Windows Control Panel.

Format Get (HostName)

Result A text string containing the name of the host computer.

This example shows a typical result from this function:

```
Get ( HostName ) // result = Production-FileMaker-Server.local
```

Get (InstalledFMPlugins)

This function returns a list of installed plug-ins.

Format Get (InstalledFMPlugins)

Result A carriage return-delimited list of every installed plug-in, listing the name, version, and status of each with a semicolon delimiter. If the plug-in developer has included the version information into the Windows resource file it will be included. The status options are:

Enabled = The plug-in is enabled and can be loaded

Disabled = The plug-in is not enabled and can be loaded

Ignored = The plug-in is unable to load possibly due to an incompatibility

This example shows a typical result from this function:

```
Get ( InstalledFMPlugins )
// result =
    ExamplePlugin;2.4;Enabled
    HypotheticalPlugin;1.3;Enabled
    UselessPlugin;1.0;Disabled
    VersionlessPlugin; ;Enabled
```

Get (LastError)

This function returns the error number of the most recently executed script step.

Format Get (LastError)

Result A number indicating the error that occurred.

This example shows a typical result from this function:

```
Get ( LastError )
// Field is missing result = 102
// Record access is denied result = 200
// Unable to create file on disk result = 800
```

■ **Note** For a complete list of errors visit: http://www.filemaker.com/help/14/fmp/en/html/error_codes.html

Get (LastExternalErrorDetail)

This function returns details about errors that occur externally to FileMaker.

Format Get (LastExternalErrorDetail)

Result The text of any errors that occurred during ODBC imports, Execute SQL script steps, or actions that attempt to connect to a host securely.

Get (LastMessageChoice)

This function returns a number indicating the button clicked in a dialog created with the Show Custom Dialog script step (Chapter 35). This allows a script to perform a different action depending on which button the user clicks.

Format Get (LastMessageChoice)

Result The button position, from 1 to 3, starting from the right of the message dialog.

This example shows a typical result from this function:

```
Get ( LastMessageChoice ) // result = 1
```

Get (LayoutAccess)

This function returns the current level of access the user has to records on the current layout.

Format Get (LayoutAccess)

Result A number indicating the level to records on the current layout access granted to the user:

- 0 = no access to records
- 1 = view only access to records
- 2 = modifiable access to records

This example shows a typical result from this function:

```
Get ( LayoutAccess )    // result = 2
```

Get (LayoutCount)

This function returns a count of all the layouts in the current database file.

Format Get (LayoutCount)

Result A number indicating the total number of layouts in the file.

This example shows a typical result from this function:

```
Get ( LayoutCount )    // result = 7
```

Get (LayoutName)

This function returns the name of the layout displayed in the current window.

Format Get (LayoutName)

Result A text string containing the name of the current layout.

This example shows a typical result from this function:

```
Get ( LayoutName )    // result = "People - List"
```

Get (LayoutNumber)

This function returns the number of the layout displayed in the current window.

Format Get (LayoutNumber)

Result A number indicating the position of the current layout in the list of all available layouts.

This example shows a typical result from this function:

```
Get ( LayoutNumber )    // result = 3
```

Get (LayoutTableName)

This function returns the name of the table for the layout of the current window.

Format Get (LayoutTableName)

Result A text string containing the layout's table name.

This example shows a typical result from this function:

```
Get ( LayoutTableName ) // result = "People"
```

Get (LayoutViewState)

This function returns the current window's layout view state.

Format	Get (LayoutViewState)
Result	A number indicating the current view. The possible values are: 0 = Form View 1 = List View 2 = Table View

This example shows a typical result from this function:

```
Get ( LayoutViewState ) // result = 1
```

Get (MenubarState)

This function determines the current status of the menu bar in FileMaker Go or FileMaker WebDirect.

Format	Get (MenubarState)
Result	A number indicating the menu bar status. The possible values are: 0 = hidden and unlocked 1 = visible and unlocked (FileMaker Pro's only option) 2 = visible and locked 3 = hidden and locked

This example shows a typical result from this function:

```
Get ( MenubarState ) // result = 1
```

Get (ModifiedFields)

This function returns a list of fields for the current table that have been modified since the last time the current record was committed: the equivalent of unsaved changes.

Format	Get (ModifiedFields)
Result	A carriage return-delimited list of field names whose value has changed for the current record since the last record commit. If no fields have been modified, an empty string is returned.

This example shows a typical result from this function:

```
Get ( ModifiedFields )
// result =
    First Name
    Last Name
    Status
```

Get (MultiUserState)

This function shows the sharing status of the current file on the network.

Format Get (MultiUserState)

Result A number indicating the current sharing status of the file. The possible values are:

- 0 = network sharing is off
- 1 = sharing is on and the current user is accessing the file from the host computer and some or all users have access to the file on the network
- 2 = sharing is on and the current user is accessing the file from a client computer and some or all users have access to the file on the network.

This example shows a typical result from this function:

```
Get ( MultiUserState ) // result = 1
```

Get (NetworkProtocol)

This function returns the network protocol that FileMaker is using on the local computer.

Format Get (NetworkProtocol)

Result A text string containing the name of the protocol.

This example shows a typical result from this function:

```
Get ( NetworkProtocol ) // result = TCP/IP
```

Get (NetworkType)

This function returns the current network connection when using FileMaker Pro on an iOS device. This allows a database to vary behavior depending on the connection type and potentially save a user bandwidth charges for excessive data use.

Format `Get (NetworkType)`

Result A number indicating the network type. The possible values are:
 "" = (empty string) the current file is not running in FileMaker Go
 0 = the current file is local on the iOS device so network type is not applicable
 1 = unknown network type
 2 = cellular network
 3 = Wi-Fi network

This example shows a typical result from this function:

```
Get ( NetworkType )    // result = 3
```

Get (PageNumber)

This function returns the number of the current page when printing or previewing.

Format `Get (PageNumber)`

Result A number representing the current page. Returns a 0 if no page is being previewed or printed.

This example shows a typical result from this function:

```
Get ( PageNumber )    // result = 3
```

Get (PersistentID)

This function returns a unique identifier for the device used to access the database file.

Format `Get (PersistentID)`

Result A text string containing a 32-digit hexadecimal string that uniquely identifies the computer or device the user is operating.

This example shows a typical result from this function:

```
Get ( PersistentID )    // result = J55559W953DC11520389A441F0342754
```

Get (PreferencesPath)

This function returns the path to the user's local preferences folder.

Format `Get (PreferencesPath)`

Result A text string containing a folder path.

This example shows a typical result from this function:

```
Get ( PreferencesPath )
// macOS result = /Macintosh HD/Users/admin/Library/Preferences/
// Windows result = /C:/Users/Admin/AppData/Local/
```

Get (PrinterName)

This function returns the name of the default printer.

Format Get (PrinterName)

Result A text string containing the name of the printer.

This example shows a typical result from this function:

```
Get ( PrinterName )    // result = "HP LaserJet 8150 Series"
```

Get (QuickFindText)

This function returns the text that is currently entered in the Quick Find text box in the toolbar.

Format Get (QuickFindText)

Result A text string containing the current Quick Find text.

This example shows a typical result from this function:

```
Get ( QuickFindText )    // result = "New York"
```

Get (RecordAccess)

This function returns a value indicating the user's access to the current record based on the privilege settings of the account they used to log into the database.

Format Get (RecordAccess)

Result A number indicating their access level. The possible values are:

- 0 = No access to view or edit the record
- 1 = Some or all access to view the record
- 2 = Some or all access to edit the record

This example shows a typical result from this function:

```
Get ( RecordAccess )    // result = 2
```

Get (RecordID)

This function returns the unique FileMaker ID for the current record.

Format Get (RecordID)

Result A number indicating the id for the record.

This example shows a typical result from this function:

```
Get ( RecordID )            // result = 256859
```

Get (RecordModificationCount)

This function returns a count of the total number of times a particular record has been modified and committed.

Format Get (RecordModificationCount)

Result A number representing the count of past modifications.

This example shows a typical result from this function:

```
Get ( RecordModificationCount ) // result = 4
```

Get (RecordNumber)

This function returns the current record's position within the found set of the current window.

Format Get (RecordNumber)

Result A number indicating the position of the record in the found set.

This example shows a typical result from this function:

```
Get ( RecordNumber )        // result = 15
```

Get (RecordOpenCount)

This function returns the number of records (local and related) in the current window that have uncommitted modifications.

Format Get (RecordOpenCount)

Result A number indicating the records with modifications pending.

This example shows a typical result from this function:

```
Get ( RecordOpenCount ) // result = 2
```

Get (RecordOpenState)

This function returns the current state of a record indicating if it is committed (closed), new, or modified.

Format Get (RecordOpenState)

Result A number indicating the current state. The available options are:

0 = a committed (closed) record

1 = a new, uncommitted record

2 = a modified, uncommitted record

This example shows a typical result from this function:

```
Get ( RecordOpenState ) // result = 2
```

Get (RegionMonitorEvents)

This function returns information about events that triggered the script specified by a `Configure Region Monitor Script` step monitoring for iBeacon encounters.

Format Get (RegionMonitorEvents)

Result The name of the region monitor, a timestamp indicating when the iOS device entered or exited the region being monitored, and a numeric Boolean indicating if the device entered the region (1) or exited the region (2).

More Info For more information on iBeacons, visit:
<https://developer.apple.com/ibeacon/>.

This examples shows results that indicate that the iOS device entered the region of an iBeacon named `StoreBeacon` at 3:24 PM on March 15, 2017, and then exited the region 20 minutes later.

```
Get ( RegionMonitorEvents )
// result =
StoreBeacon
15/3/2017 3:24 PM;1
15/3/2017 3:44 PM;0
```

Get (RequestCount)

This function returns a count of the number of find requests defined in the current window.

Format `Get (RequestCount)`

Result A number indicating the number of requests that exist.

This example shows a typical result from this function:

```
Get ( RequestCount )    // result = 3
```

Get (RequestOmitState)

This function indicates whether omit is selected for the current find request or not.

Format `Get (RequestOmitState)`

Result A Boolean indicating if omit is selected or not.

This example shows a typical result from this function:

```
Get ( RequestOmitState )    // result = 0
```

Get (ScreenDepth)

This function returns the color depth of the main script, which is the number of bits required to represent the color of a single pixel.

Format `Get (ScreenDepth)`

Result A number indicating the bit depth of the screen.

This example shows a typical result from this function:

```
Get ( ScreenDepth )    // result = 32
```

Get (ScreenHeight)

This function returns the height of the computer's monitor screen.

Format `Get (ScreenHeight)`

Result A number indicating the height of the computer monitor in points. If the computer has multiple screens, it will return the width for the screen enclosing the current window. If the current window spans multiple screens, it will return the width of the screen that contains most the window's width.

This example shows a typical result from this function:

```
Get ( ScreenHeight )    // result = 1440
```

Get (ScreenScaleFactor)

This function returns the scale factor of the current device screen.

Format Get (ScreenScaleFactor)

Result An integer indicating the scale factor based on the relative pixel density of the current screen.
 1 = a device without a Retina display (also the default for Custom Web Publishing)
 2 = a Mac with a Retina display or an iPhone 6
 3 = an iPhone 6 Plus

This example shows a typical result from this function:

```
Get ( ScreenScaleFactor ) // result = 1
```

Get (ScreenWidth)

This function returns the width of the computer's monitor screen.

Format Get (ScreenWidth)

Result A number indicating the width of the computer monitor in points. If the computer has multiple screens, it will return the width for the screen enclosing the current window. If the current window spans multiple screens, it will return the width of the screen that contains most of the window's width.

This example shows a typical result from this function:

```
Get ( ScreenWidth )     // result = 2560
```

Get (ScriptAnimationState)

This function returns the current layout animation setting (FileMaker Go only).

Format Get (ScriptAnimationState)

Result A Boolean indicating the current state of the layout animation setting:
 0 = layout animations is false
 1 = layout animations is true

This example shows a typical result from this function:

```
Get ( ScriptAnimationState )     // result = 1
```

Get (ScriptName)

This function returns the name of the script that is currently running or paused.

Format	Get (ScriptName)
Result	A text string containing the name of the script.

This example shows a typical result from this function:

```
Get ( ScriptName )    // result = "Create New Contact"
```

Get (ScriptParameter)

This function returns the parameter that was passed to the current script.

Format	Get (ScriptParameter)
Result	A text string containing the contents that were passed to the script.

This example detects the absence of a script parameter and displays an error.

```
If [ Get ( ScriptParameter ) = "" ]
    Show Custom Dialog [ "Error: No Script Parameter provided." ]
    Exit Script [ ]
End If
```

■ **Note** Script parameters and results are discussed in Chapter 34, “Exchanging Data Between Scripts.”

Get (ScriptResult)

This function returns the result to a script from the last sub-script that it ran.

Format	Get (ScriptResult)
Result	Any data the script returns including text, number, date, time, timestamp, or container.

This example shows a typical result from this function:

```
Get ( ScriptResult )    // result = "Task Complete"
```

This function shows a script using an If statement to take conditional action depending on the results from a sub-script. For example, an error in the sub-script might require a different course of action or an alert dialog for the user. This can be accomplished by placing a few extra script steps after a Perform Script step.

```
Perform Script [ "Generate Reports" ]
If [ Get ( ScriptResult ) = "Error" ]
    Show Custom Dialog [ "The report script encountered an error." ]
    Exit Script [ Text Results: ]
End If
```

Get (SortState)

This function returns the current sort state of the found set of records in the current window.

Format Get (SortState)

Result A number indicating the sort state. The possible values are:

- 0 = unsorted
 - 1 = sorted
 - 2 = semi-sorted
-

This example shows a typical result from this function:

```
Get ( SortState )    // result = 1
```

■ **Note** When records are imported into a found set of sorted records, it will result in a semi-sorted state, meaning that some records are sorted and some are not. A script can use this function to determine such a state and use the Sort script step to resort the records.

Get (StatusAreaState)

This function returns the current state of the status toolbar.

Format Get (StatusAreaState)

Result A number indicating the toolbar state. The possible values are:

- 0 = hidden
 - 1 = visible
 - 2 = visible and locked
 - 3 = hidden and locked
-

This example shows a typical result from this function:

```
Get ( StatusAreaState )    // result = 0
```

Get (SystemDrive)

This function returns the name of the hard drive running the current operating system.

Format Get (SystemDrive)

Result The macOS startup volume name or the Windows drive letter.

This example shows a typical result from this function:

```
Get ( SystemDrive ) // result = "Macintosh HD"
```

Get (SystemIPAddress)

This function returns a list of active Internet Protocol (IP) addresses for the current computer. The results may include multiple addresses if the computer is connected to multiple networks, such as Ethernet and WiFi.

Format Get (SystemIPAddress)

Result A carriage return-delimited list of addresses for the current computer.

This example shows a typical result from this function:

```
Get ( SystemIPAddress )
// result =
    10.0.1.27
    10.0.1.5
```

Get (SystemLanguage)

This function returns the name of the language set for the computer in the macOS System Preferences, Windows Control Panel, or the language of the Web Browser in WebDirect.

Format Get (SystemLanguage)

Result An English text string indicating the name of the language currently used.

This example shows a typical result from this function:

```
Get ( SystemLanguage ) // result = "English"
```

Get (SystemNICAddress)

This function returns the hardware address of all installed Network Interface Controller (NIC) circuit board cards that allow the current user's computer to be connected to a network. This can identify the specific hardware that a user is accessing a database from.

Format Get (SystemNICAddress)

Result A text string containing a carriage return-delimited list of network addresses for the computer's hardware.

This example shows a typical result from this function:

```
Get ( SystemNICAddress )
// result =
    4c:37:04:1j:67:2b
    dc:be:88:10:71:c2
    f6:9f:64:44:hd:12
    8e:00:5d:10:k7:00
    8e:00:5d:10:k7:01
    8e:00:5d:10:k7:04
    8e:00:5d:10:k7:05
```

Get (SystemPlatform)

This function returns the platform of the current device and can be used to create branching scripts that provide different functionality depending on the device.

Format	Get (SystemPlatform)
Result	A number indicating the current platform. Available values are: 1 = Intel-based Mac 2 = Windows 3 = iOS Device (iPhone or iPad) 4 = FileMaker WebDirect

This example shows a typical result from this function:

```
Get ( SystemPlatform ) // result = 1
```

Get (SystemVersion)

This function returns the version number of the current operating system or device.

Format	Get (SystemVersion)
Result	A text string containing the version number or name of device or web browser upon which the FileMaker database is running.

This example shows a typical result from this function:

```
Get ( SystemVersion ) // result = 10.11.5
Get ( SystemVersion ) // result = 10.12.4
```

Get (TemporaryPath)

This function returns a path to an temporary session folder automatically generated by FileMaker Pro application and the computer system on which it is running. This path can be used as a staging location to which to export files that will be emailed or otherwise processed when ready. Once the FileMaker application is shut down, the folder is automatically deleted.

Format Get (TemporaryPath)

Result A text string containing a path to a temporary folder. The exact path will vary depending on the type and version of the operating system.

This example shows a typical result from this function:

```
Get ( TemporaryPath )
// result = /Macintosh HD/var/folders/79/4gh7yxxj7zd1xpd2dcqm0qw80000gn/T/S10/
```

Get (TextRulerVisible)

This function indicates if the text ruler is visible or hidden.

Format Get (TextRulerVisible)

Result A Boolean indicating if the ruler is visible (1) or hidden (0).

This example shows a typical result from this function:

```
Get ( TextRulerVisible ) // result = 0
```

Get (TotalRecordCount)

This function returns the total count of records in the current table.

Format Get (TotalRecordCount)

Result A number indicating the total number of records in the table (including all records, even those excluded from the current found set).

This example shows a typical result from this function:

```
Get ( TotalRecordCount ) // result = 1523
```

Get (TouchKeyboardState)

This function determines if the touch keyboard will be displayed automatically when needed or not. This feature is only supported in FileMaker Go and Windows.

Format Get (TouchKeyboardState)

Result A Boolean indicating the state of the touch keyboard:
 0 = the touch keyboard is disabled
 1 = the touch keyboard is enabled

This example shows a typical result from this function:

```
Get ( TouchKeyboardState )      // result = 1
```

Get (TriggerCurrentPanel)

This function will return information about the panel being switched from when using the `OnPanelSwitch` script trigger (Chapter 37). This allows a script to identify which panel the user was on prior to requesting navigation to a different one.

Format `Get (TriggerCurrentPanel)`

Result A text string containing the number of the tab panel or slide panel being switched from. If the object has a name assigned, it is included on a second paragraph.

This example assumes a tab control named `Contact Details` with four tabs and an `OnPanelSwitch` script trigger that runs a script with a step containing the formula below. The user, viewing tab 3, clicks on tab 1. The result will be the number of the tab they were viewing (3) and the name of that tab control (`Contact Details`) as shown here:

```
Get ( TriggerCurrentPanel )
// result =
    3
    Contact Details
```

Get (TriggerExternalEvent)

This FileMaker Go function returns a number indicating the event that activated an `OnExternalCommandReceived` script trigger (Chapter 37).

Format `Get (TriggerExternalEvent)`

Result A number indicating the source of the external command. The available options are:

- 0 = Unknown
- 1 = RemotePlayMedia
- 2 = RemotePause
- 3 = RemoteTogglePlayPause
- 4 = RemotePlayNext
- 5 = RemotePlayPrevious
- 6 = RemoteSeek (start or end seeking forward or backwards)
- 7 = RemoteStop

This example shows a typical result from this function:

```
Get ( TriggerExternalEvent )   // result = 4
```

Get (TriggerGestureInfo)

The `Get (TriggerGestureInfo)` function returns information about a gesture on an iOS or Windows device that activates the `OnGestureTap` script trigger (Chapter 37).

Format `Get (TriggerGestureInfo)`

Result A carriage return-delimited list of the following values:

The word “Tap” indicating the script was activated by an `OnGestureTap` script trigger

A number indicating the number of taps received (one or two)

A number indicating how many fingers were involved

A number indicating the x coordinate in the window where the gesture occurred (center point when multiple figures involved)

A number indicating the y coordinate in the window where the gesture occurred (center point when multiple figures involved)

The following combinations are possible:

A single tap with 1-3 fingers (iOS)

A double-tap with one figure (iOS)

A tap with two fingers (Windows)

This example shows a typical result from this function:

```
Get ( TriggerKeystroke )
// result =
    Tap
    1
    2
    185
    312
```

Get (TriggerKeystroke)

This function returns the character that activated an `OnObjectKeystroke` or `OnLayoutKeystroke` script trigger (Chapter 37).

Format `Get (TriggerKeystroke)`

Result The text character typed.

This example shows a typical result from this function:

```
Get ( TriggerKeystroke )            // result = "A"
```

Get (TriggerModifierKeys)

This function returns the modifier keys that were held down at the moment the script trigger was activated. This function returns the same information as `Get (ActiveModifierKeys)` but may be received after the user has already let go of the keys so it is more useful when running a script trigger.

Get (TriggerTargetPanel)

This function will return information about the panel being switched to when using the `OnPanelSwitch` script trigger (Chapter 37). This allows a script to identify which panel the user has clicked on.

Format `Get (TriggerTargetPanel)`

Result A text string containing the number of the tab panel or slide panel being switched to. If the object has a name assigned it is included on a second paragraph.

This example assumes a tab control named `Contact Details` with four tabs and an `OnPanelSwitch` script trigger that runs a script with a step containing the formula below. The user, viewing tab 3, clicks on tab 1. The result will be the number of the tab they were viewing (3) and the name of that tab control (`Contact Details`) as shown here:

```
Get (TriggerTargetPanel)
// result =
    1
    Contact Details
```

Get (UserCount)

This function returns the number of users actively connected to a database file.

Format `Get (UserCount)`

Result A number indicating the user count.

This example shows a typical result from this function:

```
Get ( UserCount )            // result = 3
```

Get (UserName)

This function returns the name of the current user, which is specified in Preferences.

Format `Get (UserName)`

Result A text string containing the name of the user.

This example shows a typical result from this function:

```
Get ( UserName )            // result = Joanne Miller
```

Get (UseSystemFormatsState)

This function indicates if the `Format > Use System Formats` format menu bar is on or off (Chapter 3, “File Options”).

Format `Get (UseSystemFormatsState)`

Result A Boolean indicating if the menu option is on (checked) or off (unchecked).

This example shows a typical result from this function:

```
Get ( UseSystemFormatsState )     // result = 0
```

Get (UUID)

This function generates a Universally Unique Identifier (UUID), which is a non-guaranteed unique id number that can be used to decentralize certain aspects of database record tracking. If used as the primary key for database records, users in a duplicate file who are in a remote location or offline could create new records without causing duplicate ids when their records are later imported into the central database.

Format `Get (UUID)`

Result A unique 16-byte (128-bit) string representing the unique identifier.

More Info For more information about UUID visit:
https://en.wikipedia.org/wiki/Universally_unique_identifier.

This example shows a typical result from this function:

```
Get ( UUID )     // result = 1E01C315-4807-41C5-B233-E7F4F19EAD89
```

Get (WindowContentHeight)

This function returns the height of the visible content area of the current window.

Format `Get (WindowContentHeight)`

Result A number indicating the height of the portion of the window that can contain layout content; it excludes the window border, the zoom control area, and page margins, making it slightly smaller than the number returned by `Get (WindowHeight)`.

This example shows a typical result from this function:

```
Get ( WindowContentHeight )     // result = 1560
```

Get (WindowContentWidth)

This function returns the width of the visible content area of the current window.

Format Get (WindowContentWidth)

Result A number indicating the width of the portion of the window that can contain layout content; it excludes the window border, the scroll bars, and page margins, making it slightly smaller than the number returned by Get (WindowWidth).

This example shows a typical result from this function:

```
Get ( WindowContentWidth )     // result = 2430
```

Get (WindowDesktopHeight)

This function returns the height of the computer's desktop space.

Format Get (WindowDesktopHeight)

Result A number indicating the height of the desktop space in points.

This example shows a typical result from this function:

```
Get ( WindowDesktopHeight )     // result = 1417
```

Get (WindowDesktopWidth)

This function returns the width of the computer's desktop space.

Format Get (WindowDesktopWidth)

Result A number indicating the width of the desktop space in points. If the computer has multiple screens, it will return the width for the screen enclosing the current window. If the current window spans multiple screens, it will return the width of the screen that contains the majority of the window's width.

This example shows a typical result from this function:

```
Get ( WindowDesktopWidth )     // result = 2560
```

Get (WindowHeight)

This function returns the height of the current window.

Format Get (WindowHeight)

Result A number indicating the height of the window in points.

This example shows a typical result from this function:

```
Get ( WindowHeight ) // result = 852
```

Get (WindowLeft)

This function returns the distance from the left of the screen to the outside left of the current window.

Format Get (WindowLeft)

Result A number indicating the distances in points for the left position of the current window.

This example shows a typical result from this function:

```
Get ( WindowLeft) // result = 35
```

Get (WindowMode)

This function returns the mode of the current window.

Format Get (WindowMode)

Result A number indicating the current mode of the window. The available values are:

0 = Browse mode

1 = Find mode

2 = Preview mode

3 = Printing is in progress

4 = Layout mode (only from FileMaker Pro Advanced Edit Expression dialog from the Data Viewer interface)

If the current window is in layout mode, a script using this function will cause the window to revert to browse mode and return a zero.

This example shows a typical result from this function:

```
Get ( WindowMode ) // result = 0
```

Get (WindowName)

This function returns the name of the current window.

Format Get (WindowName)

Result A text string containing the title of the window.

This example shows a typical result from this function:

```
Get ( WindowName ) // result = "Contacts"
```

Get (WindowOrientation)

This function returns the orientation of the current window, which can be most useful with FileMaker Go on a tablet or phone that can be rotated into different views.

Format	Get (WindowOrientation)
Result	A number indicating the orientation of the window. The available values are: -2 = landscape left -1 = landscape right 0 = square 1 = portrait 2 = portrait upside down

This example shows a typical result from this function:

```
Get ( WindowOrientation ) // result = 1
```

Get (WindowStyle)

This function returns the style of the current window.

Format	Get (WindowStyle)
Result	A number indicating the style of the window. The available values are: 0 = a document window 1 = a floating document window 2 = a dialog window

This example shows a typical result from this function:

```
Get ( WindowStyle ) // result = 0
```

Get (WindowTop)

This function returns the distance from the bottom of the menu bar to the outside top of the current window.

Format	Get (WindowTop)
Result	A number indicating the distances in points for the top of the current window.

This example shows a typical result from this function:

```
Get ( WindowTop ) // result = 78
```

Get (WindowVisible)

This function determines if the current window is visible or not.

Format Get (WindowVisible)

Result A Boolean indicating if the window is visible or hidden.

This example shows a typical result from this function:

```
Get ( WindowVisible ) // result = 1
```

Get (WindowWidth)

This function will return the width of the current window.

Format Get (WindowWidth)

Result A number indicating the width of the window in points.

This example shows a typical result from this function:

```
Get ( WindowWidth )    // result = 1373
```

Get (WindowZoomLevel)

This function returns the zoom percentage of the current window.

Format Get (WindowZoomLevel)

Result A number indicating the percentage of zoom to which the user has the window set.

This example shows a typical result from this function:

```
Get ( WindowZoomLevel ) // result = 100
```

Summary

In this chapter, we explored all the built-in functions for accessing many different static values.

CHAPTER 23



Design Functions

Design functions are used to acquire information about the structure of a database file.

Design Function Reference

FileMaker provides the following built-in design functions:

- DatabaseNames
- FieldBounds
- FieldComment
- FieldIDs
- FieldNames
- FieldRepetitions
- FieldStyle
- FieldType
- GetNextSerialValue
- LayoutIDs
- LayoutNames
- LayoutObjectName
- RelationInfo
- ScriptIDs
- ScriptNames
- TableIDs
- TableNames
- ValueListIDs
- ValueListItems
- ValueListNames
- WindowNames

DatabaseNames

The DatabaseNames function returns a list of database files currently open.

Format	DatabaseNames
Parameters	Not applicable.
Result	A carriage return-delimited list of open database names.

This example shows a typical result from this function:

```
DatabaseNames
// result =
    Contacts
    Invoices
    Projects
```

FieldBounds

The FieldBounds function returns the location of a field on a layout.

Format	FieldBounds (fileName ; layoutName ; fieldName)
Parameters	<p>fileName = a text expression or field containing the name of a database file.</p> <p>layoutName = a text expression or field containing the name of a layout.</p> <p>fieldName = a text expression or field containing a text-based field reference.</p>
Result	Four space-delimited values indicating the distance of the field from the top left corner of the layout in points, in the following order: left, top, right, bottom.

This example shows a typical result from this function:

```
FieldBounds ( "Contacts" ; "People - Data Entry" ; GetFieldName ( People::First Name ))
// result = 25 50 380 70
```

Combine this result with other functions to calculate the width and height of the field. The example below places the field bounds into a variable named data and uses the GetValue function to parse each measurement into a separate variable. These are then used to calculate the width and height and place them each into a variable that can then be used by other code.

```
Let ( [
    fieldName = GetFieldName ( People::First Name ) ;
    data = FieldBounds ( "Contacts" ; "People - Data Entry" ; fieldName ) ;
    left = GetValue ( data ; 1 ) ;
    top = GetValue ( data ; 2 ) ;
    right = GetValue ( data ; 3 ) ;
```

```

    bottom = GetValue ( data ; 4 ) ;
    width = right - left ;
    height = bottom - top
] ;
    // Add additional code
)

```

■ **Tip** To help future-proof your code, use dynamic references when functions require a file or field name. Use `Get (FileName)` instead of “Contacts” and `GetFieldName (People::First Name)` instead of “People::First Name” to allow your code to dynamically adapt to future name changes.

FieldComment

The `FieldComment` function returns the comment for a field from the field definitions dialog, if one has been entered.

Format	<code>FieldComment (fileName ; fieldName)</code>
Parameters	<p><code>fileName</code> = a text expression or field containing the name of a database file.</p> <p><code>fieldName</code> = a text expression or field containing a text-based field reference.</p>
Result	A text string containing the specified field’s comment.

This example shows a typical result from this function:

```

FieldComment ( "Contacts" ; GetFieldName ( People::Status ))
// result = "This field is used to indicate if a contact record is active or not"

```

FieldIDs

The `FieldIDs` function returns a list of the FileMaker ids of every field on a layout.

Format	<code>LayoutIDs (fileName ; layoutName)</code>
Parameters	<p><code>fileName</code> = a text expression or field containing the name of a database file.</p> <p><code>layoutName</code> = a text expression or field containing the name of a layout or a table.</p>
Result	A carriage return-delimited list of IDs of every field appearing on the specified layout. If a table name is provided for the <code>layoutName</code> parameter and there is no layout by the same name, the function will return a list of all field IDs for that table, regardless of layout. If fields have been deleted, the list of values returned may be non-consecutive.

This example shows a typical result from this function:

```
FieldIDs ( "Contacts" ; "People" )
// result =
    1
    2
    5
   10
   12
  Etc.
```

FieldNames

The FieldNames function returns a list of the name of every field on a layout.

Format	FieldNames (fileName ; layoutName)
Parameters	fileName = a text expression or field containing the name of a database file. layoutName = a text expression or field containing the name of a layout or a table.
Result	A carriage return-delimited list of names of every field appearing on the specified layout. If a table name is provided for the layoutName parameter and there is no layout by the same name, the function will return a list of all field names for that table, regardless of layout.

This example shows a typical result from this function:

```
FieldNames ( "Contacts" ; "People - List" )
// result =
  First Name
  Last Name
  Record ID
  Status
  Notes
  Etc.
```

FieldRepetitions

The FieldRepetitions function returns the number and orientation of repetitions applied to a field on a layout.

Format	FieldRepetitions (fieldName ; layoutName ; fieldName)
Parameters	fileName = a text expression or field containing the name of a database file. layoutName = a text expression or field containing the name of a layout. fieldName = a text expression or field containing a text-based field reference.
Result	A text value containing two space-delimited values: The number of repetitions assigned to the field on the layout specified The orientation of those repetitions: horizontal or vertical Fields defined to be non-repeating will return a value of 1 vertical

This example shows a typical result from this function:

```
FieldRepetitions ( Get ( FileName ) ; Get ( LayoutName ) ; GetFieldName ( People::Status ) )
// result = 1 Vertical
```

FieldStyle

The `FieldStyle` function returns the formatting applied to a field on a layout.

Format	<code>FieldStyle (fileName ; layoutName ; fieldName)</code>
Parameters	<p><code>fileName</code> = a text expression or field containing the name of a database file. <code>layoutName</code> = a text expression or field containing the name of a layout. <code>fieldName</code> = a text expression or field containing a text-based field reference.</p>
Result	<p>A text string indicating the format of the field on the layout specified. The available options are:</p> <p><code>Standard</code> = a standard field without special formatting <code>Scrolling</code> = a standard field formatted with a vertical scroll <code>Popuplist</code> = a field formatted with a drop-down list <code>Popupmenu</code> = a field formatted with a pop-up menu <code>Checkbox</code> = a field formatted as a check box <code>RadioButton</code> = a field formatted as a radio button <code>Calendar</code> = a field formatted with a drop-down calendar</p>

This example shows a typical result from this function:

```
FieldStyle ( Get ( FileName ) ; Get ( LayoutName ) ; GetFieldName ( People::Status ) )
// result = Checkbox
```

FieldType

The `FieldType` function returns information about a field including data type, index status, storage settings, and number of repetitions.

Format	<code>FieldType (fileName ; fieldName)</code>
Parameters	<p><code>fileName</code> = a text expression or field containing the name of a database file. <code>fieldName</code> = a text expression or field containing a text-based field reference.</p>
Result	<p>a text value containing four space-delimited values:</p> <p><code>Status</code> — one of the following values: <code>Standard</code>, <code>StoredCalc</code>, <code>Summary</code>, <code>UnstoredCalc</code>, <code>External(Secure)</code>, <code>External(Open)</code>, or <code>Global</code> <code>Type</code> — the data type of the field or of the calculation result: <code>text</code>, <code>number</code>, <code>date</code>, <code>time</code>, <code>timestamp</code>, or <code>container</code> <code>Index Status</code> — <code>Indexed</code> or <code>Unindexed</code> <code>Repetitions</code> — a number representing the maximum number of repetitions defined for the field (1 for non-repeating fields)</p>

This example shows a typical result from this function:

```
FieldType ( "Contacts" ; GetFieldName ( People::Record ID ) )
// result = Standard Text Indexed 1
```

The function can be combined with other functions to extract specific values. This example uses a Let statement to place the field's information into a variable named data and then uses the GetValue function to parse the various components into separate variables:

```
Let ( [
  data = FieldType ( Get ( FileName ) ; GetFieldName ( Example Input ) ) ;
  data = Substitute ( data ; " " ; "¶" ) ;
  storage = GetValue ( data ; 1 ) ;
  type = GetValue ( data ; 2 ) ;
  index = GetValue ( data ; 3 ) ;
  repetitions = GetValue ( data ; 4 )
] ;
  // Add additional code
)
```

GetNextSerialValue

The GetNextSerialValue function returns the next serial number for a field.

Format	GetNextSerialValue (fileName ; fieldName)
Parameters	fileName = a text expression or field containing the name of a database file. fieldName = a text expression or field containing a text-based field reference for a field with an auto-entered serial number.
Result	A number indicating the next serial number that will be assigned for the field specified if a new record were to be created. If the field does not have an auto-entered serial number assigned, an empty string will be returned. Since this does not reserve the next number, in a multi-user environment, the number returned from this function would be obsolete if another user creates a new record after this function returns a value.

This example shows a typical result from this function:

```
GetNextSerialValue ( "Contacts" ; GetFieldName ( People::Record ID ) )
// result = 1562
```

LayoutIDs

The LayoutIDs function returns a list of the ID of every layout in a database file.

Format	LayoutIDs (fileName)
Parameters	fileName = a text expression or field containing the name of a database file.
Result	A list of carriage return-delimited layout IDs. If layouts have been deleted, the list of values returned may be non-consecutive.

This example shows a typical result from this function:

```
LayoutIDs ( "Contacts" )
// result =
    1
    2
    5
   10
   12
   13
```

LayoutNames

The `LayoutNames` function returns a list of the names of every layout in a database file.

Format	<code>LayoutNames (fileName)</code>
Parameters	<code>fileName</code> = a text expression or field containing the name of a database file.
Result	A carriage return-delimited list of layout names.

This example shows a typical result from this function:

```
LayoutNames ( "Contacts" )
// result =
    Company - List
    People - List
    People - Data Entry
    People - Notes
    Correspondence - List
    Phone Numbers - List
```

LayoutObjectNames

The `LayoutObjectNames` function returns a list of the names of every named object on a layout.

Format	<code>LayoutObjectNames (fileName ; layoutName)</code>
Parameters	<code>fileName</code> = a text expression or field containing the name of a database file. <code>layoutName</code> = a text expression or field containing the name of a layout file.
Result	A carriage return-delimited list of every object with a name assigned on the layout specified. Any object with no name assigned is not included.

This example shows a typical result from this function:

```
LayoutObjectName ( "Contacts" ; "People - Data Entry" )
// result =
    Correspondence Portal
    Company Website Viewer
    Phone Numbers Portal
```

RelationInfo

The RelationInfo function returns data on the relationship(s) connected to a specified table.

Format	RelationInfo (fileName ; tableName)
Parameters	fileName = a text expression or field containing the name of a database file. tableName = a text expression or field containing the name of a table.
Result	A list of four carriage return-delimited pieces of information about each relationship, separated from other lists by two carriage returns: The name of file containing the base table (not the occurrence) with a prefix of "Source:" The name of the related table occurrence with a prefix of "Table:" The relationship option check boxes that were selected on the side of the related table in the Edit Relationship dialog; Delete, Create, Sorted; delimited with a space after the "Options:" prefix A list of the field connections that form the relationship, one per paragraph

This example shows a typical result from this function:

```
RelationInfo ( "Contacts" ; "Person" )
// result =
    Source:Contacts
    Table:Person | Phone Numbers
    Options:Delete Create Sorted
    Person::Record ID = Person | Phone Number::Record Parent ID

    Source:Contacts
    Table:Person | Correspondence
    Options:
    Person::Record ID = Person | Correspondence::Record Parent ID
    Person::Person Correspondence Status = Person | Correspondence::Record Status
```

ScriptIDs

The ScriptIDs function returns a list of the ID of every script in a database file.

Format	ScriptIDs (fileName)
Parameters	fileName = a text expression or field containing the name of a database file.
Result	A list of carriage return-delimited script IDs. If scripts have been created and deleted, the list may be non-consecutive.

This example shows a typical result from this function:

```
ScriptIDs ( "Contacts" )
// result =
    9
   14
   15
   26
```

ScriptNames

The ScriptNames function returns a list of the names of every script in a database file.

Format	ScriptNames (fileName)
Parameters	fileName = a text expression or field containing the name of a database file.
Result	A carriage return-delimited list of script names.

This example shows a typical result from this function:

```
ScriptNames ( "Contacts" )
// result =
    Create New Contact
    Create New Correspondence
    Find Contacts at the Same Company
    Etc.
```

TableIDs

The TableIDs function returns a list of the IDs of every table in a database file.

Format	TableIDs (fileName)
Parameters	fileName = a text expression or field containing the name of a database file.
Result	A list of carriage return-delimited table IDs. If tables have been deleted, the list of values returned may be non-consecutive.

This example shows a typical result from this function:

```
TableIDs ( "Contacts" )
// result =
    2
    4
   10
   11
```

TableNames

The TableNames function returns a list of the names of every table in a database file.

Format	TableNames (fileName)
Parameters	fileName = a text expression or field containing the name of a database file.
Result	A carriage return-delimited list of table names.

This example shows a typical result from this function:

```
TableNames ( "Contacts" )
// result =
    Company
    Contact
    Phone Numbers
    Correspondence
```

ValueListIDs

The ValueListIDs function returns a list of the IDs of every values list in a database file.

Format	ValueListIDs (fileName)
Parameters	fileName = a text expression or field containing the name of a database file.
Result	A list of carriage return-delimited value list IDs. If value lists have been deleted, the list of values returned may be non-consecutive.

This example shows a typical result from this function:

```
ValueListIDs ( "Contacts" )
// result =
    1
   15
   18
   31
```

ValueListItems

The ValueListItems function returns a list of the values that make up a value list.

Format	ValueListItems (fileName ; valueList)
Parameters	fileName = a text expression or field containing the name of a database file. valueList = a text expression or field containing the name of a value list.
Result	A carriage return-delimited list of the values in the specified value list.

This example shows a typical result from this function:

```
ValueListItems ( "Contacts" ; "Phone Types" )
// result =
    Home
    Work
    Mobile
```

Create a more dynamic approach by nesting other functions. This example will get a list of value list names for the current file and then get the items for the first three lists:

```
Let ( [
    lists = ValueListNames ( Get ( FileName ) ) ;
    items1 = ValueListItems ( GetValue ( lists ; 1 ) ) ;
    items2 = ValueListItems ( GetValue ( lists ; 2 ) ) ;
    items3 = ValueListItems ( GetValue ( lists ; 3 ) )
] ;
    // additional code
)
```

ValueListNames

The ValueListNames function returns a list of the names of every values list in a database file.

Format	ValueListNames (fileName)
Parameters	fileName = a text expression or field containing the name of a database file.
Result	A carriage return-delimited list of value list names.

This example shows a typical result from this function:

```
ValueListNames ( "Contacts" )
// result =
    Status
    Phone Types
    Industry List
    Salutation Options
```

WindowNames

The WindowNames function returns a list of the names of open FileMaker Pro windows.

Format	When no parameters are specified: WindowNames Optionally include the name of a database file to only return a list of the windows open for that file: WindowNames (fileName)
Parameters	fileName = (optional) a text expression or field containing the name of a database file for which window names should be limited.
Result	A carriage return-delimited list of the name of every open window or every open window for the file specified.

This example shows a typical result from this function:

```
WindowNames
// result =
    Contacts
    Phone Numbers
    Correspondence
    Invoices
    Projects
```

Limit the results to a specific file by including the file name:

```
WindowNames ( "Contacts" )
// result =
    Contacts
    Phone Numbers
    Correspondence
```

Summary

In this chapter, we explored all the built-in functions for accessing various design information.

CHAPTER 24



Mobile Functions

Mobile functions are used to get information that is specific to a mobile device running a database on FileMaker Go.

Mobile Function Reference

FileMaker provides the following built-in mobile functions:

- Location
- LocationValues
- RangeBeacons

Location

The Location function returns the location of an iOS device running FileMaker Go.

Format	Without optional parameters: Location (accuracy) An optional parameter can be added to override the default timeout: Location (accuracy ; timeout)
Parameters	accuracy = a numeric expression or field that contains a number to represent a distance in meters for the desired accuracy of the results. timeout = (optional) a numeric expression or field containing a number that indicates the maximum number of seconds to wait for a response (default is 60).
Result	A list of comma-delimited values: Latitude Longitude Horizontal Accuracy (+/- in meters)

This example shows a typical result from this function:

```
Location ( 100 ; 70 )
// result =
    50.682532, -85.915032, 5
```

To extract a value from these results, combine this with other functions:

```
Let ( [
    data = Location ( 100 ; 70 ) ;
    data = Substitute ( data ; "," ; "¶" )
    latitude = GetValue ( data ; 1 ) ;
    longitude = GetValue ( data ; 2 ) ;
    accuracyHorizontal = GetValue ( data ; 3 )
] ;
    // Add additional code here
)
```

■ **Tip** To maximize speed and avoid battery consumption when using `Location` and `LocationValues`, use larger accuracy and smaller timeouts. It can take 40 seconds to return accuracy to 100 meters.

LocationValues

The `LocationValues` function returns a list of values containing the latitude, longitude, and altitude of an iOS device running FileMaker Go, including an evaluation of accuracy and minutes since the values were returned.

Format	Without optional parameters: <code>LocationValues (accuracy)</code> An optional parameter can be added to override the default timeout: <code>LocationValues (accuracy ; timeout)</code>
Parameters	<code>accuracy</code> = a numeric expression or field that contains a number to represent a distance in meters for the desired accuracy of the results. <code>timeout</code> = (optional) a numeric expression or field containing a number that indicates the maximum number of seconds to wait for a response (default is 60).
Result	A list of carriage return-delimited values: Latitude Longitude Altitude Horizontal Accuracy (+/- in meters) Vertical Accuracy (+/- in meters) Age of value in minutes

This example shows a typical result from this function:

```
LocationValues ( 5 )
// result =
    50.682532
   -85.915032
     5
     8
    0.004355
```

To extract a value from these results, combine this with other functions:

```
Let ( [
    data = LocationValues ( 5 ) ;
    latitude = GetValue ( data ; 1 ) ;
    longitude = GetValue ( data ; 2 ) ;
    accuracyHorizontal = GetValue ( data ; 3 ) ;
    accuracyVertical = GetValue ( data ; 4 ) ;
    age = GetValue ( data ; 5 )
] ;
    // Add additional code here
)
```

RangeBeacons

The RangeBeacons function returns a list of iBeacons and their proximity to an iOS device. iBeacons use Bluetooth Low Energy (BLE) to establish a region around an object, allowing iOS devices to determine when it has entered or left the region, along with an estimation of proximity to a beacon.

Format	<p>Without optional parameters:</p> <pre>RangeBeacons (UUID)</pre> <p>To control how the function response and with what information, several optional parameters can be added:</p> <pre>RangeBeacons (UUID ; timeout ; major ; minor)</pre>
Parameters	<p>UUID = a text string containing a Universally Unique Identifier (UUID) to search.</p> <p>timeout = (optional) a number indicating how long to wait before returning a value; default is 5 seconds.</p> <p>major = (optional) a text string containing an identification of a group of specified iBeacons.</p> <p>minor = (optional) a text string containing an identification of specified iBeacons within a group.</p>

(continued)

Result	<p>a carriage return-delimited list of nearby iBeacons with the following comma-space delimited values for each:</p> <p>UUID = the id of the current iBeacon. major = a value identifying the group of iBeacons. minor = a value indentifying specific iBeacons within the group. proximity = a number representing the proximity of the device to an iBeacon, from 0 to 3 indicating: Unknown, Immediate, Near, Far. accuracy = a number representing the accuracy in meters of the proximity value with any negative value indicating accuracy was indeterminate. rssi = a number representing the received signal strength in decibels of the iBeacon.</p>
More Info	<p>For more information about iBeacons visit: https://developer.apple.com/ibeacon/.</p>

This example shows a typical result from this function:

```
RangeBeacons ( "W5T5M11J-AAAA-BBBB-23T4-5H55A5BCD55E" )
// result =
W5T5M11J-AAAA-BBBB-23T4-5H55A5BCD55E, 1, 1, 2, 10, 5
W5T5M11J-AAAA-BBBB-23T4-5H55A5BCD55E, 1, 2, 3, 6, 3
W5T5M11J-AAAA-BBBB-23T4-5H55A5BCD55E, 2, 5, 0, -13, 2
```

Summary

In this chapter, we explored all the built-in functions for accessing information on a mobile device.

CHAPTER 25



Creating Custom Functions

When built-in functions don't provide the functionality that your database requires, and you find yourself reusing portions or entire calculation formulas repeatedly with little or no modification, it's time to create a custom function. Custom functions make possible the creation of open-ended, reusable code that reduces the complexity and repetition in the formulas that call them. This helps to condense code, making it more efficient and easier to navigate. As a result, it makes a solution more stable, easier to troubleshoot and more amenable to future expansion.

In this chapter, we will discuss the process of building and using custom functions with topics such as:

- Defining custom functions
- Exploring the custom function dialogs
- Building custom functions

Defining Custom Functions

A *custom function* is a developer named and defined formula that can be used from any formula throughout the database. With custom functions, *you* decide the *name* of the function, define its *parameters*, and write the *code* that is executed. Once installed, custom functions are identical in appearance and accessibility as built-in functions. The only difference is that custom functions can't have *optional parameters*.

Custom functions can improve code just like subroutines in other programming languages. They can be created from existing formulas by extracting some or all of a statement and adjusting it as needed into a generic and reusable function. They can be created from scratch as new open-ended functionality that anticipates usage from various contexts in a database. Any formula or portion of a formula that is or can be made generic enough to be widely useful, or which is used repeatedly within a single formula or throughout a database file, can and should be converted into a custom function.

For example, a database with multiple formulas that compile separate address fields into a single value for a label or envelope, would benefit from a custom functions that carried out this task and can be called repeatedly as follows:

```
createAddress ( first ; last ; street1 ; street2 ; city ; state ; zip )
```

■ **Note** Custom functions can only be created and edited within the FileMaker Pro Advanced application. However, once a function is installed in a database file using this application, they can be accessed and used by calculations and scripts running a database with any other application in the FileMaker family of software.

Exploring the Custom Function Dialogs

Custom functions are managed and edited from within two dialogs.

The Manage Custom Functions Dialog

Custom functions are created and managed from the Manage Custom Functions dialog, shown in Figure 25-1.

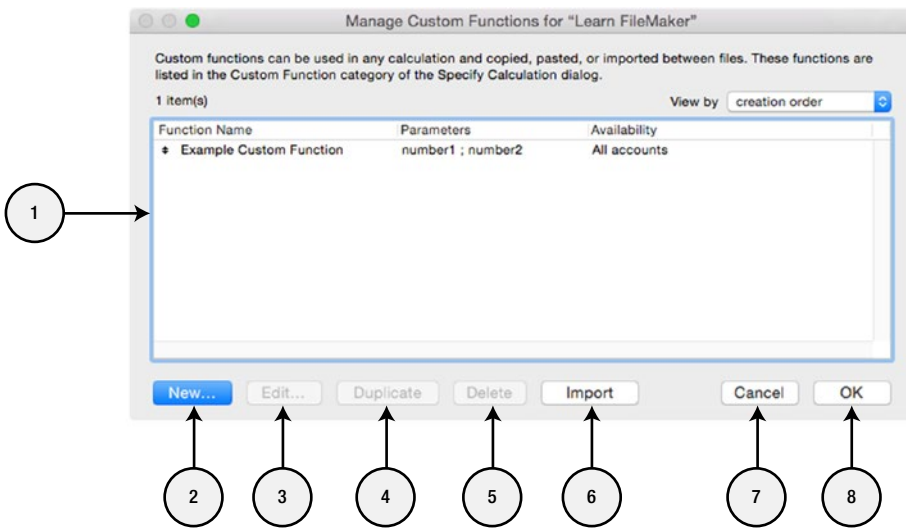


Figure 25-1. The Manage Custom Functions dialog

To open this dialog, select the File > Manage > Custom Functions menu item.

This dialog displays a list of all custom functions installed in the current database file. From here functions can be viewed, sorted, created, edited, duplicated, deleted, and imported using the following controls:

1. **Function list** — This area displays every function installed in the current database file. The list can be sorted by clicking on one of the three column headings or by selecting one of the following options from the **View by** pop-up list in the upper-right corner:
 - **Creation Order**
 - **Function Name**
 - **Custom Order** where the functions can be sorted by dragging items up or down the list
2. **New** — This button will open a new **Edit Custom Function** dialog.
3. **Edit** — This button will open the selected function in the **Edit Custom Function** dialog.
4. **Duplicate** — This button will create a duplicate copy of the selected function(s).
5. **Delete** — This button will delete the selected function(s).
6. **Import** — This button will allow one or more custom functions from a selected database to be imported into the current database.
7. **Cancel** — This button will close the dialog window with a warning that no changes will be saved.
8. **OK** — This button will close the dialog window, saving all changes.

Importing Custom Functions

To duplicate custom functions from one database to another, either use the **Copy** and **Paste** commands or use the **Import** button on the **Manage Custom Functions** dialog by following these steps:

1. Open the **Manage Custom Functions** dialog.
2. Click the **Import** button.
3. In the **Open File** dialog, navigate to the file containing the functions that you want to import into the current file.
4. An **Import Custom Functions** dialog will appear, listing every custom function available in the selected file.
5. Select the desired functions and then click **OK** to import them into the current file.

Review the results in the **Import Summary** dialog, which lists the number of items that were imported, how many were renamed due to a conflict, and how many errors were detected. It is a good idea to review each function to confirm that it imported correctly. Any functions that have a dependency on another function or field that is not available in the current file will be imported but the code will be commented out. These must be uncommented and corrected before they will operate as expected.

The Edit Custom Function Dialog

The Edit Custom Function dialog window, shown in Figure 25-2, provides the interface for editing custom functions in an old style Specify Formula dialog.

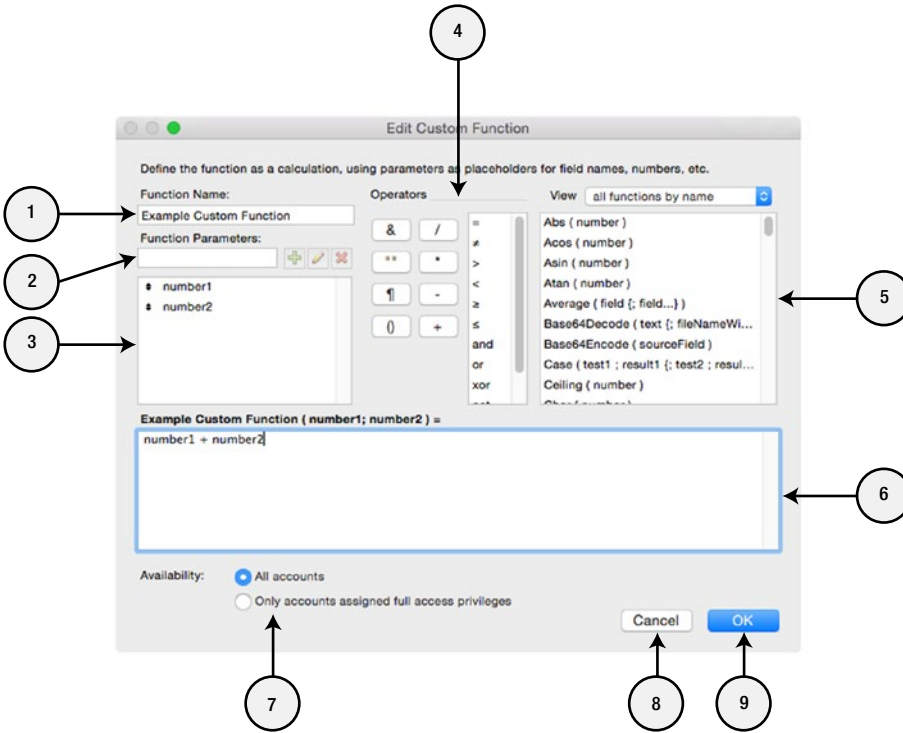


Figure 25-2. The Edit Custom Functions dialog window

To open this window, click on the New button on the Manage Custom Functions dialog. The Edit Custom Function dialog contains the following controls:

1. **Function Name** — This text area contains the name of the function, which will be used to call the function from other formulas.
2. **Parameter Name** — This text area contains the name of a new or currently selected parameter. Once a parameter name is entered or modified, the parameter can be created, updated, or deleted using the adjacent buttons.
3. **Parameter List** — This area contains the list of every parameter defined for the function. These can be rearranged by dragging up or down the list. Parameters are *positional*, so values included as parameters in a call to the function in other formulas will be inserted into the parameter at the corresponding position shown here. Changing the order of parameters will require rearranging the values in every existing call to the function for them to be inserted into the correct variable.

4. **Operators** — This area contains controls for the quick insertion of an operator into the code. The most popular operators are included as two rows of buttons with the remaining available in a list. A single-click on a button or a double-click on a list item will insert the corresponding operator into the current selection point of the function's formula.
5. **Functions** — This area contains a list of functions. By default, all functions are displayed sorted by name. However, the pop-up menu above the list contains options to display them categorized by type or to filter the list to a single category. The list not only includes all built-in functions but also any other custom functions and external functions from any installed plug-ins.
6. **Formula Text Area** — This text area is where the calculation formula is written. As with the standard *Specify Calculation* dialog, you can type, drag/drop, copy/paste, or insert content by double-clicking on the parameter or function list.
7. **Availability** — By default, any new custom function will be accessible to all accounts. However, selectively you can choose to make a function accessible to only accounts with full access (Chapter 40).
8. **Cancel** — This button will close the dialog window *without* saving changes.
9. **OK** — This button will close the dialog window, saving all changes.

Building Custom Functions

The process of building a custom function is like writing a calculation formula. The only differences are that custom functions can have parameters, can be recursive, and adding fields must be done manually and *very* carefully.

Creating a Simple Custom Function

Creating a custom function without parameters is identical to creating a regular calculation formula. Simply write the statements that expresses the desired functionality.

As a simple example, consider a function we will name *GetMonday* that will dynamically calculate the date for Monday of the current week. The formula will use the *Get (CurrentDate)* function to ascertain today's date and establish today's day number for the current week.

For example, imagine that the current date is a Friday, which is the *sixth* day of the week. Since we want to determine the corresponding Monday, the *second* day of the week, we can subtract 2 from 6 to arrive at the number of days we need to subtract from the current day in order to arrive at a Monday for the current week.

Using a *Let* statement to initialize a couple of local variables, we can accomplish this task with the following formula:

```
Let ( [
    dateToday = Get ( CurrentDate ) ;
    numAdjustment = DayOfWeek ( dateToday ) - 2
] ;
    dateToday - numAdjustment
)
```


To create the custom function, follow these steps:

1. Open the Manage Custom Functions dialog.
2. Click the New button to open the Edit Custom Function dialog.
3. Enter the name of the function as GetMonday.
4. Enter the formula.

Once entered, the custom function should appear as shown in Figure 25-3 and can be saved by clicking the OK button.

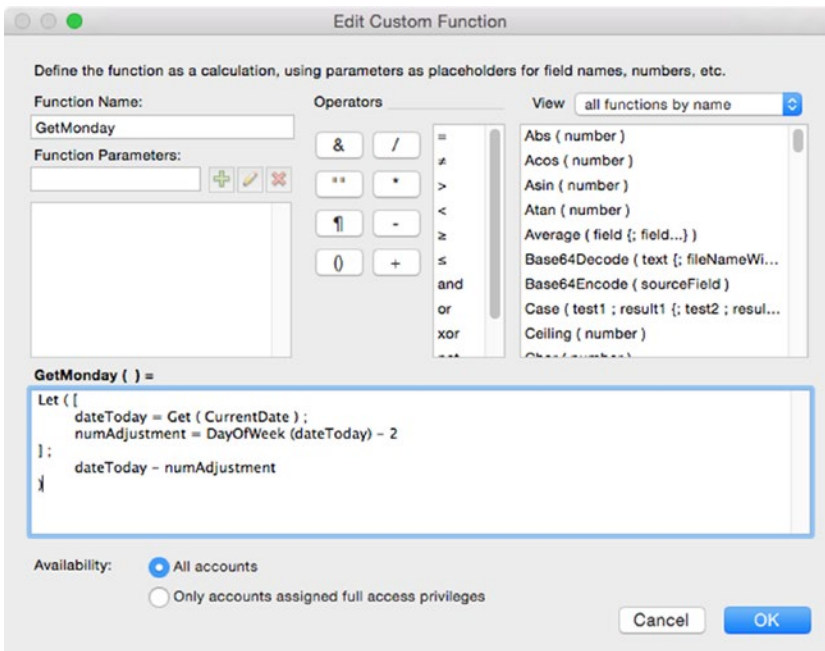


Figure 25-3. The GetMonday function, ready for saving

Once saved, any formula can call the function. Test this in the Sandbox table by following these steps:

1. Open the Fields tab for the Sandbox table in the Manage Database dialog.
2. Double-click the Example Calculation field or select it and click the Options button.
3. Change the field's formula to GetMonday.
4. Change the field's Calculation result type to date.
5. Click OK to save the calculation formula and close the Specify Calculation dialog.
6. Click OK to save and close the Manage Database dialog.
7. View the result on the Sandbox Form layout for any record.

If the calculation containing this function call were evaluated on January 5, 2017, a Thursday, the result should be the preceding Monday, 1/2/2017.

Adding Parameters to a Custom Function

The `GetMonday` custom function from the previous example is fine as a simple demonstration of how to construct a function without parameters. However, it is somewhat lacking in usefulness due to its limited flexibility. The use of the `Get (CurrentDate)` function does allow *some* variability since it will continue functioning into the future but it excludes the possibility of processing based on a past or future date. Similarly, by hard-coding the result to *always* return a Monday, it excludes the possibility of using the function to calculate *any* of the other six days in a week relative to the date being processed. Both limitations can be overcome with parameters.

■ **Tip** When choosing a name for a parameter, be sure to give it an adequate amount of thought. There is no way to restrict the type of data a calling formula passes into a parameter, so be sure to choose a name that clearly describes the intended data type(s) to avoid developer confusion and errors. For example, a parameter named `input` is too vague so consider using `dateToProcess` instead to clearly state what type of data it is expecting. Look at built-in function parameters for guidance on this topic.

Adding a Date Parameter

To create a new custom function that can calculate the corresponding Monday from *any* date provided by the calling formula, follow these steps to duplicate and modify the previous function example:

1. Open the Manage Custom Functions dialog.
2. Select the `GetMonday` function and click the `Duplicate` button. This will create a new function named `GetMonday Copy`, which should already be selected.
3. Click the `Edit` button to open the `Edit Custom Function` dialog.
4. Change the name of the function to `GetMonday2`.
5. Type the name `dateInput` into the `Function Parameters` text box, then click the `+` button to create the parameter in the list below.
6. Modify the code so that the `dateInput` parameter is used in place of the current date and placed into a renamed variable of `dateToProcess`, which is applied throughout as shown in bold here:

```
Let ( [
    dateToProcess = GetAsDate ( dateInput ) ;
    numAdjustment = DayOfWeek ( dateToProcess ) - 2
] ;
    dateToProcess - numAdjustment
)
```

Once this code is entered, the custom function should appear as shown in Figure 25-4. To save it, click the OK button.

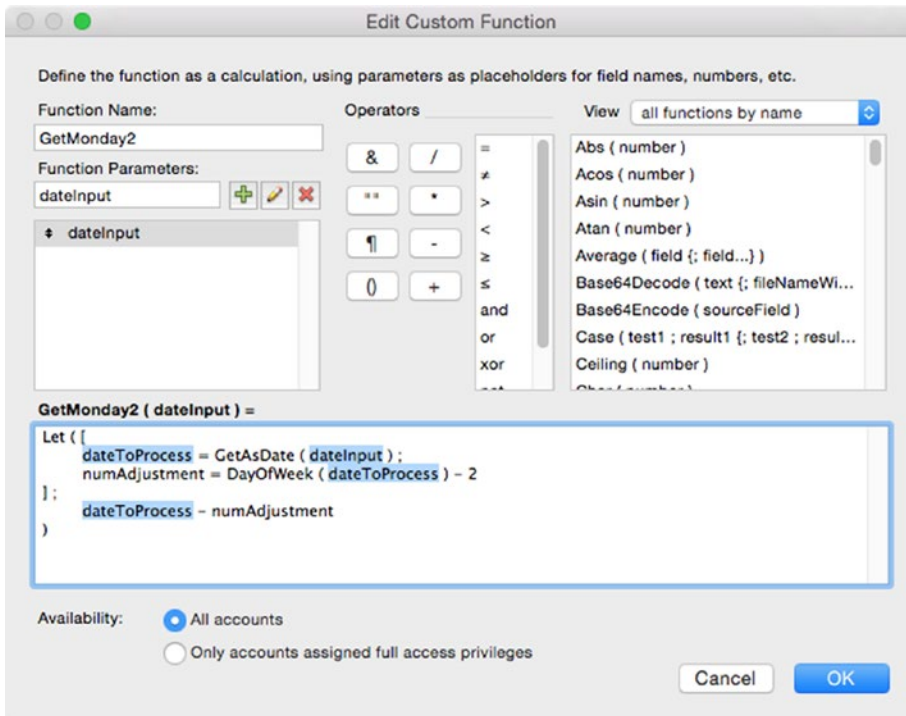


Figure 25-4. The *GetMonday2* function, ready for saving

Now modify the formula in the Example Calculation to refer to the new *GetMonday2* function. Since the function converts the input to a date automatically, the parameter can be any of the following:

- A text field containing a date value — `GetMonday2 (Sandbox::Example Text)`
- A date field — `GetMonday2 (Sandbox::Example Date)`
- A literal text string containing a date — `GetMonday2 ("1/5/2017")`
- A date value built with a function — `GetMonday2 (Date (1 ; 5 ; 2017))`

Depending on the date provided to the function in the new parameter, the result should be the Monday relative to that date.

Adding a Day Requested Parameter

To further expand the function’s flexibility, an additional parameter will allow it to return the date for *any* day of the week relative to the `dateInput` provided. To get started:

1. Create a duplicate of the *GetMonday2* function.
2. Since the new, expanded function will no longer be locked to returning a Monday, the new function’s name should be changed more dramatically; from *GetMonday2* to *GetDay*.
3. Add a new, second parameter named `dayRequested`.

The code will need some changes to accommodate the desired functionality, as shown highlighted here:

```
Let ( [
    dateToProcess = GetAsDate ( dateInput ) ;
    list = "Sunday||Monday||Tuesday||Wednesday||Thursday||Friday||Saturday" ;
    list = Left ( list ; Position ( list ; dayRequested ; 1; 1 ) ) ;
    dayNumber = ValueCount ( list ) ;
    numAdjustment = DayOfWeek ( dateToProcess ) - dayNumber
] ;
    dateToProcess - numAdjustment
)
```

A new `list` variable is first initialized to contain a carriage return-delimited list of weekday names, which will be used to determine where in the week the requested day falls. In the next line, `list` is modified using the `Left` and `Position` functions to include only the days prior to the day name being requested. Finally, the `ValueCount` function is used to determine the number of days remaining in the list, which is placed into the `dayNumber` variable. This is used to determine the `numAdjustment` value in a manner like the other versions of this function.

Once the new function is saved, modify the formula in the `Example Calculation` to refer to the new `GetDay` function. In this example, the result will be the date of Friday for the week that January 5, 2017, falls within:

```
GetDay ( "1/5/2017" ; "Friday" ) // result = 1/6/2017
```

The pattern of progression through these three expanding steps demonstrates the true power of custom functions. They allow the development of *open-ended* formulas that can be used repeatedly with parameter-driven variability.

Adding a Default Current Date

When a parameter requests a value like a date, it is sometimes useful to consider adding a default condition that fills in the parameter automatically when it is not specified by the calling formula. For example, right now, a formula that was requesting the Friday for today would be required to explicitly include that in the call:

```
GetDay ( Get ( CurrentDate ) ; "Friday" )
```

By modifying the first line of the `Let` statement from the last example to include a condition to default to the current date when none is provided, the formula can pass an empty string instead and still receive a result.

Modify the first line with a `Case` statement that uses the current date if `dateInput` is empty or the date provided if `dateInput` has a value, as shown here:

```
dateToProcess = Case ( dateInput = "" ; Get ( CurrentDate ) ; GetAsDate ( dateInput ) ) ;
```

Now, a formula can request a day relative to the current date by passing in an empty string in the first parameter:

```
GetDay ( "" ; "Friday" )
```

Stressing the Importance of Thorough Testing

Any new or modified formula should always be carefully tested before being used and relied on in a live production database. Since custom functions are globally accessible and can be used repeatedly, it is *critical* that they be well tested.

In our simple example, testing a single date and receiving a correct result is not enough. Instead, we must perform as many tests as possible with a variety of different input to confirm that it will handle a full range of possible values.

The testing requirements of each custom function will be different. Start by asking what variety of input might be received. In our example, what if the date provided is at the *start* of the week; a Sunday or Monday? What if the date is at the *end* of the week; a Friday or Saturday? What if the day requested is the same day of the date provided? Will the function work under each of these circumstances? Will it operate correctly when requesting *any* day of the week relative to *any* date? Do all the anticipated data types — date, timestamp, and text string with a date — return an accurate result? To confirm this, we must test each condition and confirm that the formula is working correctly.

To begin, convert the list of questions into a list of test scenarios that will be adequate to confirm the desired functionality with a large sampling of input. In our example, at the minimum, the following tests should be performed:

- `dateInput` — Run seven tests, one test for each day of a week as input. Also, at least one test of each accepted data type: date, timestamp, text string with date, text string with timestamp.
- `dayRequested` — Run seven tests, one test for each day of the week.

This indicates the need for *sixteen* tests on the `GetDay` custom function. Rather than perform these tests one by one, the following single calculation formula, entered in the `Example Calculation` field in the `Sandbox` table that covers them all:

```
Let ( [
    dateInput = Date ( 1 ; 15 ; 2017 )
] ;
    "Input = " & dateInput & " (" & DayName (dateInput) & ")¶" &
    "+1 Sunday=" & GetDay ( dateInput + 1 ; "Sunday" ) & "¶" &
    "+2 Sunday=" & GetDay ( dateInput + 2 ; "Sunday" ) & "¶" &
    "+3 Sunday=" & GetDay ( dateInput + 3 ; "Sunday" ) & "¶" &
    "+4 Sunday=" & GetDay ( dateInput + 4 ; "Sunday" ) & "¶" &
    "+5 Sunday=" & GetDay ( dateInput + 5 ; "Sunday" ) & "¶" &
    "+6 Sunday=" & GetDay ( dateInput + 6 ; "Sunday" ) & "¶" &

    "Sunday=" & GetDay ( dateInput; "Sunday" ) & "¶" &
    "Monday=" & GetDay ( dateInput ; "Monday" ) & "¶" &
    "Tuesday=" & GetDay (dateInput; "Tuesday" ) & "¶" &
    "Wednesday=" & GetDay ( dateInput; "Wednesday" ) & "¶" &
    "Thursday=" & GetDay ( dateInput; "Thursday" ) & "¶" &
    "Friday=" & GetDay ( dateInput; "Friday" ) & "¶" &
    "Saturday=" & GetDay ( dateInput; "Saturday" ) & "¶" &
    "Timestamp=" & GetDay ( GetAsTimestamp ( dateInput ) ; "Sunday" ) & "¶" &
    "Text Date=" & GetDay ( GetAsText ( dateInput ) ; "Sunday" ) & "¶" &
    "Text TS=" & GetDay ( GetAsText ( GetAsTimestamp ( dateInput ) ) ; "Sunday" ) & "¶"
)
```

```
// result =
  Input = 1/15/2017 (Sunday)
+1 Sunday=1/15/2017
+2 Sunday=1/15/2017
+3 Sunday=1/15/2017
+4 Sunday=1/15/2017
+5 Sunday=1/15/2017
+6 Sunday=1/15/2017
Sunday=1/15/2017
Monday=1/16/2017
Tuesday=1/17/2017
Wednesday=1/18/2017
Thursday=1/19/2017
Friday=1/20/2017
Saturday=1/21/2017
Timestamp=1/15/2017
Text Date=1/15/2017
Text TS=1/15/2017
```

The formula uses a `Let` statement to make repeated calls to the `GetDay` function with a variety of different input, placing each result in a variable. It then concatenates those variables into a single string. It is important to change the `Example Calculation` field's `Calculation` result to `Text` for this result for it to be displayed properly in the field.

The first six results show that, as the `dateInput` is incremented to cover an entire week, the resulting value for the requested Sunday is the same. The next seven results show that using the *same* `dateInput` while requesting a *different* day of the week works since the results span a seven-day period. Finally, the last three results show that when the `dateInput` is a *timestamp*, *text-based date*, or *text-based timestamp*, the result is always the same.

With this test completed, it should now be safe to use this custom function with a wide variety of input.

Accessing Fields from Custom Functions

Although the `Edit Custom Function` dialog interface does not include any tools for selecting and inserting fields into a formula, it is possible to include field references. There are two ways to do this:

- Accept a field reference as a parameter
- Type or paste a field reference within the formula

As with any non-field formula, a custom function does *not* have the ability to specify an evaluation context, so care must be taken when using field references, especially when hard-coding them into the formula.

When a formula calls a function that accepts a field reference as a parameter, it is important to be certain that the reference provided is valid within the context at the time the function is called.

Hard-coded field references are almost never a good idea in a custom function unless the function will be limited in use to a single context or is accessing a global field, which works regardless of context.

Building Recursive Custom Functions

In FileMaker, the only formulas that can be recursive are custom functions. A *recursive function* is a function that can call itself from within itself, allowing an overall functional process to be solved in repeated smaller steps. This has the effect of allowing a formula to act as a repeating loop, incrementally stepping through a list of data or performing a sequence actions. Functions that benefit from recursion are those that would require too much code to achieve within a single calculation formula or are impossible to perform without placing some limit on the number of iterations allowed. For example, adding a list of numbers into a total or merging multiple lists of values into a blended table are two examples that would be difficult to perform without recursion.

By its looping nature, as illustrated in Figure 25-5, recursion can be *very* confusing to conceptualize, describe, and use effectively without accidentally ending up in an endless loop and hitting the limits imposed by FileMaker. It is imperative that every recursive function be carefully constructed and that have an exit condition.

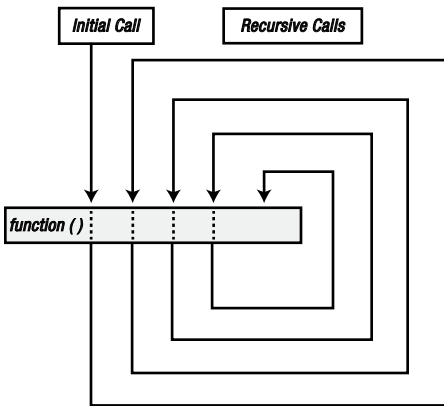


Figure 25-5. An illustration of a function calling itself recursively four times after the initial call from another formula

Building a Simple Recursive Function

A simple example of recursion is a custom function that will add a carriage return-delimited list of numbers as demonstrated by this example named `AddNumbers (numberList)`:

```
Let ( [
    current = GetValue ( numberList ; 1 ) ;
    remaining = RightValues ( numberList ; ValueCount ( numberList ) - 1 )
] ;
    current + Case ( remaining = "" ; 0 ; AddNumbers ( remaining ) )
)
```

In this example, the `Let` statement first splits the input in `numberList` into two variables. The first, named `current`, is initialized to contain the *first* number in the `numberList`. The second, named `remaining`, is initialized to contain any *remaining* numbers after the first one. In the calculation portion of the statement, the number in `current` is added to the result of a `Case` statement, which is either an exit condition returning zero or a recursive call to the function if there are numbers left within the `remaining` variable.

Once created and saved, the function can be called as follows with any quantity of carriage return-delimited numbers and it will return the total of these, as shown in this example:

```
AddNumbers ( "5¶10¶20" ) -- Result = 35
```

Understanding the Limitations to Recursion

Recursion allows a formula to iterate as many times as necessary if it does not violate either of two technical limitations. If either of these two is exceeded, the result of the function will be an error expressed as a question mark. These limits depend on the type of recursion used in the function:

- 50,000 total recursive calls when using properly optimized, tail-recursion.
- 10,000 deep call stack when using head-recursion.

The term *tail-recursion* refers to the placement of a recursive call as the final step at the end of the function's formula so that no further processing is required to complete the current iteration of the function at the point when it calls itself for the next iteration. In other words, when the recursive call returns a result to the current iteration, no unfinished processing tasks are piling up in a pending stack. This is the preferred method and has the larger limit of 50,000 total recursive calls because there is no need to juggle the growing call stack.

By contrast, *head-recursion* is a term that refers to the placement of a recursive call anywhere within the formula. In this case, when the recursive call is made, there are still additional processing steps that must be carried out once a result is returned. FileMaker must queue up all the pending processes in a *call stack* while it waits for the results of every successive recursive call.

The previous section's example of the AddNumbers function might at first seem to be an example of tail-recursion since the recursive call is at the end of the calculation portion of the Let statement. However, it is not. Even though the recursive call is at the end, the value in the current variable must be held in the call stack waiting for the results of the Case statement.

```
current + Case ( remaining = "" ; 0 ; AddNumbers ( remaining ) )
```

Once each iteration is complete, its current value must be held in place until the final number in the list is reached, and then each number is added in reverse as the stack is collapsed back up to produce a result.

So, this example is limited to 10,000 iterations instead of 50,000. While this limit usually far exceeds the needs of most custom functions, it is a good idea to be aware of how to work around it. In this case, the function can be converted to use tail-recursion with a local variable.

Using Local Variables Across Iterations of Recursion

Any formula can initialize local or global variables using a Let statement. Local variables are especially useful with recursive custom functions since they persist across all iterations of the function session. Since they are restricted to the function session and don't persist beyond it, there is no risk of conflict with other processes, unlike with global variables.

Local variables are a safe and effective way to store running totals or results for recursive functions and can even be beneficial in converting a function limited by head-recursion into the more versatile tail-recursion, expanding the maximum iteration limit from 10,000 to 50,000.

With a couple of quick changes, the `AddNumbers` function can be converted to use a local variable named `$Total` to store the running total of values and freeing itself from the head-recursion limitation. These changes are highlighted in this example:

```
Let ( [
    current = GetValue ( numberList ; 1 ) ;
    $Total = $Total + current ;
    remaining = RightValues ( numberList ; ValueCount ( numberList ) - 1 )
] ;
    Case ( remaining = "" ; $Total ; AddNumbers ( remaining ) )
)
```

In each iteration of this revised version, the current number is added to the running total leaving the `Case` statement to either return that total as the result or call the next recursive iteration. Since there is nothing else remaining to be added or processed, FileMaker does not need to add anything to the call stack and can, therefore, handle substantially more recursive loops.

Embedding Test Code Inside a Function

Earlier we discussed writing elaborate test code to confirm that a custom function can handle a wide range of different input. Recursion opens the possibility of storing that test code *inside* of the custom function being tested. This is a rather advanced idea and, if you are new to custom functions, might seem daunting to undertake. However, given the importance of testing after any function modifications made in the future, having test code for a function stored within the function makes future testing an easy proposition.

A `Case` statement can be used to detect special values in one or more parameters indicating a request for test results. In this example, the `dateInput` parameter would be used to determine if the call to the function was asking for test results or normal operations:

```
Case ( dateInput = "Test" ; <<perform test>> ; <<normal operations>> )
```

Combining this statement with the original code and our test code from the previous examples, we can duplicate the `GetDay` function to create a new function called `GetDay2` with the same two parameters but containing the following code:

```
Case ( dateInput = "Test" ;

    Let ( [
        dateInput = Date ( 1 ; 15 ; 2017 )
    ] ;
        "Input = " & dateInput & " (" & DayName (dateInput) & ")¶" &
        "+1 Sunday=" & GetDay ( dateInput + 1 ; "Sunday" ) & "¶" &
        "+2 Sunday=" & GetDay ( dateInput + 2 ; "Sunday" ) & "¶" &
        "+3 Sunday=" & GetDay ( dateInput + 3 ; "Sunday" ) & "¶" &
        "+4 Sunday=" & GetDay ( dateInput + 4 ; "Sunday" ) & "¶" &
        "+5 Sunday=" & GetDay ( dateInput + 5 ; "Sunday" ) & "¶" &
        "+6 Sunday=" & GetDay ( dateInput + 6 ; "Sunday" ) & "¶" &

        "Sunday=" & GetDay ( dateInput; "Sunday" ) & "¶" &
        "Monday=" & GetDay ( dateInput ; "Monday" ) & "¶" &
        "Tuesday=" & GetDay (dateInput; "Tuesday" ) & "¶" &
        "Wednesday=" & GetDay ( dateInput; "Wednesday" ) & "¶" &
```

```

    "Thursday=" & GetDay ( dateInput; "Thursday" ) & "¶" &
    "Friday=" & GetDay ( dateInput; "Friday" ) & "¶" &
    "Saturday=" & GetDay ( dateInput; "Saturday" ) & "¶" &
    "Timestamp=" & GetDay ( GetAsTimestamp ( dateInput ) ; "Sunday" ) & "¶" &
    "Text Date=" & GetDay ( GetAsText ( dateInput ) ; "Sunday" ) & "¶" &
    "Text TS=" & GetDay ( GetAsText ( GetAsTimestamp ( dateInput ) ) ;
    "Sunday" ) & "¶"
)
;
Let ( [
    dateToProcess = GetAsDate ( dateInput ) ;
    list = "Sunday¶Monday¶Tuesday¶Wednesday¶Thursday¶Friday¶Saturday" ;
    list = Left ( list ; Position ( list ; dayRequested ; 1 ; 1 ) ) ;
    dayNumber = ValueCount ( list ) ;
    numAdjustment = DayOfWeek ( dateToProcess ) - dayNumber
] ;
    dateToProcess - numAdjustment
)
)

```

Now, if the `dateInput` receives a value of "Test" it will perform the test routine; otherwise it will assume the parameter contains a date and will perform its normal function.

To test its normal operations, enter the following formula for the Example Calculation field of the Sandbox table and make sure it is set to return a date value:

```
GetDay2 ( "1/5/2017" ; "Friday" ) // result = 1/6/2017
```

To trigger the function's test mode operations, enter the following formula for the Example Calculation field of the Sandbox table and make sure it is set to return a text value:

```
GetDay2 ( "test" ; "" )
```

This should return the same results we received in the previous test example.

■ **Tip** This example illustrates that recursion shouldn't be thought of as only a looping phenomenon but a self-referencing one. It is possible to create recursive functions with multiple modes that self-refer for a variety of different functions.

Summary

In this chapter, we discussed the powerful benefits of developing your own custom functions.



Introducing ExecuteSQL

The *Structured Query Language (SQL)* is a standardized programming language that is used to manage relational databases and perform numerous operational functions to the data they store. SQL was initially created in the 1970s and soon became the standard programming language for relational databases. The American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) adopted an official SQL standard in 1986 and 1987 respectively. Since then, many updates to the standard have been released jointly by both organizations, with the most recent version adopted in December of 2011. Numerous companies now develop proprietary and open source SQL-compliant database systems.

In version 9.0 (2007), FileMaker introduced its first support for a SQL feature with the ability to create live connections to external ODBC data sources (Chapter 7). FileMaker version 12.0 (2012) introduced the ExecuteSQL function, adding the ability to perform SQL queries against FileMaker tables from any calculation formula within a FileMaker database.

A *SQL Query* is a text-based statement used to instruct a database to perform an action. The most frequently used type of query and the only one supported by the ExecuteSQL function in FileMaker is the SELECT query, which contains data retrieval instructions for a desired result set.

Experienced SQL programmers will appreciate the back-end functionality that this command adds to FileMaker but may find the function's limitation to the SELECT command constraining. Newcomers to database programming will find the function a little intimidating and divergent from FileMaker's more intuitive methods of data access. However, strictly from a FileMaker perspective, ExecuteSQL is like a miracle opening a new realm of possibility. The benefits of this function are immeasurable.

In FileMaker version 16, the ExecuteSQL function is matured. It allows a search, sort, and summarization of data to be performed directly from a calculation formula in a completely *context-independent* manner. It can perform a search in any table occurrence, including those that have no layout representation, and can even form temporary relationships dynamically. The functions it makes possible would otherwise require the creation of a multi-line script and extra table occurrences, relationships, and layouts to establish the necessary context to achieve the same result. Using this command, it is possible to severely limit the number of relationships required for data retrieval.

You don't need to learn SQL or use this command to create databases with FileMaker since the function is intended as a complement rather than a replacement to the more intuitive features available. However, once you realize what this function can do and how many resources it makes obsolete, you will want to use it everywhere it proves practical.

In this chapter, we will define and discuss the process of using the ExecuteSQL function, with topics such as:

- Defining the ExecuteSQL function
- Creating SQL queries
- Accessing the database schema

■ **Tip** To experiment with the ExecuteSQL function, be sure to enter some test data into the database. To prepare the Learn FileMaker database for use with the examples in this chapter, import the `us-500.csv` download (Chapter 5) into the Contact table, matching up the name and address fields that we have already created.

Defining the ExecuteSQL Function

The ExecuteSQL function allows *any* calculation formula to retrieve data from *any* table occurrence within the relationship graph *completely independent* of any defined relationship or user interface context. Data can be accessed directly from the data table of any table occurrence without requiring hard-coded relationships, leaving the relationship graph less cluttered and convoluted.

A call to the ExecuteSQL function must include three parameters but can accept one or more optional arguments parameters, as shown in these two examples:

```
ExecuteSQL ( sqlQuery ; fieldSeparator; rowSeparator )  
ExecuteSQL ( sqlQuery ; fieldSeparator; rowSeparator ; arguments )
```

The parameters in the statement are defined as follows:

- `sqlQuery` — a text expression or reference to a field that contains a SELECT statement.
- `fieldSeparator` — a text string containing the character(s) that should be used as a separator between fields in the result. If left empty, a comma will be used as a default separator.
- `rowSeparator` — a text string containing the character(s) that should be used as a separator between records in the result. If left empty, a carriage return will be used as a default separator.
- `arguments` — one or more text values that are used as dynamic parameters in the query, replacing question marks typically in a WHERE clause.

How the ExecuteSQL Function Works

The ExecuteSQL function works by accepting a Select statement that specifies which field(s) and record(s) should be extracted from a table and returned as text. It can optionally include delimiters to be inserted between fields (columns) and records (rows) in the result, as illustrated in Figure 26-1.

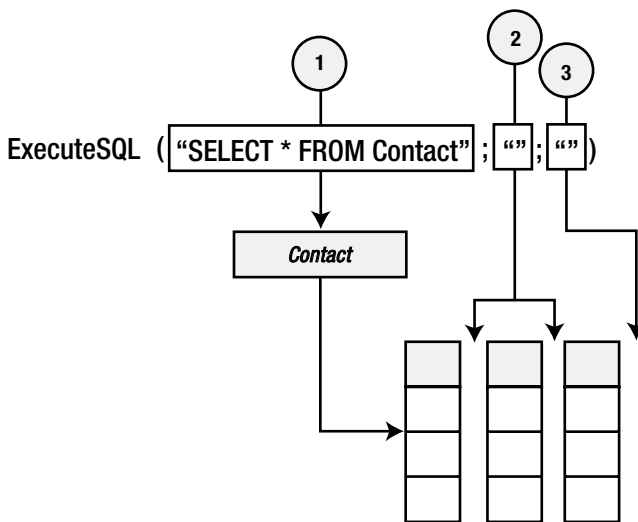


Figure 26-1. An illustration of the basic functioning of the `ExecuteSQL` function, which specifies the data to be retrieved and optional delimiters to be inserted into that data

The following three parameters are required by the function:

1. The `sqlQuery` parameter contains a `SELECT` query statement that informs FileMaker which field(s) to extract, which record(s) to include, and the table from which to extract them. The extracted information is in a temporary abstract columnar format that will be converted into a text result.
2. The optional `fieldSeparator` parameter allows you to define a delimiter, other than the default comma, which will be used between each field value returned when more than one field is requested.
3. The optional `rowSeparator` parameter allows you to define a delimiter other than the default carriage return, which will be used between each record.

The result is returned to the calling formula as a text string. When default delimiters are used, the string will take the form of a list of comma-separated field values in carriage-return separated records.

This example shows the formula pattern for requesting every field from a fictional table using default delimiters:

```
ExecuteSQL ( "Select * FROM Table" "" , "" )
-- Result =
Record1Field1,Record1Field2,Record1Field3,Record1Field4¶
Record2Field1,Record2Field2,Record2Field3,Record2Field4¶
Record3Field1,Record3Field2,Record3Field3,Record3Field4¶
```

This example shows a formula pattern for requesting the same information using an alternative `fieldSeparator`:

```
ExecuteSQL ( "Select * FROM Table" "--", "" )
-- Result =
    Record1Field1--Record1Field2--Record1Field3--Record1Field4¶
    Record2Field1--Record2Field2--Record2Field3--Record2Field4¶
    Record3Field1--Record3Field2--Record3Field3--Record3Field4¶
```

This example shows the formula pattern of a request using both an alternative `fieldSeparator` and `rowSeparator`:

```
ExecuteSQL ( "Select * FROM Table" "--", "#" )
-- Result =
    Record1Field1--Record1Field2--Record1Field3--Record1Field4#Record2Field1--
    Record2Field2--Record2Field3--Record2Field4#Record3Field1--Record3Field2--
    Record3Field3--Record3Field4#
```

Understanding Some Limitations of ExecuteSQL

Before moving on to actual examples, there are some limitations to the `ExecuteSQL` functions that should be understood to avoid frustration. These include:

- As mentioned in the introduction of this chapter, the `ExecuteSQL` function is currently limited to the `SELECT` command only. It is *not compatible* with other standard SQL functions that perform record changes or database schema modification such as `DELETE`, `INSERT`, `UPDATE`, `INSERT INTO`, `CREATE TABLE`, `DELETE TABLE`, etc.
- Relationships created in FileMaker are not recognized or required. However, a `SELECT` statement *can* use a `JOIN` clause to dynamically create temporary relationships for use within the query.
- The function does *not* recognize or require layout context. Instead it allows access to table records *directly* based on a named table occurrence.
- Values *must* be sent in an SQL-92 compliant date and time formats with no braces to be properly recognized. To apply the correct formatting to dates and times in a query, use a `DATE`, `TIME`, or `TIMESTAMP` conditioning operator, preceding the string to have it properly recognized. Failure will cause the value to be evaluated as a literal string rather than a date or time. `ExecuteSQL` will *not* accept the ODBC/JDBC formats for date, time, and timestamp constants contained in braces.
- `ExecuteSQL` will return date, time, and number data using the Unicode/SQL format rather than the date and time settings of the database file or operating system.
- Sorting for SQL in FileMaker uses the Unicode binary sort order.
- `ExecuteSQL` is a *calculation function* and is not the same as the similarly named `Execute SQL script step`.

Creating SQL Queries

At minimum, the `sqlQuery` portion of the `ExecuteSQL` function requires a `SELECT` statement, which has numerous optional clauses available.

Defining SELECT Statements

The `ExecuteSQL` function's `sqlQuery` parameter must be a properly formatted `SELECT` statement. Minimally, it must contain an indication of *what* to find and from *where*, following this pattern:

```
SELECT <what> FROM <where>
```

The `<what>` is the name of one or more fields that should be selected and the `<where>` is the name of the table occurrence from which to extract them. Therefore, the pattern above can be formally expressed as the following formula:

```
SELECT <field(s)> FROM <table>
```

For example, this statement would return the contents of a field named `Name` from every record in a table occurrence named `Contact`:

```
SELECT Name FROM Contact
```

The above formula defines a `SELECT` statement requesting *all records* without further processing. In addition to this basic requirement, the statement can include many optional clauses that expand the functionality of the query and offer refinement of and manipulation to the results. The `SELECT` command makes available features such as:

- `SELECT` — Request one or more fields from one or more tables.
- `SELECT DISTINCT` — Eliminate duplicates from the result.
- `JOIN` — Create temporary relationships.
- `WHERE` — Find sets of records matching specific criteria.
- `AS` — Create dynamic aliases of tables and fields to shorten the length of clauses.

The full formula, including every optional clause except aliasing, is:

```
SELECT/SELECT DISTINCT <fields>
FROM <tables>
JOIN <table> ON <formula>
WHERE <formula>
GROUP BY <fields>
HAVING <formula>
UNION <select>
ORDER BY <fields>
OFFSET <number> ROW/ROWS
FETCH FIRST <number> PERCENT/ROWS/ROW/ONLY/WITH TIES
```

Each available clause is defined in Table 26-1.

Table 26-1. *The definitions of each available clause of a SELECT statement*

Keyword	Clause Description
SELECT	Specifies one or more fields to select. Can include fields, constants, calculations, and functions. Use an asterisk to select all fields.
SELECT DISTINCT	Adding the DISTINCT operator will remove any duplicates from the result.
FROM	Specifies one or more tables from which to select the fields.
JOIN	Contains a table and relationship formula to allow for the extraction of fields through a temporarily established relationship.
WHERE	Contains one or more formulas that specify the criteria that all results must match, like filtering the results based on search criteria.
GROUP BY	Specifies one or more fields to use to summarize the results.
HAVING	Contains one or more formulas that specify the criteria for the inclusion of a summarized result. HAVING is to a GROUP BY what a WHERE is to a SELECT.
UNION	Used to combine two or more SELECT statements into a single result.
ORDER BY	Specifies one or more fields to use to sort the results.
OFFSET	Used to specify the number at which results should begin, thereby excluding a set of records prior to that number.
FETCH FIRST	A number specifying the how many rows that should be retrieved from the starting point.
AS	Used to create a shorter alias for a table name that can be used in its place elsewhere in the statement as a prefix to identify a field's table, especially when there is more than one table involved as when using a JOIN clause.

Each of these operational clauses are woven into a text string with other combinations of literals, keywords, table occurrences, and field names to form a SELECT SQL query, which can be passed as the first parameter to the ExecuteSQL function.

Formatting Requirements

There are a few formatting requirements to keep in mind when writing SELECT statements for the ExecuteSQL function. These include:

- *Command and Object Names* — The names of tables, fields, and even statement commands within a query are *not* case sensitive. For example, using either select or SELECT will work the same. However, using capitalization for all commands and operators can help to differentiate them from the value of specific tables, fields, and other variable information in the query. The examples in this chapter will adhere to that approach.
- *Criteria* — Literal criteria, such as that used within a JOIN, WHERE, and HAVING clause, *is* case sensitive and will fail to locate matching values of a different case. Also, all textual criteria *must* be enclosed in single quotations.

- *Name Separators* — When listing multiple tables and fields, always use a comma-space delimiter between them.
- *Quotations* — Table and field names don't need to be enclosed in double quotations *unless they contain spaces*. Table names that begin with non-alphabetic characters must be enclosed in double quotations even when they don't contain spaces.

Using the SELECT Statement

Although the SELECT statement is the only one supported by the ExecuteSQL function in FileMaker, it is quite powerful and flexible. Before delving into the many different optional clauses, let's explore the basic statement's capabilities and discuss techniques for using it effectively.

Selecting an Entire Table

The most basic SQL query is one in which every field/column will be selected for every record/row. This can be performed with a simple statement:

```
SELECT * FROM <Table>
```

The statement must *always* begin with the word SELECT since that is the only primary function currently supported by FileMaker. After this the statement must indicate which fields to select and FROM which table. In this case, the asterisk informs FileMaker to select *all fields*. The <Table> placeholder is replaced with the name of an actual table occurrence whose base table the function should access. Change this portion of the formula to Contact and the following SELECT statement we fetch every field from the Contact table will read as follows:

```
SELECT * FROM Contact
```

This SELECT statement becomes the first parameter of the ExecuteSQL statement:

```
ExecuteSQL ( "SELECT * FROM Contact" ; "" ; "" )
```

Open the Learn FileMaker database and enter this as the formula for the Example Calculation field. After confirming that the Calculation result type is of Text, save the formula, and the result displayed in Figure 26-2 should appear in the field. Each paragraph of the result is a single record that contains a comma-separated list of field values. Both records and fields are displayed in creation order.

```
000001,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,James,Butt,,6649 N Blue Gum St,New Orleans,70116,Orleans,LA
000002,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Josephine,Darakjy,,4 B Blue Ridge Blvd,Brighton,48116,Livingston,MI
000003,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Art,Venere,,8 W Cerritos Ave #54,Bridgeport,08014,Gloucester,NJ
000004,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Lenna,Paprocki,,639 Main St,Anchorage,99501,Anchorage,AK
000005,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Donette,Foller,,34 Center St,Hamilton,45011,Butler,OH
000006,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Simona,Morasca,,3 Mcauley Dr,Ashland,44805,Ashland,OH
000007,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Mitsue,Tollner,,7 Eads St,Chicago,60632,Cook,IL
000008,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Leota,Dilliard,,7 W Jackson Blvd,San Jose,95111,Santa Clara,CA
000009,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Sage,Wieser,,5 Boston Ave #88,Sioux Falls,57105,Minnehaha,SD
000010,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Kris,Marrier,,228 Runamuck Pl #2808,Baltimore,21224,Baltimore City,MD
000011,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Minna,Amigon,,2371 Jerrold Ave,Kulpsville,19443,Montgomery,PA
000012,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Abel,Maclead,,37275 St Rt 17m M,Middle Island,11953,Suffolk,NY
000013,15:30:14,2016-11-04,2016-11-04,3:30:14 PM,Mark Munro,Mark Munro,,Kiley,Caldarera,,25 E 75th St #69,Los Angeles,90034,Los Angeles,CA
```

Figure 26-2. An example result of the ExecuteSQL function when requesting every field from every record of the Contact table

■ **Caution** Any table names that include spaces or begins with non-alpha characters *must* be enclosed in double quotations to work correctly in an SQL query. Since the `Select` statement is contained in quotes within the `ExecuteSQL`, the inner quotations around the table name must be escaped (backslash-quote) as shown in this example showing a hypothetical table named `Expense Data`:

```
ExecuteSQL ( "SELECT * FROM \"Expense Data\" ; "" ; "" )
```

Selecting Individual Fields

While there may be situations where a formula requires access to *every* field for *every* record in a table, most of the time you will be selecting data with more specificity. Fortunately, the `SELECT` statement allows the selection of one or more individual fields, producing a more focused result.

■ **Caution** Like table names, field names that contain spaces or reserved words must be enclosed in double quotations.

Specifying a Single Field

To select a *single field* from a `Contact` table, change the asterisk to the name of the field. The formula for this would be:

```
SELECT <Field> FROM <Table>
```

To select the `Contact Address City` field from the `Contact` table, write the following formula:

```
ExecuteSQL ( "SELECT \"Contact Address City\" FROM Contact" ; "" ; "" )
```

This statement will result in a carriage return-delimited list of city names from every record in the table, partially shown in Figure 26-3. Notice that the list includes the field value for *every* record, so it will include many duplicates like those highlighted.

New Orleans
 Brighton
 Bridgeport
 Anchorage
 Hamilton
 Ashland
 Chicago
 San Jose
 Sioux Falls
 Baltimore
 Kulpville
 Middle Island
 Los Angeles
 Chagrin Falls
 Laredo
 Phoenix
 Mc Minnville
 Milwaukee
 Taylor
 Rockford
 Aston
 San Jose
 Irving
 Albany
 Middlesex
 Stevens Point
 Shawnee
 Easton
 New York

Figure 26-3. The list of city names selected from the Contact table

Specifying Multiple Fields

To select multiple fields from the Contact table with a single query, list each field in a comma-space separated string. The formula for this would be:

```
SELECT <Field1>, <Field2>, <Field3> FROM <Table>
```

To select the Contact Address City and Contact Address State fields from the Contact table, write the following formula:

```
ExecuteSQL (
  "SELECT \"Contact Address City\", \"Contact Address State\" FROM Contact" ;
  "" ; "" )
```

The result of this statement will be a comma-delimited list of city and state names, partially shown in Figure 26-4.

```

New Orleans,LA
Brighton,MI
Bridgeport,NJ
Anchorage,AK
Hamilton,OH
Ashland,OH
Chicago,IL
San Jose,CA
Sioux Falls,SD
Baltimore,MD
Kulpsville,PA
Middle Island,NY
Los Angeles,CA
Chagrin Falls,OH
Laredo,TX
Phoenix,AZ
Mc Minnville,TN
Milwaukee,WI
Taylor,MI
Rockford,IL
Aston,PA
San Jose,CA
Irving,TX
Albany,NY
Middlesex,NJ
Stevens Point,WI
Shawnee,KS
Easton,MD
New York,NY

```

Figure 26-4. The list of cities and states selected from the Contact table

Keeping References Dynamic

When building a query, it is a good idea to avoid typing the name of tables and fields as static text and instead use dynamic references. Then, if the name of a table or field changes in the future, the query will continue to function without modification since the reference to the field updates automatically.

The following formula demonstrates a simple example of this technique:

```

Let ( [
    reference = GetFieldName ( Contact::Contact Name First ) ;
    reference = Substitute ( reference ; ":" ; "¶" ) ;
    table.name = GetValue ( reference ; 1 ) ;
    field.name = Quote ( GetValue ( reference ; 2 ) )
] ;
    ExecuteSQL ("SELECT " & field.name & " FROM " & table.name ; "" ; "" )
)

```

The Let statement converts the dynamic field reference into a text string and places it into the reference variable. The Substitute function is then used to replace the two colons with a paragraph return, placing the following text value into the reference variable:

```
Contact
Contact Name First
```

Finally, each value is extracted and placed into the table.name and field.name variables respectively. From there, they can be used to construct the SELECT statement that is placed into an ExecuteSQL function.

This extra effort ensures that any changes to the table or field name will automatically be updated and the query will continue to function as expected.

Using SELECT DISTINCT to Return Unique Values

To automatically eliminate duplicate values and alphabetize the results, use the SELECT DISTINCT command, as shown in this formula:

```
SELECT DISTINCT <Field> FROM <Table>
```

The following formula will generate a list of alphabetically sorted, unique values from the Contact Address City field of the Contact table:

```
ExecuteSQL ( "SELECT DISTINCT \"Contact Address City\" FROM Contact" ; "" ; "" )
```

The uniqueness of the list is based on the *entire* record value of the result, not individual fields within it. For example, when selecting only the city, the results will include only one entry for San Jose. However, if multiple fields are selected, like the street address and city, as shown in the example below, the results will include multiple entries for San Jose since the rest of the record/row includes other values.

```
ExecuteSQL (
"SELECT DISTINCT \"Contact Address Street\", \"Contact Address City\" FROM Contact"
; "" ; "" )
```

Since 123 First Street, San Jose is not equal to 555 Selma Ave, San Jose, both will appear in the result.

Reformatting SELECT Statements for Clarity

The preceding examples are intentionally short and easy to read. However, a SELECT statement can and will grow and begin wrapping onto multiple lines due to the following:

- Additional SELECT clauses.
- Dynamic construction of a SELECT clause using FileMaker resources: field references, functions, and variables.
- Dynamic construction of a SELECT clause with variable components.
- Requesting multiple fields in a single clause.

There are two basic methods one can employ to reformat the statements to improve clarity and readability: adding carriage returns and tabs or using a Let statement.

Adding Carriage Returns and Tabs

FileMaker will ignore carriage returns and tabs in the `sqlQuery` text string so these can be used to separate the statement into hierarchically readable blocks. Doing so will break up the continuous text flow and make the statement easier to read and troubleshoot, as shown here:

```
ExecuteSQL ( "
    SELECT <field>
    FROM <table>
    JOIN <table> ON <formula>
    WHERE <formula>
    ORDER BY <field>
" ; "" ; "" )
```

In the formula above, a carriage return is used to separate the actual query from the enclosing `ExecuteSQL` statement. Also, a combination of a carriage return and tab is used to separate each clause of the statement onto its own line, where it is indented to stand out from the enclosing statement.

When multiple fields, find conditions, or other components are used, those can be pushed onto their own line, further indented for additional clarity as shown here:

```
ExecuteSQL ( "
    SELECT
        <field1>,
        <field2>,
        <field3>
    FROM <table>
    JOIN <table> ON <formula>
    WHERE
        <condition1> and
        <condition2> and
        <condition3>
    ORDER BY
        <field1>,
        <field2>
" ; "" ; "" )
```

Using a LET Statement

Another method of eliminating the visual confusion of a complex query is the use of a `LET` statement. The entire `SELECT` query can be built in pieces using separate variables and then combined into a single variable before being inserted into the `ExecuteSQL` statement, as shown here:

```
Let ( [
    s.Fields = "SELECT <Fields>" ;
    s.Table = "FROM <Table> " ;
    s.Join = "JOIN <table> ON <formula> " ;
    s.Where = "WHERE <formula> " ;
    s.Group = "GROUP BY <fields> " ;
```

```

    SQL = s.Fields & s.Table & s.Join & s.Where & s.Group
] ;
ExecuteSQL ( SQL ; "" ; "" )
)

```

Exploring the Benefits of Aliases

In a SQL query, an *alias* is a short text string made up of one or more characters that can act as a proxy for a table name elsewhere in a SELECT statement. When a SELECT statement contains references to more than one table, as in a JOIN clause, aliases are used to identify the table to which a field belongs.

Although an alias can be made up of one or more characters, as a space-saving mechanism, shorter is better. To establish an alias, use the AS clause after the identification of a table and follow it with a text alias, as shown here:

```
SELECT <field> FROM <table> AS <alias>
```

For example, to create an alias named *c* for the Contact table, the FROM clause would appear like this:

```
FROM Contact AS c
```

Once the alias *c* is established, it can be used as a prefix on any field name, in any clause, to identify the table to which a field belongs. While aliases are not required when selecting fields from a single table, this example illustrates that it is possible.

```
SELECT c.Notes FROM Contact AS c
```

■ **Note** Although the AS clause establishes the alias at the end, it can be used *anywhere* within the statement.

Again, in such a simple statement, using an alias is unnecessary as illustrated with these three examples that all generate the exact same result:

```

SELECT Notes FROM Contact
SELECT Contact.Notes FROM Contact
SELECT c.Notes FROM Contact AS c

```

When a field with spaces in its name is enclosed in double quotations, the alias prefix should precede the name, outside of the quotes, as shown here:

```
SELECT c.\"Contact First Name\" FROM Contact AS c
```

Inserting Literal Text in the Field List

Literal text strings can be inserted before, between, or after field names within the SELECT statement and will be repeated in the results for each record. Literals must be enclosed in single quotation marks anywhere they are used.

■ **Note** Literals are also used in clauses that include formulas such as `WHERE` and `JOIN`, which are discussed later in this chapter.

This example demonstrates a literal value inserted into the field results as a field label:

```
ExecuteSQL ( "
    SELECT
        'Name: ',
        \"Contact Name First\",
        \"Contact Name Last\"
    FROM Contact\"
; " " ; "" )
```

By placing a literal `'Name: '` with the field names and using a `fieldDelimiter` of a space in the second parameter of the `ExecuteSQL` function, this example will return a list of names with a preceding label:

```
Name: Josephine Darakjy
Name: Art Venere
Name: Shannon Miller
```

Concatenating Results

Concatenation is the action of linking things together in a chain or series and is possible within SQL queries. Instead of receiving a raw comma-delimited set of field values and then parsing and manipulating that information with other formulas or scripts, concatenation allows pre-processing of field values into more useful results *directly within a query statement*.

Concatenation of text values is achieved using either the `+` or `||` operators. However, the latter has been reportedly more reliable and may be less likely to be confused with the same operator used in mathematical calculations. The code below shows an example of concatenation of a contact's first and last names into a single string, with a literal space inserted between them. This is achieved by replacing the comma-space between the fields with a space, double-bar, space, single quote, space, single quote, space, double-bar, space, as shown here:

```
ExecuteSQL ( "
    SELECT \"Contact Name First\" || ' ' || \"Contact Name Last\"
    FROM Contact
; " ; "" ; "" )
```

Using the alternative plus-sign delimiter, the `SELECT` line of the above example, would be:

```
SELECT \"Contact Name First\" + ' ' + \"Contact Name Last\"
```

In both examples, each record of the result should contain the first and last name of a contact with a space between them. For example, with concatenation, the examples would return a result of `Art Venere` instead of `Art,Venere`, without it. While the same result could have been achieved using the `fieldDelimiter` parameter, by concatenating the fields directly in the statement, that delimiter remains available for use between other fields that may be included in the query but not concatenated within the same statement.

This example shows how to return a contact's full name and entire mailing address as a continuous concatenated string:

```
ExecuteSQL ( "
    SELECT
        \"Contact Name First\" + ' ' +
        \"Contact Name Last\" + ' ' +
        \"Contact Address Street\" + ' ' +
        \"Contact Address City\" + ', ' +
        \"Contact Address State\" + ' ' +
        \"Contact Address Zip\"
    FROM Contact
" ; "" ; "" )
```

Using the WHERE Clause

The WHERE command can be added to a SELECT statement to specify criteria for the records that should be included in the result, as shown here:

```
SELECT <field> FROM <table> WHERE <formula>
```

The <formula> portion of the clause must contain one or more expressions that specify the criteria used to qualify records that should be returned. These will typically include a field, operator, and either a literal value or another field.

Creating a Single Expression WHERE Clause

To select Contact records with a California address, the expression of the formula would be:

```
"Contact Address State" = 'CA'
```

■ **Note** Just like when using fields in the select list, fields in the WHERE clause must be enclosed in double quotations if they contain spaces. Also, literal textual values must always be enclosed in single quotations.

The following shows an example of a SELECT statement with this expression in a WHERE clause to extract the first and last name of every contact with an address in California from a Contact table:

```
SELECT
    "Contact Name First",
    "Contact Name Last"
FROM Contact
WHERE "Contact Address State" = 'CA'
```

Creating a Multiple Expression WHERE Clause

When using multiple expressions, they must be separated by a comparison operator of AND or OR.

For example, when searching for contacts living in a city that is common to many states, such as Milford, you might need two expressions to specify both the city *and* the state. In that case, you would use the AND operator between the two expressions, as shown in this example:

```
WHERE "Contact Address City" = 'Milford' AND "Contact Address State" = 'PA'
```

Similarly, to find contacts from two different states, for example, from Pennsylvania *or* Ohio, you would use an OR operator, as shown in this example:

```
WHERE "Contact Address State" = 'PA' OR "Contact Address State" = 'OH'
```

Using the ORDER BY Clause

The ORDER BY clause can be added to a SELECT statement to specify the field(s) by which the results should be sorted, as shown in this formula:

```
SELECT <field> FROM <table> ORDER BY <fields>
```

For example, to return a list of every contact's last name sorted by state:

```
SELECT "Contact Name Last" FROM Contact ORDER BY "Contact Address State"
```

Combining the ORDER BY with a WHERE clause, this example will return the last name of every contact living in a city named "Milford" sorted by state:

```
SELECT "Contact Name Last"  
FROM Contact WHERE "Contact Address City" = 'Milford'  
ORDER BY "Contact Address State"
```

Using the JOIN Clause

A JOIN clause can be added to a SELECT statement to create a temporary relationship between two table occurrences. Joining tables is useful when selecting fields from both a local and related table, blending them into a single result. Joins allow other clauses with field references to use fields from either or both tables. For example, a WHERE or ORDER BY clause can each contain fields from either table, once joined.

The JOIN clause must contain the name of a table that should be related to the FROM table and include a formula containing one or more expressions that specify the criteria used to connect records into a temporary relationship, as shown in this formula:

```
SELECT <field> FROM <table1> JOIN <table2> ON <formula>
```

To connect the Contact and Company tables in the Learn FileMaker database, assign them an alias of con and com respectively and add a JOIN clause. To retrieve the Contact Name First field and the related Company Name field, use the following query:

```
SELECT
    con."Contact Name First",
    com."Company Name"
FROM Contact AS con
JOIN Company AS com ON
    con."Contact Company ID" = com."Record ID"
```

The example above declares aliases in the SELECT and JOIN statements and then uses them throughout the statement.

■ **Tip** The JOIN clause can also be used to create self-joins, which is a table related to itself.

Using the GROUP BY Clause

The GROUP BY clause can be added to a SELECT statement to generate an aggregate value based on one or more fields, as shown in this formula:

```
SELECT <fields> FROM <table1> GROUP BY <field>
```

The clause acts much like a FileMaker-native summary field when placed in a sub-summary part on a layout displaying records sorted by the appropriate field. It generates a summarization of data based on a sort/grouping field.

For example, if the Contact table had a field called Invoices that contained a total of a customer's invoices, the following code *without* a GROUP BY clause will return a list of each *contact's* State and their Invoice amount:

```
SELECT State, Invoices FROM Contact ORDER BY State
-- Result =
AK,1000
AK,500
AK,250
AZ,500
AZ,750
Etc.
```

By contrast, using the Sum function on the Invoices field and adding a GROUP BY clause to group by state, this example will return a summary of invoices by state:

```
SELECT State, Sum ( Invoices ) FROM Contact GROUP BY State
-- Result =
AK,1750
AZ,1250
Etc.
```

Notice that the `ORDER BY` clause is removed from the example above since the `GROUP BY` clause will automatically sort the results by the grouping field(s).

Adding a HAVING Clause

The `HAVING` clause can be added to a `GROUP BY` clause of a `SELECT` statement to filter the results based on a formula:

```
SELECT <fields> FROM <table> GROUP BY <field> HAVING <formula>
```

The clause acts like a `WHERE` clause but for grouped results. Using the previous example, we can add a `HAVING` clause to only include states where the summary of Invoices is greater than a certain dollar amount, as shown here:

```
SELECT State, Sum ( Invoices ) FROM Contact GROUP BY State HAVING Sum ( Invoices ) > 1500
-- Result =
AK,1750
CA,2500
Etc.
```

In the results, the summarized entry for AZ has been removed because the summary of 1250 was under the threshold of 1500 that is specified in the `HAVING` clause.

Using the UNION Clause

The `UNION` clause will combine the results of two or more `SELECT` statements, whether from the same table or from different tables, if two conditions are met:

- Each `SELECT` statement must select the same number of fields.
- Each field position must return the same data format across all `SELECT` statements.

For example, if the first `SELECT` statement returns three fields with the data types of text, number, and text, the second `SELECT` statement must also return three fields with the same data types in the same order. The formula for the `UNION` clause is:

```
SELECT <fields1> FROM <table1> UNION SELECT <fields2> FROM <table2>
```

By default, the clause will automatically exclude duplicate entries from the merged result. Use `UNION ALL` to include all results, even duplicates, for example:

```
SELECT <fields1> FROM <table1> UNION ALL SELECT <fields2> FROM <table2>
```

Limiting the Results of a Query

The `OFFSET` and `FETCH FIRST` clauses can be used separately or in unison to control the number of results returned by a query.

Using the OFFSET Clause

The `OFFSET` clause can be added to a `SELECT` statement to indicate the number of records/rows in the result to skip over, as shown in this formula:

```
SELECT <fields> FROM <table> OFFSET n ROWS
```

This example will return a list of all the `Record ID` field values from the `Contact` table, starting from record 21:

```
SELECT \"Record ID\" FROM Contact OFFSET 20 ROWS
```

Using the FETCH FIRST Clause

The `FETCH FIRST` clause can be added to a `SELECT` statement to limit the number of rows returned, as shown in this formula:

```
SELECT <fields> FROM <table> FETCH FIRST n ROWS ONLY
```

This example will return the `Record ID` for the first 10 records:

```
SELECT \"Record ID\" FROM Contact FETCH FIRST 10 ROWS ONLY
```

Combining the OFFSET and FETCH FIRST Clauses

When combined, the `OFFSET` and `FETCH FIRST` clauses allow you to access groups of results one batch after the other, a process often referred to as “paging” through results, since each group is presented as the next “page” in a list of results.

```
SELECT <fields> FROM <table> OFFSET n ROWS FETCH FIRST n ROWS ONLY
```

The `OFFSET` portion indicates where the desired group starts and the `FETCH FIRST` portion limits the number of records accessed from that starting point. This code shows several examples of accessing batches of records ten at a time:

```
SELECT \"Record ID\" FROM Contact FETCH FIRST 10 ROWS ONLY
SELECT \"Record ID\" FROM Contact OFFSET 10 ROWS FETCH FIRST 10 ROWS ONLY
SELECT \"Record ID\" FROM Contact OFFSET 20 ROWS FETCH FIRST 10 ROWS ONLY
SELECT \"Record ID\" FROM Contact OFFSET 30 ROWS FETCH FIRST 10 ROWS ONLY
SELECT \"Record ID\" FROM Contact OFFSET 40 ROWS FETCH FIRST 10 ROWS ONLY
```

The first statement will return records 1 through 10. The second statement will return records 11 thru 20, the third will return records 21 thru 30 and so on.

Accessing the Database Schema

One of the lesser known uses of the ExecuteSQL function is the ability to access two “virtual tables,” hidden in every FileMaker database that provide information about the scheme. These are:

- FileMaker_Tables
- FileMaker_Fields

These tables can be used in a SELECT statement just like a custom table.

Selecting FileMaker_Tables

The FileMaker_Tables table contains one record for every table occurrence defined in the relationship graph with the following fields of information:

- TableName — The name of the table occurrence.
- TableID — FileMaker’s identification number for the table occurrence is unique within the database.
- BaseTableName — The name of the actual table upon which the table occurrence is based.
- BaseFileName — The name of the file in which the table occurrence’s base table exists.
- ModCount — The number of modifications made to the table.

Each of these fields can be used in a SELECT statement just like a custom field from your tables.

To get all five fields for every table in the database, use the asterisk, as shown here:

```
ExecuteSQL ( "SELECT * FROM FileMaker_Tables" ; "" ; "" )
-- Result =
Company,1065101,Company,Learn FileMaker,20
Company | Contact,1065105,Contact,Learn FileMaker,24
Contact,1065102,Contact,Learn FileMaker,24
Contact | Company,1065106,Company,Learn FileMaker,20
Project,1065103,Project,Learn FileMaker,4
Project | Company,1065107,Company,Learn FileMaker,20
Sandbox,1065089,Sandbox,Learn FileMaker,134
```

To get a list of every table occurrence in the database, specify the TableName field only:

```
ExecuteSQL ( "SELECT TableName FROM FileMaker_Tables" ; "" ; "" )
-- Result =
Company
Company | Contact
Contact
Contact | Company
Project
Project | Company
Sandbox
```

To limit the results to the actual table names, use `SELECT DISTINCT` and request the `BaseTableName`:

```
ExecuteSQL ( "SELECT DISTINCT BaseTableName FROM FileMaker_Tables" ; "" ; "" )
-- Result =
Company
Contact
Project
Sandboxe
```

■ **Tip** All the clauses and options available in the `SELECT` statement can be used when accessing the hidden `FileMaker_Tables` and `FileMaker_Fields` tables.

Selecting FileMaker_Fields

The `FileMaker_Fields` table contains one record for every field defined in the database file with the following fields of information, all accessible through the `ExecuteSQL` function:

- `TableName` — The name of the table occurrence whose base table contains the field exists.
- `FieldName` — The name of the field.
- `FieldType` — The SQL data type of the file.
- `FieldID` — FileMaker's identification number for the field is unique within the base table containing the field.
- `FieldClass` — The class of the field: `Normal`, `Summary`, or `Calculated`.
- `FieldReps` — The number of maximum repetitions defined for the field.
- `ModCount` — The number of modifications made to the table.

Each of these fields can be used in a `SELECT` statement just like any custom field from your tables. To get all seven fields for every field in the database, use the asterisk, as shown here:

```
ExecuteSQL ( "SELECT * FROM FileMaker_Fields" ; "" ; "" )
```

Since the hidden fields table contains fields based on table occurrences, the results will include duplicates if there is more than one occurrence for a given base table. A custom script or a recursive custom function like the one below named `Unique_FileMaker_Fields` can be used to get a list of the name of every base table and then retrieve the fields for each, resulting in a clean list of base tables and their fields without any duplicates:

```
Let ( [
    current.table = GetValue ( baseTables ; 1 ) ;
    baseTables = RightValues ( baseTables ; ValueCount ( baseTables ) - 1 )
] ;
    ExecuteSQL (
        "SELECT * FROM FileMaker_Fields WHERE TableName='" & current.table & "'"
        ; "" ; "" ) &
    Case ( baseTables ≠ "" ; Unique_FileMaker_Fields ( baseTables ) )
)
```

Exploring Other SQL Features

FileMaker's SQL functionality is a lot more feature rich than the material covered in this chapter. For example:

- Numerous functions can be embedded into a SELECT statement to manipulate the results, many providing functionality like FileMaker's own functions. These include functions for:
 - Date and time manipulation
 - String manipulation
 - Aggregating numeric values
 - Performing mathematical computations
 - Taking conditional action
- Numerous operators can be used with field values and SQL functions to manipulate results within the SELECT statement.
- The Execute SQL script step allows more robust manipulation of external ODBC/JDBC data sources from external databases.
- FileMaker can be used as an ODBC/JDBC data source and supports SQL queries from external databases.

For more information about these topics, visit filemaker.com and look for these and other guides: The *FileMaker ODBC and JDBC Guide* is available at:

https://fmhelp.filemaker.com/docs/16/en/fm16_odbc_jdbc_guide.pdf.

The *FileMaker SQL Reference* is available at:

https://fmhelp.filemaker.com/docs/16/en/fm16_sql_reference.pdf.

Summary

In this chapter, we introduced the ExecuteSQL function and discussed many features of the SELECT statement.

PART IV



Designing Interfaces

The user interface of a database is composed of one or more layouts with fields and other objects that allow a user to interact with the underlying data structure. These chapters cover the basics of constructing user interfaces:

27. *Introducing Layout Mode* — Provides a general orientation to Layout mode.
28. *Getting Started with Layouts* — Discusses how to plan, create, and manage layouts.
29. *Exploring Layout Palettes* — Explores the tools available on the Field Picker, Inspector, and Layout Objects palettes.
30. *Creating Layout Objects* — Covers the creation of layout objects.
31. *Manipulating Objects* — Discusses the placement and organization of layout objects.
32. *Using Themes and Styles* — Introduces the use of themes and styles to improve the management of the formatting of layout objects.
33. *Customizing Menus* — Explains how to create custom menus to take total control over the user interface.

CHAPTER 27



Introducing Layout Mode

A *database layout* is a developer-designed graphical template that transforms the raw data stored in fields, records, and tables into an interface: an interactive visual presentation. Layouts provide users and scripts with access to record data from the perspective of a specific table occurrence, which is called a context. A database file's layouts are limited only by disk space, up to the maximum overall file size of 8 terabytes.

In this chapter, we will introduce layout mode, discussing:

- Emphasizing the importance of layout design
- Using layout mode

■ **Caution** Many new tools and concepts about layout mode will be *briefly* introduced in this chapter and then explored in more detail in subsequent chapters.

Emphasizing the Importance of Layout Design

Every interaction of a user and stored data will occur on a layout. These include actions such as:

- Viewing records.
- Performing data entry tasks.
- Searching and navigating records.
- Viewing visualizations of complex data in various forms, including text reports or graphs.
- Previewing and printing forms, reports, and labels.

Layout design is crucially important. The entire experience a user has with the stored data is *through* a layout. Except for menus and toolbars, every representation of the data they see, and every button they click is displayed on a layout. The appearance and functionality of a layout's design will greatly influence each user's evaluation of the database and of *you* as its developer.

Layouts can range in complexity depending on functional requirements budget restrictions, user needs, and the skills of the developer building them. Often a simple, spreadsheet-like table view, constructed informally, provides just the ad hoc simplicity that adequately serves a need. Typically, a more elaborate design is appropriate and even expected by skilled users.

Poorly designed layouts can be a cluttered, visual nightmare that confuses the user, frustrates the workflow, and causes more problems than they are intended to solve. By contrast, a well-designed layout can be an artistically expressed robustly featured, efficiently visualized graphical masterpiece that anticipates a user's needs and extends powerful, time-saving tools for manipulating and repurposing data in a convenient and intuitive way.

As you begin working with layouts, remember to keep design in mind.

■ **Caution** When learning the design tools or working on a proof of concept for a new feature, design can be temporarily disregarded. Also, the layout examples shown in this book are not created with design in mind, only functionality presented for demonstration purposes. Don't assume they are intended as examples of good design.

Using Layout Mode

Layout mode is a window state that transforms the content area of a window *from* a rendered graphical user interface *into* a structural interface editing environment. In this mode, a developer can add, configure, and style objects that display data and provide controls for users to interact with when they are viewing it in Browse mode.

Layout mode can be initiated using any one of the following ways:

- Select the View > Layout Mode menu.
- Click the Edit Layout button in the status toolbar.
- Run a script that uses the Enter Layout Mode script step.

Identifying Layout Mode

In Layout mode, the appearance of the window and application menus will closely resemble those in Browse mode with a few important differences. These are especially striking when comparing a window rendered in Browse mode to the same window in Layout mode with optional rulers, grids, guides, and palettes visible, as shown side by side in Figure 27-1.

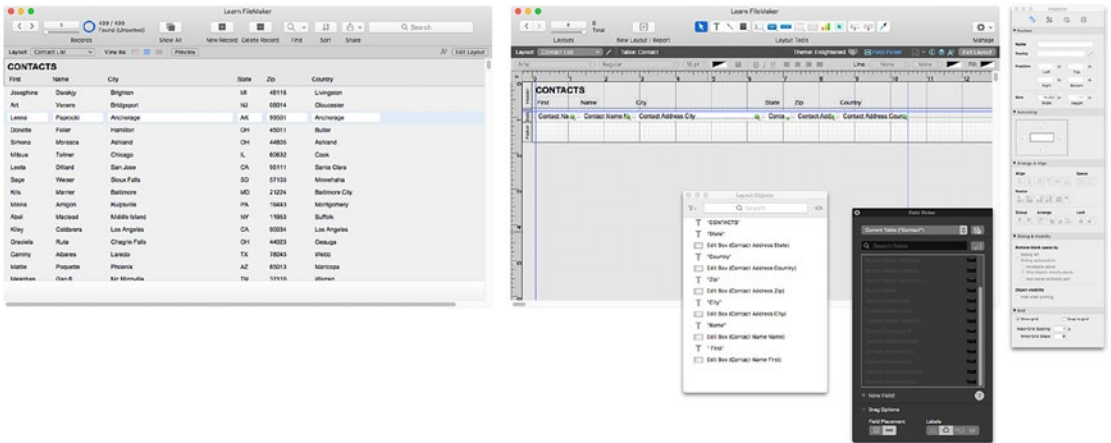


Figure 27-1. The differences between Browse mode (left) and Layout mode (right), with optional rulers, grids, guides, and three palettes currently visible

The most notable differences in Layout mode include:

- **Status Toolbar** — The status toolbar will transform to a completely different set of options (discussed next in this chapter).
- **Menus** — Several menu changes occur in Layout mode (discussed later in this chapter).
- **Design Area** — The *content* area of the window becomes a *design* area, showing all the layout elements arranged as they were in Browse mode but transformed into editable objects. Separate regions of the content area, called parts, will become visible and editable (discussed later in this chapter). Fields no longer display record data or allow data entry but can be moved, resized, and otherwise manipulated (Chapter 31).
- **Rulers, Guides, and Grids** — Depending on View settings, several design tools and overlays that are used to position objects precisely might be visible (Chapter 31):
 - The layout ruler may be seen running along the left and top of the content area.
 - Orange guides may overlay dimension on the layout, demarcating the area that will fit on certain device screens or monitor sizes.
 - Blue guides may be visible on the layout.
 - A grid pattern may be superimposed over the background of the content area.
- **Palettes** — The Field Picker, Layout Inspector, and Layout Objects palettes may be visible if they were left open during the layout editing session. These can be opened and closed as needed in layout mode (Chapter 29).

Status Toolbar (Layout Mode)

When a window is placed into Layout mode the options available in the status toolbar will change dramatically. Controls for managing records and performing data entry tasks will be replaced with tools that add objects to the layout and perform other design-related functions. These controls will be the default Layout mode items or the user’s customized set.

Default Status Toolbar Items (Layout Mode)

The default status toolbar for Layout mode is shown in Figure 27-2.

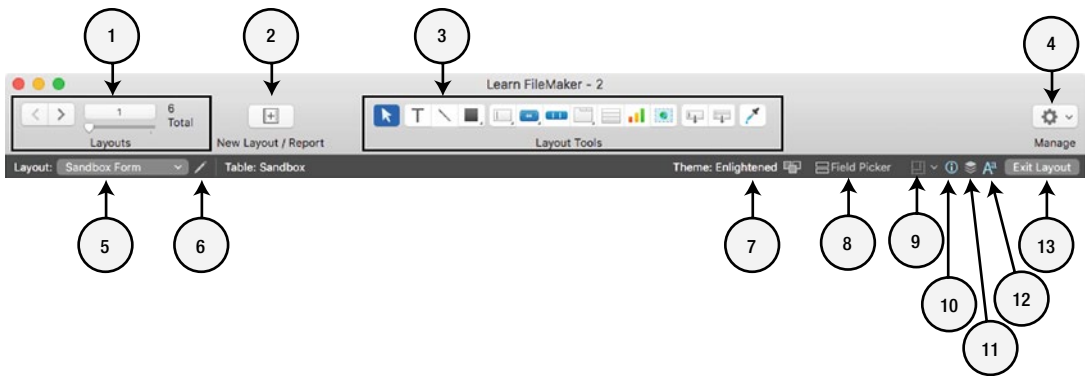


Figure 27-2. The default status toolbar for Layout mode

The toolbar can be divided into the following regions, discussed further in this section:

1. Navigation controls
2. New Layout / Report button
3. Layout tools
4. Manage Database menu
5. Layout navigation menu
6. Layout Settings button
7. Theme selector
8. Field Picker toggle button
9. Screen dimension overlay menu
10. Layout Inspector palette toggle button
11. Layout Objects palette toggle button
12. Formatting Bar toggle button
13. Exit Layout button

Navigation Controls

At the top left of the toolbar are navigation controls that appear and function exactly as they do in Browse mode with one important difference: they refer to and control *layouts* rather than *records*. The total refers to the total number of layouts that exist in the current file and the number in the text box indicates which layout is being currently viewed.

To move to a different layout:

- Click the Next or Back arrows.
- Drag the slider to the left or right.
- Type the number of the desired layout in the box above the slider and type the Enter key.

New Layout / Report Button

The only function button included in the default Layout mode toolbar is:




- **New Layout / Report** — This button begins the process of creating a new layout by opening a dialog of options.

Other function buttons, such as **Delete Layout** and **Duplicate Layout**, which are accessible in the menu by default, can be added by customizing the status toolbar on a local user's computer.

Layout Tools

Running along the center of the toolbar are a row of **Layout Tools**, defined in Table 27-1, that are used to insert layout objects. Some of these can be clicked and dragged downward to place a specific object type onto the layout. Others are clicked to temporarily change the cursor to either perform a new function or allow a corresponding object type to be *drawn* in the layout design area.

Table 27-1. Each layout tool defined

Icon	Tool Description
	<i>Selection tool</i> — This tool is used to select objects on the layout, move, or manipulate them.
	<i>Text tool</i> — This tool is used to edit text on a layout or within certain object types such as buttons.
	<i>Line tool</i> — This tool is used to draw a line on a layout. Once selected, click and hold the cursor at a point on the layout and then drag in the direction of the line. When the line has reached the desired length, release the cursor. To maintain a straight line, hold the Shift key while dragging horizontally or vertically.

(continued)

Table 27-1. (continued)

Icon	Tool Description
	<p><i>Shape menu</i> — Click this icon to automatically select the Rectangle shape tool. Click and hold to reveal the hidden menu with the following three options:</p> <ul style="list-style-type: none"> • The Rectangle shape tool is used to draw a rectangle. • The Rounded Rectangle shape tool is used to draw a rectangle with a default Corner radius setting of 5%. • The Oval shape tool is used to draw an ellipse. <p>To draw a shape maintaining a uniform height and width (a square or circle), hold down the Option key while dragging out its boundaries.</p>
	<p><i>Field menu</i> — This icon hides a menu that allows the quick insertion of a field onto a layout with a choice of a Control style. Click and hold on the icon to see the menu.</p>
	<p><i>Button menu</i> — Click this icon to automatically select the Button tool. Click and hold to reveal the hidden menu with the following two options:</p> <ul style="list-style-type: none"> • The Button tool is used to draw a push-style button. • The Popover Button tool is used to draw a button that, when clicked, will pop open an attached content area that floats over the layout.
	<p><i>Button Bar tool</i> — This tool is used to draw a segmented bar that can be configured with multiple attached buttons and/or popover button controls.</p>
	<p><i>Multi-Panel Object menu</i> — Click this icon to automatically select the Tab Control tool. Click and hold to reveal the hidden menu with the following two options:</p> <ul style="list-style-type: none"> • The Tab Control tool is used to draw an object made up of one or more content areas identified and accessible through tabs. • The Slide Control tool is used to draw an object made up of one or more content areas represented by a dot icon at the bottom and accessible by clicking a dot or swiping left or right on an iOS device.
	<p><i>Portal tool</i> — This tool is used to draw a portal.</p>
	<p><i>Chart tool</i> — This tool is used to draw a chart.</p>
	<p><i>Web Viewer tool</i> — This tool is used to draw a web viewer object.</p>
	<p><i>Field tool</i> — This tool is used to drag a new field down onto a layout.</p>
	<p><i>Part tool</i> — This tool is used to drag a new layout part onto a layout.</p>
	<p><i>Format Painter tool</i> — This tool is used to copy formatting from one object to another. With an object selected, click this tool to copy its formatting into memory. The cursor will change to a combination selection tool and eyedropper, indicating that the next object you click on will be reformatted with the settings previously copied.</p>

Locking a Layout Tool Selection

Tools that operate as toggle buttons, being actively selected or not, can be locked as active by double-clicking on them. This allows you to quickly create multiple objects of the same type, one after the next, without having to reselect the tool. When finished, the tool can be unlocked by clicking on a different tool.

Manage Database Menu

On the far right of the top of the Layout mode toolbar is a Manage Database menu. Clicking the icon reveals a pop-up menu containing the developer options that are also accessible through the File > Manage menu (Chapter 2).

Layout Menu

The lower, non-customizable level of the toolbar starts with the Layout menu on the left. This menu is used to select the layout one wishes to view in the window, which can also be achieved by selecting a layout from the View > Go To Layout submenu.

Unlike the Browse mode equivalent of this menu, which displays *only* those layouts that have the Include in Layouts Menu option selected, in Layout mode this menu displays a list of *all* layouts in the file. It includes a Manage Layouts option at the top that will open a Manage Layouts dialog.

Layout Settings Button

The *layout settings* button opens the Layout Setup dialog where various layout options and behaviors can be specified for the current layout.

Theme Selector

The *layout theme selector* displays the name of the theme assigned to the layout with a button that will open a dialog to select a different theme. A *layout theme* is a collection of stylistic settings that can be applied to layout parts and objects (Chapter 32).

Field Picker Toggle Button

The *field picker toggle button* is used to toggle the visibility of the Field Picker palette. As a visual indicator, the color of the button text and icon changes blue when the palette is visible (Chapter 29).

Screen Stencil Menu

The *screen stencil menu* is a combination of a small dual-box icon and arrow that provide dual functionality. Clicking the arrow to the right of the icon will reveal a menu of screen dimension overlays. A selection of one or more options from the list will draw an orange border outlining the dimensions of a device or monitor screen. These guides are useful when designing layouts optimized for distinct screen sizes, especially those on mobile devices. Clicking the box portion of the icon will toggle the visibility of all the overlay boxes for any currently selected dimensions.

Inspector Toggle Button

The *Inspector toggle button* is a small icon that, when clicked, will toggle the visibility of the Layout Inspector, which is a palette of tools used to edit the appearance, behavior, and content of layout objects (Chapter 29).

Layout Objects Toggle Button

The *Layout Objects toggle button* is a small icon that, when clicked, will toggle the visibility of the Layout Objects palette, which contains tools used to select objects, manipulate their stacking order and more (Chapter 29).

Formatting Bar Button

The *formatting bar button* will toggle the visibility of a bar between the status toolbar and the design area of the window that contains text-formatting controls (Chapter 3, “Formatting Bar”).

Exit Layout Button

The Exit Layout button will switch the window back to Browse mode with a preference-optional dialog asking if you want to save changes (Chapter 2, “Layout Settings”).

Customizing the Status Toolbar (Layout Mode)

The status toolbar in Layout mode is customizable at the user-computer level exactly as it is in Browse mode (Chapter 3, “Customizing the Status Toolbar”). The only difference is that the button functions available are layout specific.

To begin customizing the status toolbar, enter Layout mode and then select the View > Customize Toolbar menu to open the customization panel attached to the window as shown in Figure 27-3.

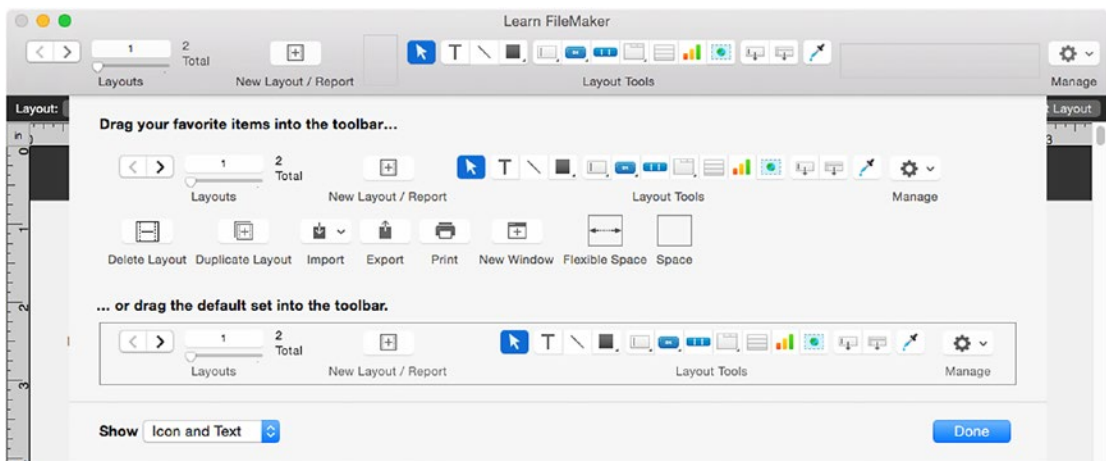


Figure 27-3. The customize toolbar panel attached to the Learn FileMaker database window in Layout mode

Once open, items can be added, removed, or rearranged in the same manner as in Browse mode.

Menu Changes (Layout Mode)

In Layout mode, several changes occur in the menu bar. In addition to the Records menu being completely removed, the following menus are changed or added:

- Edit
- View
- Insert
- Format
- Layouts (New)
- Arrange (New)

Edit Menu

There are many changes to the items in the Edit menu when switching from Browse to Layout mode, as shown in Figure 27-4.

Browse Mode	Layout Mode
Can't Undo ⌘Z	Undo New ⌘Z
Can't Redo ⇧⌘Z	Redo New ⇧⌘Z
Cut ⌘X	Cut ⌘X
Copy ⌘C	Copy ⌘C
Paste ⌘V	Paste ⌘V
Paste Text Only ⇧⇧⌘V	Paste Text Only ⇧⇧⌘V
Clear	Clear
Select All ⌘A	Copy Object Style ⇧⌘C
Find/Replace ▶	Paste Object Style ⇧⌘V
Spelling ▶	Revert Changes to Style
Export Field Contents...	Duplicate ⌘D
	Select All ⌘A
	Find/Replace ▶
	Spelling ▶

Figure 27-4. A comparison of the Edit menu in Browse and Layout modes

The differences to the Edit menu in Layout mode include:

- Copy Object Style — This new item allows the style information of a selected layout object to be copied to the clipboard.
- Paste Object Style — This new item allows style information held in the clipboard to be applied to a selected layout object.

- Revert Changes to Style — This new item will revert the style of the selected object to the settings prior to pasting style information.
- Duplicate — This new item will create a duplicate of any selected layout objects.
- Export Field Contents — This function is not applicable in Layout mode and is removed.

View Menu

There are many changes to the items in the View menu when switching from Browse to Layout mode, as shown in Figure 27-5.

Browse Mode	Layout Mode
<ul style="list-style-type: none"> ✓ Browse Mode ⌘B Find Mode ⌘F Layout Mode ⌘L Preview Mode ⌘U 	<ul style="list-style-type: none"> Browse Mode ⌘B Find Mode ⌘F ✓ Layout Mode ⌘L Preview Mode ⌘U
Go to Layout ▶	Go to Layout ▶
<ul style="list-style-type: none"> ✓ View as Form View as List View as Table 	<ul style="list-style-type: none"> Page Margins Page Breaks
<ul style="list-style-type: none"> ✓ Status Toolbar ⌘⌥S Customize Status Toolbar... Formatting Bar 	<ul style="list-style-type: none"> ✓ Rulers ⌘⌥⇧R Grid ▶ Guides ▶ ✓ Dynamic Guides ⌘⌥'
<ul style="list-style-type: none"> ✓ Ruler 	<ul style="list-style-type: none"> Show ▶
<ul style="list-style-type: none"> Zoom In Zoom Out 	<ul style="list-style-type: none"> ✓ Status Toolbar ⌘⌥S Customize Status Toolbar... ✓ Formatting Bar Field Picker ⌘K
	<ul style="list-style-type: none"> ✓ Inspector ⌘I New Inspector Accessibility Inspector Layout Objects ⌘⌥T
	<ul style="list-style-type: none"> Actual Size Zoom In ⌘⌥+ Zoom Out ⌘⌥-

Figure 27-5. A comparison of the View menu in Browse and Layout modes

The differences to the View menu in Layout mode include:














- Go to Layout — Like the Layout menu in the lower portion of the toolbar, this submenu displays *all* layouts when in Layout mode. Also, a new Manage Layout menu item is added that opens a dialog used to managing layouts.
- View as — The three View as menu items are not applicable in Layout mode and are removed.

- **Ruler / Rulers**— The only difference here is one of quantity. In Layout mode, the menu is pluralized and indicates that it will toggle the visibility of both horizontal and vertical rulers. In Browse mode, the menu is singular and indicates that it will toggle the visibility of a horizontal ruler.
- **Page Margins** — When selected, this menu item transforms the layout's background by superimposing a graphical representation of the page borders that will be imposed by the current print settings. Unlike preview mode, which displays record and field data as a print preview, this feature shows a pseudo-preview overlay while still in Layout mode. This allows adjustment to design elements from the context of a printed page.
- **Page Breaks**— When selected, this menu item superimposes dashed lines on the background of the layout demarcating page breaks based on the current settings. When the **Page Margins** menu is on, these page breaks only show horizontal divisions between pages within the actual print area. However, when that option is off, page breaks will run both horizontally and vertically, breaking the entire layout area into a grid of page-sized portions.
- **Grid** — This menu item contains a submenu of two choices that control the visibility and magnetic quality of a grid of major and minor lines, reminiscent of graph paper, which are intended to assist in more precise alignment of objects:
 - **Show Grid** — This item toggles the visibility of grid lines on layouts in the current database file.
 - **Snap to Grid** — This item toggles the magnetic property of the grid lines. When activated, objects dragged on a layout will be pulled to the next available grid line, helping ease the burden of precision alignment.
- **Guides** — This menu item contains a submenu of two choices that control the visibility of guide lines:
 - **Show Guides** — This item toggles the visibility of guide on layouts in the current database file.
 - **Snap to Guides** — This item toggles the magnetic property of guides. When activated, objects dragged on a layout will be attracted to the next available guide, helping ease the burden of precision alignment.
- **Dynamic Guides** — When selected, this menu item controls the visibility of automatic guides that are used to help position items with uniform precision. When an item is dragged around a layout, dynamic guides will appear around and between objects showing distance and spacing in relation to other objects on the layout. Developers with experience using modern interface design tools such as Apple's Xcode will recognize and appreciate this feature.

■ **Note** See a more detailed discussion of Rulers, Grids, Guides and Dynamic Guides in Chapter 31, “Layout Positioning Helpers.”

- **Show** — This submenu contains a list of options for how objects are displayed when in Layout mode including:
 - Instead of fields displaying their name in Layout mode, there is an option to display Sample Data instead.
 - There are options to show Text Boundaries and Field Boundaries, making visible an object’s dimension regardless of styled appearance.
 - The remaining options all toggle the visibility of a small icon called an *object badge* superimposed over objects that indicates the use of the feature corresponding to the menu, each defined in Table 27-2.

Table 27-2. A list of every layout badge that can be superimposed on a layout object to indicate its use of a feature

Icon	Description
	The object is formatted as a button.
	The object has conditional formatting features applied.
	The object will not be visible when printing.
	The object has placeholder text applied.
	The object is a popover button.
	The object has a formula that may hide the object under certain conditions.
	The object is searchable with Quick Find (Chapter 4, “Searching with Quick Find”).
	The object is searchable with Quick Find but will be slower due to indexing or other considerations.
	The object has Quick Find disabled.
	The object or layout has one or more script triggers assigned.
	The object will slide left when printing.
	The object will slide up when printing.
	The object has a tooltip text assignment.

- **Field Picker** — This item will toggle the visibility of the Field Picker.
- **Inspector** — This item will toggle the visibility of the Layout Inspector, which is used to configure layout objects.
- **New Inspector** — This item will create a new instance of the Inspector. FileMaker allows multiple inspectors to be visible at once, each with a different tab selected or in different places around your computer screen(s).
- **Accessibility Inspector** — This new item opens a small window into which allows Title and Help Text to be assigned to a field label.
- **Layout Objects** — This new item opens the Layout Objects palette for selecting, layering, and hiding objects.

■ **Note** See a more detailed discussion of the Field Picker, Inspector, and Layout Objects palette in Chapter 29.

Insert Menu

There are many changes to the items in the Insert menu when switching from Browse to Layout mode, as shown in Figure 27-6.

Browse Mode	Layout Mode
Picture...	Field...
Audio/Video...	Part...
PDF...	Graphic Object ▶
File...	Field/Control ▶
Current Date ⌘-	Portal...
Current Time ⌘;	Tab Control...
Current User Name ⇧⌘N	Slide Control
From Index... ⌘I	Web Viewer...
From Last and Advance ⇧⌘'	Chart...
	Button
	Popover Button
	Button Bar
	Picture...
	Current Date ⌘-
	Current Time ⌘;
	Current User Name ⇧⌘N
	Date Symbol
	Time Symbol
	User Name Symbol
	Page Number Symbol
	Record Number Symbol
	Other Symbol...
	Merge Field... ⌘⌘M
	Merge Variable

Figure 27-6. A comparison of the Insert menu in Browse and Layout modes

The first two sections of the Layout mode version of the Insert menu provide an alternate method of inserting any type of layout object, essentially mirroring the toolbar icon features.

In the middle are several shortcuts for placing static text directly on a layout in a manner like inserting the same into a field:

- Picture
- Current Date
- Current Time
- Current User Name

Below these are options that provide a shortcut for the insertion of various dynamic placeholder symbols. These are specially formatted text that is automatically replaced with current values at the time a layout is rendered in other non-layout modes. These include:

- **Date Symbol** — This item inserts the text symbol for the display of the current date; `{{CurrentDate}}`.
- **Time Symbol** — This item inserts the text symbol for the current date; `{{CurrentTime}}`.
- **User Name Symbol** — This item inserts the text symbol in the current user’s computer name; `{{UserName}}`.
- **Page Number Symbol** — This item inserts the text symbol for the current page number when a page is printed or previewed; `{{PageNumber}}`.
- **Record Number Symbol** — This item inserts the text symbol for the current record’s number within the found set; `{{RecordNumber}}`.
- **Other Symbol** — This item opens a dialog that allows the selection of any of over a hundred different symbols, each corresponding to a Get function (Chapter 22).

While the symbols above allow dynamic insertion of numerous standard values, it is also possible to place a merge field or merge variable into layout text. A *merge value* is a text string enclosed between a set of left shift (<<) and right shift (>>) symbols that indicates to FileMaker that it is a placeholder that should be dynamically replaced with the actual contents of a field or variable when a layout is rendered in a mode other than Layout.

These features are at the bottom of the Insert menu:

- **Merge Field** — This item opens a Specify Field dialog and places the selected local or related field on the layout, formatted as a *merge field*; `<<FieldName>>` or `<<TableName::FieldName>>`.
- **Merge Variable** — This item inserts two dollar symbols enclosed in merge brackets `<<$$>>` into which you can type the name of a global variable. For example, a global variable named `$$CurrentSeason` would be enclosed in symbols like a merge field to form the string: `<<$$CurrentSeason>>`.

Once a standard symbol or merge value is present on the layout, it can be manipulated and formatted like any other text or combined with static text in a single string. Any symbol or merge value will be detected and replaced with dynamic values when FileMaker renders the layout for Browse, Find, or Preview mode.

■ **Note** Dynamic placeholder symbols, merge fields, and merge variables are also discussed in Chapter 30, “Working with Text.”

Format Menu

There are many changes to the items in the Format menu when switching from Browse to Layout mode, as shown in Figure 27-7. In Layout mode, items in this menu are enabled or disabled depending on the type of object(s) currently selected on the layout.

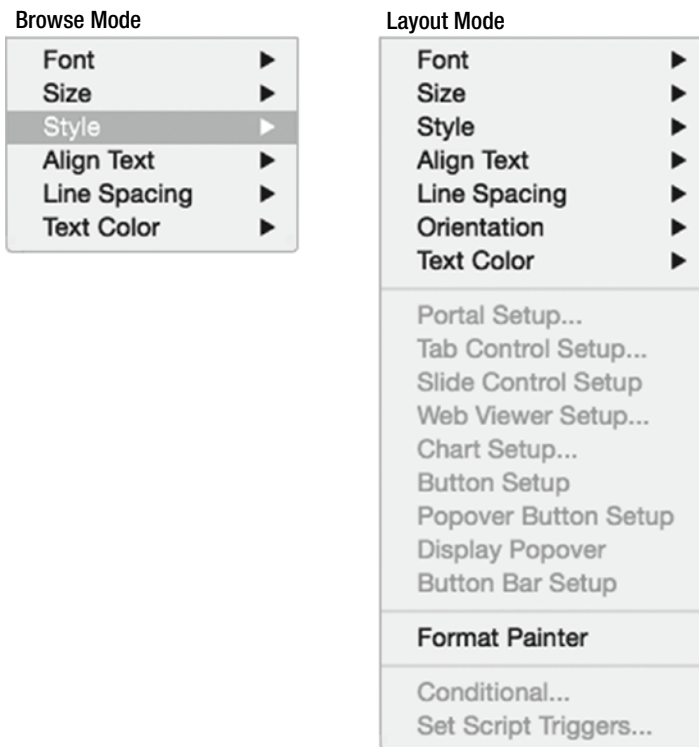


Figure 27-7. *The Format menu in Browse and Layout modes*

The differences to the Insert menu in Layout mode include:

- **Orientation** — This new submenu has two options; Horizontal, which is the default; and Sideways (Asian text only).
- **Setup Options** — The central section of the menu contains a list of Setup menu items that are enabled when an object of the corresponding type is selected. Selecting one of these will open the appropriate setup dialog based on the object type one is configuring (Chapter 30).
- **Format Painter** — This new menu item performs the same function as the Format Painter tool in the toolbar: copying format settings of a selected object that can then be applied to any other object with a click.
- **Conditional** — This new menu item opens the Conditional Formatting dialog, in which contextual formatting settings are applied based on a calculation formula (Chapter 31).
- **Set Script Triggers** — This new menu item opens the Set Script Triggers dialog in which scripts can be assigned to run when certain events are performed to an object (Chapter 37).

Layouts Menu

The *Layouts menu*, shown in Figure 27-8, provides access to functions related to managing, designing, and configuring layouts.

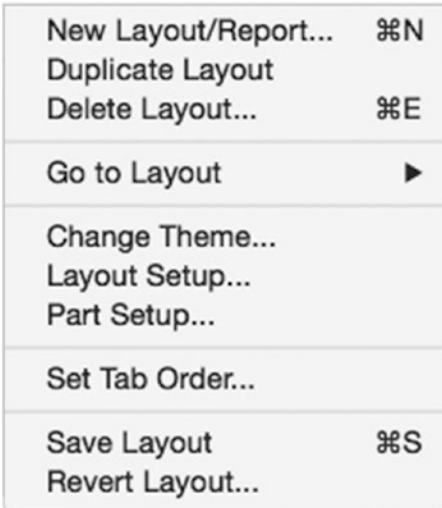


Figure 27-8. The *Layouts menu*

- **New Layout/Report** — Starts the process of creating a new layout.
- **Duplicate Layout** — Creates a duplicate of the current layout.
- **Delete Layout** — Deletes the current layout after displaying a warning dialog.

■ **Note** See a more detailed discussion of creating and deleting layouts in Chapter 28.

- **Go To Layout** — Displays a submenu of options to allow a layout designer to go to the next or previous layout or to enter a Layout number to quickly navigate within the list of layouts.
- **Change Theme** — Opens the Change Theme dialog allowing the assignment of a different theme to the current layout (Chapter 32).
- **Layout Setup** — Opens the Layout Setup dialog in which layout properties and behaviors are configured (Chapter 28).
- **Part Setup** — Opens the Part Definition dialog in which layout part properties and behaviors are configured (Chapter 28).
- **Set Tab Order** — Opens the Set Tab Order dialog, which allows configuration of the order of objects a user will jump to when typing Tab in Browse mode (Chapter 31).

- **Save Layout** — Saves any accumulated unsaved layout changes while remaining in Layout mode.
- **Revert Layout** — After a confirmation dialog, this item discards any unsaved changes and reverts the layout to its previously saved state while remaining in Layout mode.

Arrange Menu

The *Arrange menu*, shown in Figure 27-9, provides access to functions used when moving or layering objects on layouts, most of which are described in Chapter 31.

Group	⌘R
Ungroup	⇧⌘R
Lock	⌘L
Unlock	⇧⌘L
Bring to Front	⌘[
Bring Forward	⇧⌘[
Send to Back	⌘]
Send Backward	⇧⌘]
Rotate	⌘R
Align	▶
Distribute	▶
Resize To	▶

Figure 27-9. *The Arrange menu*

- **Group** — This item groups all the currently selected objects into a single object. Grouped objects can be selected and moved while remaining in positions relative to each other. A selection can include groups, which means you can create nested groups.
- **Ungroup** — This item ungroups the currently selected object(s), one level back toward their singular status.
- **Lock** — This item will lock the selected object(s) positions and settings. The object can't be modified or moved until it is unlocked.
- **Unlock** — This item will unlock the selected object(s) for editing.
- **Bring to Front** — This item will move the selected object(s) to the top of the stack of objects in front of any other objects.
- **Bring Forward** — This item will move the selected object(s) forward one step in the stack of objects.
- **Send to Back** — This item will move the selected object(s) to the bottom of the stack, behind of any other objects.

- **Send Backward** — This item will move the selected object(s) backward one step in the stack of objects.
- **Rotate** — This item will rotate the selected object(s) 90 degrees (clockwise).
- **Align** — This submenu contains commands that align selected objects horizontally (Left Edges, Center, Right Edges) or vertically (Top Edges, Middle, Bottom Edges).
- **Distribute** — This submenu contains commands to evenly distribute selected objects Horizontally or Vertically.
- **Resize To** — This submenu contains commands to resize selected objects relative to the smallest or largest of the group by height, width, or both.

Summary

In this chapter, we introduced Layout mode and identified changes to the window, toolbar, and menu.

CHAPTER 28



Getting Started with Layouts

A new database file is created with a *single* empty layout representing the default table. As a developer, you are responsible to define the fields and relationships that make up the data structure. Similarly, you are responsible for the user interface and must add, rename, design, delete, and organize layouts.

In this chapter, we explore the following subjects:

- Planning layouts
- Working with layout parts
- Adding layouts
- Deleting layouts
- Using the manage layouts dialog
- Optimizing layout performance

Planning Layouts

Before building layouts, take a moment to review the classification methods available for categorizing layouts and to sketch out a layout navigation flow.

Classifying Layouts into Type Categories

When planning an interface, it is important to consider the different types of layouts required to fulfill the needs of users and automated functions. Identifying similarities in design and function between different layouts will help establish unifying standards in how you refer to and design layouts for different tables.

While the categorization may vary from one project to the next based on a variety of unique circumstances, there are many kinds of layouts that can be found in most databases. For example, *every* table that requires user interaction typically has at least one layout displaying multiple records in a list view and another layout showing one record at a time for data entry tasks as a form view. Similarly, most databases will require one or more layouts with groups of buttons to allow for the main menu of navigation, a choice of available actions or quick selection from a group of properties, or other resources.

Before delving further, however, let's take a moment to clarify a point of confusion about layout types in FileMaker. Strictly speaking, *there are no official layout types in FileMaker*. Fundamentally, there is only a single type of layout: an empty space onto which you can add parts and define objects for any function.

However, two areas of focus in the application development interface use terms that can be easily blurred into a mistaken assumption that there *are* official layout types:

- Content view
- Layout creation options

Let’s take a moment to clarify these before focusing on the more important *functional* categories of layouts that are common to most databases.

Refuting Content View Options as a Layout Type

Although the names used to identify window content views are like those used to describe the structure and function of a layout, these are really different ways of *viewing* a layout and do not refer to any real *structural* categorization criteria. When enabled in the Layout Setup Dialog (described later in this chapter), a user can apply them to a layout *regardless* of its design structure or intended function.

To illustrate this point, consider that content views each control two aspects of the way a layout is viewed:

- The number of records displayed at once in a scrollable window (one or many).
- The rendered appearance of the layout as designed or not.

These two properties overlap between the three view options, as shown in Figure 28-1.

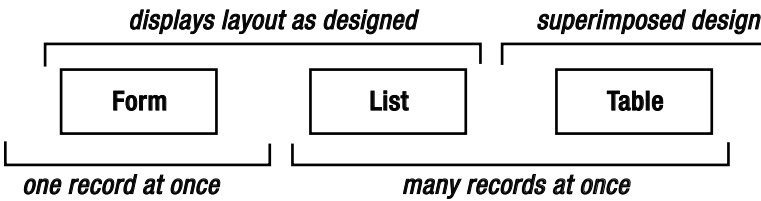


Figure 28-1. The different perspectives of the available content view options

Form and List views are *similar* in that they both always display the layout as designed and are *different* only in how many records are visible at a time.

List and Table views are *similar* in that they both display a list of the records in the found set but *different* in that one displays the layout as designed while the other imposes a predetermined display format that supersedes the layout design.

The similarity in terminology here, especially for List and Form, blur the line between function and content view and are easily confused or informally construed to be the layout’s type. However, from the perspective of the classification of the *layout* instead of how it is *viewed*, it’s generally best to ignore the content view options.

The only relevant consideration is that a layout’s design or function will usually imply a specific view and might influence which views you enable for user selection. If a layout is structured to serve the function of displaying a list of records a user can view it as a single record Form instead, if that view option is enabled. Similarly, if a layout is structured to serve the function of displaying a single record at a time as a form, a user can view it as a List instead, if that view option is enabled. The layout design doesn’t change when a different view is selected, only its appearance as viewed in the window.

Refuting Layout Creation Options as a Layout Type

As we will see later in this chapter, when creating a new layout, FileMaker offers a choice of six default configuration options: Form, List, Table, Report, Label, Envelope. The first three blur the distinction between structure and view by using identical terminology while the last three introduce a new dilemma, since they each are just a more structurally evolved version of a List view.

These classifications are important when creating a record if you remember that you are choosing between different sets of *defaults properties* for the new layout, and select the one that most closely represents the intended function of the layout. Although we will refer to these layout types later, the choice is between sets of helpful options in the layout creation wizard for the default configuration of a new layout. They do not necessarily indicate or dictate the functional intent of the layout.

Classifying Properly by Primary Functional Intent

When designing a layout, the intended *function* should be the primary point of focus in classifying the layout as a certain type. Here the type may be a summarization of the fundamental purpose, device type, record structure, shape, appearance, and purpose of the layout. The chart illustrated in Figure 28-2 shows twelve different commonly used type categories for layouts, each organized hierarchically in relationship to each other and to various other criteria.

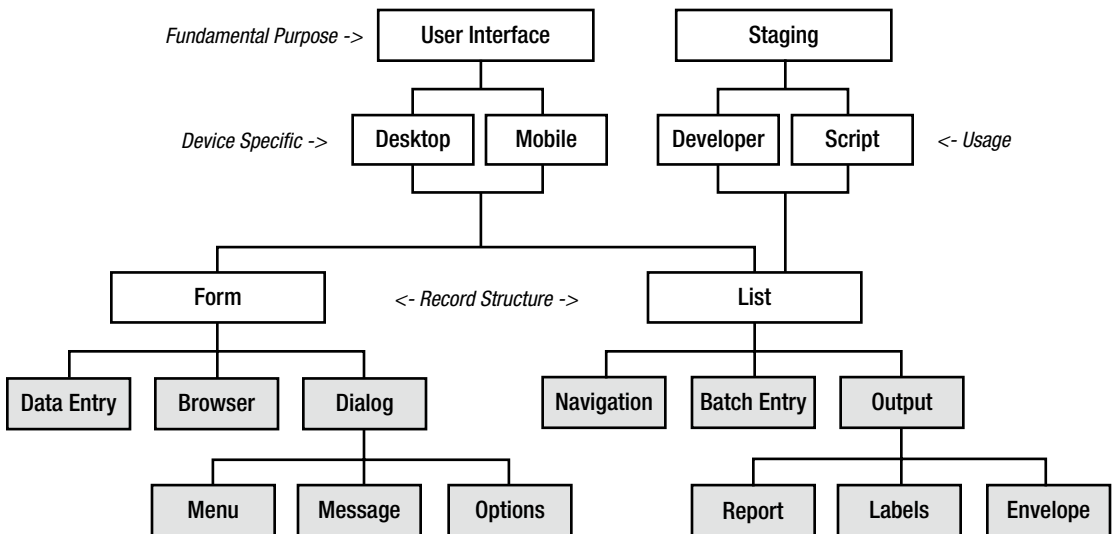


Figure 28-2. An organization of common layout types into a hierarchy based on various factors

This chart has a couple of levels of hierarchy at the top to help illustrate some broader categories for a layout's purpose and usage. These aren't critically important but help to establish some of the different wider groupings into which a layout may fall.

For example, a layout can typically serve one of two different *fundamental purposes*. It may be used as either:

- A *user interface* access point where humans view and manage data. These may be designed for a specific type of device, as needed.
- A *hidden staging* context point that provides access for developers to test and troubleshoot or for scripts to perform automatic functions from a specific context.

Finally, a layout is usually designed to display either a single record form or a multi-record list. As mentioned earlier, these may imply a view choice but they aren't necessarily synonymous.

Identifying Form Layouts

A *form layout* displays a single record with lots of fields and controls arranged and optimized to accommodate a singular purpose. These allow users to focus on a single record or set of options without distraction.

Some examples of *form layouts* include:

- A *data entry* layout is used to display a single record, containing many fields into which a user can enter, view, edit, and otherwise manage record information.
- A *browser* layout is used to display a single record that acts as an overview of one or more different tables, summarizing information and possibly listing various related records in portals. For example, a browser may act as a user control panel on which a user sees related task assignments and other records that require their attention, all summarized into a single view.
- A *dialog* layout is a layout that is used in conjunction with the New Window script step to open a window as a dialog or palette for any number of purposes including:
 - A *menu* dialog contains a set of buttons used to a navigate to layouts and windows within the database solution.
 - A *message* dialog presents information to a user such as warnings or advice, which are often used in place of the Show Custom Dialog script step when fields or other layout objects are required.
 - An *options* dialog presents a set of buttons or other interface elements that act like a palette of optional functions.

Identifying List Layouts

A *list layout* displays a record with a small subset of fields, usually arranged in a single horizontal row that is viewed with other records so a user can quickly scroll through a found set. A list view is typically constructed with at least three parts: a Header, Body, and Footer. The Header contains labels acting as field/column identifiers, running horizontally, each above a corresponding field contained in the Body. Lists may have other parts for navigation and summarizing data as needed.

The fields included in a list view can vary, but should at least include key fields with identifying information to allow a user to quickly know what entity the record represents. For example, a Contact table's list layout should contain the name of the person represented while a Project table's layout should contain the company and project name.

Generally, lists should have a minimum of one button to provide a convenient single click to navigate to a Form view for any record displayed. Numerous other fields and controls can be added as needed for a specific list.

Some examples of different functional types of list layouts include:

- A *navigation* layout is used to present the minimal information required for a user to identify records in which they can scroll through a found set of records and click to navigate to a form layout for data entry. Search scripts often will present the resulting found set on a list layout.
- A *batch entry* layout allows a user to perform some specialized data entry task on a found set of records.
- An *output* layout is used by scripts to generate material that will be sent to a printer, saved as a PDF, exported, or emailed, such as:
 - A *report* layout contains parts and fields specifically configured to present and summarize information in a logical manner.
 - A *label* layout contains parts and fields specially configured to print onto a specific label product.
 - An *envelope* layout contains parts and fields configured to print onto a specific envelope size.

■ **Tip** Most references to a layout type in this book will be focused on the *function* of the layout unless specifically referring to a FileMaker feature or settings in a dialog.

The common layout types described above can be combined with other specific interface needs demanded by a project to compile a custom list of layout requirements.

Sketching a Layout Flow Chart

Once you have a list of the types of layouts required to cover the needs of every table and function, consider creating a navigational flow chart showing how the user will traverse the interface. It's not a requirement, but it can be helpful to visualize the total interface in a single view. Even a rough sketch of layout planning can be helpful, like the one shown in Figure 28-3.

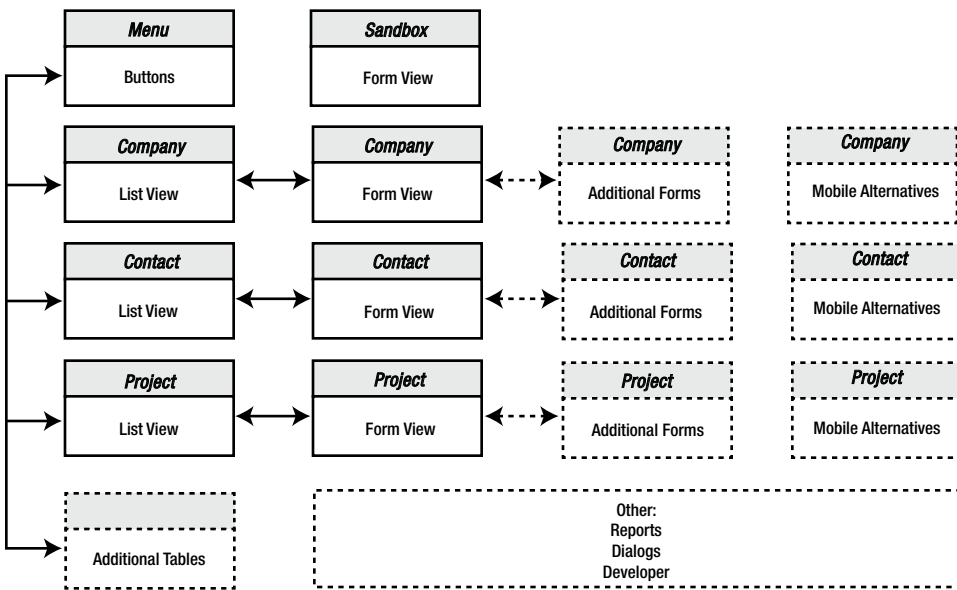


Figure 28-3. A rough sketch of a rather simple hypothetical layout flow with optional placeholders denoting areas for expansion

In this diagram, we have a rough representation of the Learn FileMaker database with an added menu layout and placeholders for hypothetical other layouts. Remember, the navigational arrows are only illustrating hypothetical connections that may be required. These may vary in complexity depending on the number of tables, the style of the navigational controls you develop, and other factors.

Working with Layout Parts

A *layout part* is a horizontal section of a layout that determines how the components contained within the area will behave when rendered in a non-Layout mode. Depending on creation method and selected configuration options, each newly created layout will have at least three default parts: Header, Body, and Footer. Every layout must have at least *one* part but can be made up of as many parts as necessary for your design.

Exploring Layout Regions and Controls

There are several layout regions and controls pertaining to parts, highlighted in Figure 28-4. These include visible layout parts, part boundaries, part labels, and a part label orientation toggle button.

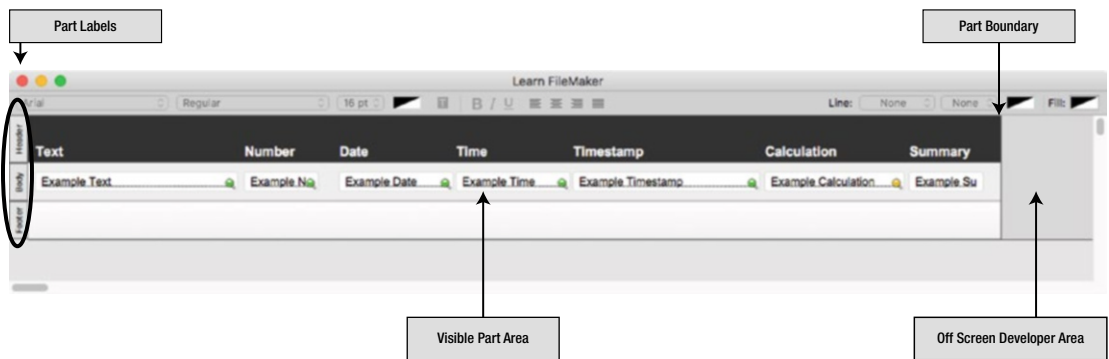


Figure 28-4. A typical layout with minimum default layout parts, highlighting various components

Defining the Part Label

A *part label* is a small box attached to the left side of the part. These contain the part type name and can act as a button, providing access to three functions:

- Double-click on the label to open the Part Definition dialog and modify the part's settings.
- Click on any label while holding the Command (macOS) or Windows (Windows) key to toggle the labels between a vertical and horizontal alignment. When horizontal, the label can be dragged up or down for easier part resizing.
- Open the label's contextual menu which contains the following features:
 - Part Definition — Opens the Part Definition dialog and allows modification of the part's settings.
 - Part Style — Opens a submenu listing all the available part styles for the layout's selected Theme (Chapter 32). Selecting a style will reformat the layout part according to the styles settings.
 - Fill Color — Opens a submenu containing a color palette, which is used to assign a new fill color to the part.

Identifying the Visible Layout Area of a Part

The *visible part area* of the layout defines a delimited portion of the layout space that runs horizontally from the left side of the window to the part boundary. This area contains the *stack* of all parts that make up the entire layout. This is the total area into which elements can be placed and still be rendered when viewing the window in a non-Layout mode. Nothing can be placed below the part stack as that area is non-layout space into which the part stack can extended as needed.

Identifying the Part Boundary and Off-screen Developer Area

The *part boundary* represents a vertical line that separates the entire stack of visible parts on its left from the *off-screen developer area* to its right. When a window is viewed in any other mode, everything on the left of this boundary will be rendered as the content area while everything to the right will remain hidden off-screen.

The off-screen area on the right can store notes and other layout elements that are not accessible to the end user since they will not be rendered.

Resizing a Part Area

Parts can be resized vertically and horizontally to any dimension that does not exceed the limit of 32,000 x 32,000 points.

To *vertically resize* a part, position the cursor at the line below the part area until the cursor changes into a short horizontal black line with two arrows on either size, pointing up and down. Then click, hold, and drag the cursor up or down to contract or expand the *height* of the part as needed. When the part labels are viewed horizontally, you can grab the label and drag to resize the part.

To *horizontally resize* the part stack, position the cursor at the part boundary line until it changes into a short vertical black line with two arrows on either size, pointing left and right. Then click, hold, and drag the cursor left or right to contract or expand the *width* of the part stack as needed.

Defining Part Types

There are ten different available part types, each controlling a certain type of information display and behavior. These are broken down into two categories: *standard (non-summary) parts* and *summary parts*.

Defining Standard Parts

A *standard part* displays layout elements without any summarization function. Each layout is limited to a *single* instance of each and they *must* fall within an automatically enforced stack order.

There are seven different standard part types:

- **Top Navigation** — This part type will appear *on screen only* at the top of the layout but will *not* print or appear in preview mode. It is intended for general information and navigation buttons and other controls. Fields placed in this part will display content drawn from whichever record is currently selected.
- **Title Header** — This part type will appear once at the top of the first screen or page when printing or previewing a layout but will not be displayed in Browse mode. When printing, it will replace the first instance of a Header and act like a unique title page or section. Fields placed in this part will display the content drawn from the first record in the found set.
- **Header** — This part type will appear at the top of every screen or printed page, except for the first instance when printing or previewing if a Title Header is present.
- **Body** — This part represents a single instance of a record. When viewing in a non-Layout mode such as Browse in a list view, the Body and every object placed within it will be repeated once for every record in the found set. When viewed as a form, this part type will display only the current record.

- **Footer** — Like a **Header**, this part type will appear at the bottom of every screen or page, except for the first instance if a **Title Footer** is present.
- **Title Footer** — Like a **Title Header**, this part type will appear at the bottom of the first screen or page, replacing the first instance of a **Footer** if one is present.
- **Bottom Navigation** — Like **Top Navigation**, this part type is intended to contain navigation buttons and other controls. It will only appear onscreen at the bottom of the layout but will not print or appear in preview mode.

Defining Summary Parts

A *summary part* is used to insert summarization values for groups of records. There are two types of summary parts, each with a leading and trailing variety, depending on their placement within the part stack relative to the **Body** (before or after it).

- **Grand Summary** — A summary field placed within this part type will display a summarized value for *all records in the found set*. It is used to calculate grand totals and can be placed at the beginning (**Leading Grand Summary**) or end (**Trailing Grand Summary**) of a report layout.
- **Sub-summary when sorted by** — A summary field placed within this part type will display summarized values for *a sorted sub-grouping of records* within the found set. It is used to calculate *subtotals* based on a specified break field that is used to separate records into sorted groups. One or more sub-summary parts can be placed both above or below the **Body** and they will appear *only* if records are sorted by the specified break field. A single sub-summary part will be repeated once for every group of records that result from the break field sort.

Managing Parts

Beyond the default parts automatically added during the layout creation process, parts can be added, deleted, and reordered as needed to create a simpler or more complex layout. This can be done directly in the window in **Layout mode** or through the **Part Setup** dialog.

Managing Parts Directly

When working directly in a window in **Layout mode**, you can add, edit, and delete parts.

Adding a Part Using the Toolbar Button

To add a new part into an existing layout using the toolbar, grab the **Part Tool** in the toolbar, shown in [Figure 28-5](#). As you drag down into the layout area, the cursor will turn into a clenched fist dragging a thin, black horizontal line. Move this line into a position just above or below a part that approximates the location you wish to insert the new part, and then let go. This will open a **Part Definition** dialog into which you can select and configure the desired part type. Once finished and the dialog is closed, you can resize the part as needed and begin adding layout objects within it.



Figure 28-5. *The Part Tool*

Tip Adding a part to a layout with this method is somewhat imprecise and always changes the size of an existing part. Instead, use the Part Setup dialog described later in this chapter to add new parts without adversely affecting existing ones.

Editing a Part

When creating a new part, the Part Definition dialog, shown in Figure 28-6, is automatically opened. An existing part can be edited by opening this dialog using any of the following methods:

- Double-clicking on the part label.
- Selecting a part label and then the Layout > Part Setup menu.
- Choosing Part Definition from the contextual menu when clicking on a part label.

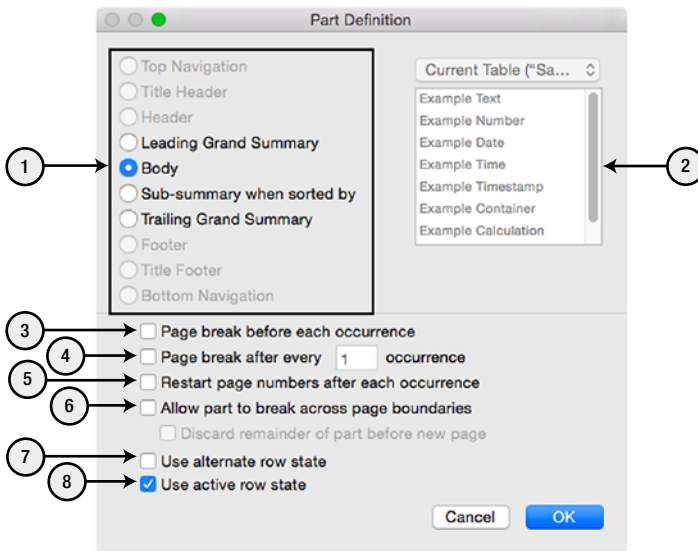


Figure 28-6. *The Part Definition dialog in which parts are configured*

This Part Definition dialog contains the following settings:

1. Part Type — This list of radio buttons allows the selection of a part type that will be assigned to the part being added or edited. These are automatically enabled or disabled to limit the selection to only those part types that are appropriate based on the part’s location within the part stack. For example, if editing a Top Navigation part, you can’t change it to a Title Header if there is already one defined nor can you change it to any other part type because of its location in the stack.

2. **Field Selection** — This combination of a pop-up menu of table occurrences and corresponding field list allows the selection of a field that will be used as the break field when defining a Sub-Summary part. For all other part types, these controls are disabled.
3. **Page break before each occurrence** — Enable this check box to insert a page break before a Trailing Grand Summary, Body or Sub-Summary.
4. **Page break after every X occurrence** — Enable this check box to specify the insertion of a page break after the part has been displayed a specified number of times. This makes it possible to limit the number of times a Body or a Sub-Summary part can be repeated on a single page.
5. **Restart page numbers after each occurrence** — Enable this check box to cause FileMaker's page numbering to restart after each occurrence of the part. This can be used with a Title Header to effectively create a title page that is not included in the numbering sequence or with a Sub-Summary to create separately numbered sections.
6. **Allow part to break across page boundaries** — FileMaker will not automatically split a part across a page boundary unless the contents of the part won't fit on a single page. Instead it will leave blank space at the bottom of the current page and start the part on the next page. However, enabling this check box will force the part to be split across a page break. The adjacent Discard remainder of part before new page check box will truncate any remaining content for the part instead of displaying it on the next page, effectively only displaying the portion of the part that fits in the available space at the bottom of the page.
7. **Use alternate row state** — Enable this check box to cause Body parts to be alternatively styled to visually provide a clear delineation between one record and the next.
8. **Use active row state** — Enable this check box to cause the Body part of the currently selected record to be differently styled to visually provide an indication of that state. This formatting option only affects onscreen display of records and does not appear in preview mode or when a layout is printed.

Deleting a Part

To delete a part, select the part label and type Delete.

Managing Parts with the Part Setup Dialog

A special Part Setup dialog, shown in Figure 28-7, can be opened by the following two methods:

- Select the Layout > Part Setup menu.
- Select Part Setup from the contextual menu available anywhere on the visible part area.

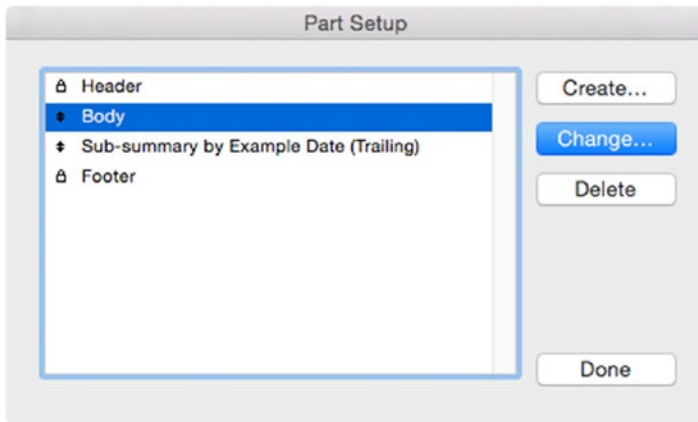


Figure 28-7. The *Part Setup* dialog allows for more precise control over the *part stack*

The *Part Setup* dialog displays a list of all the parts defined on the current layout and makes available the following actions:

- *Reorder* — Drag and drop existing parts to reorder them within the list if they are not locked into a position within the hierarchical stack because of their type restrictions, indicated with a lock icon next to the part name.
- *Add* — To create parts with more precision click the *Create* button. This will open the *Part Definition* dialog allowing configuration of the part type immediately. The new part will be inserted into the layout *between* existing parts without resizing or adversely affecting them.
- *Edit* — To change a part’s type or settings, select it in the list and click the *Change* button. This opens the *Part Definition* dialog for the selection.
- *Delete* — To delete a part from the list and the layout, select it and click the *Delete* button.

Adding Layouts

There are *two* different ways to add layouts to a database file:

- Creating a new layout
- Duplicating an existing layout

Creating a New Layout

A new layout can be created by any of the following methods:

- Select the *Layout > New Layout/Report* menu item.
- Clicking the *New Layout/Report* button in the toolbar.
- Clicking the *New* button on the *Manage Layouts* dialog.

Each of these will open the New Layout/Report dialog, shown in Figure 28-8. This dialog is a setup assistant, like a “wizard” in other systems, which guides you through the settings for the new layout. The options allow configuration of default settings based on the type of device for which the layout is intended or the function it will serve. However, any of these settings can be modified later and some developers just choose Computer and click Finish to bypass the setup assistant and quickly create a new empty layout.

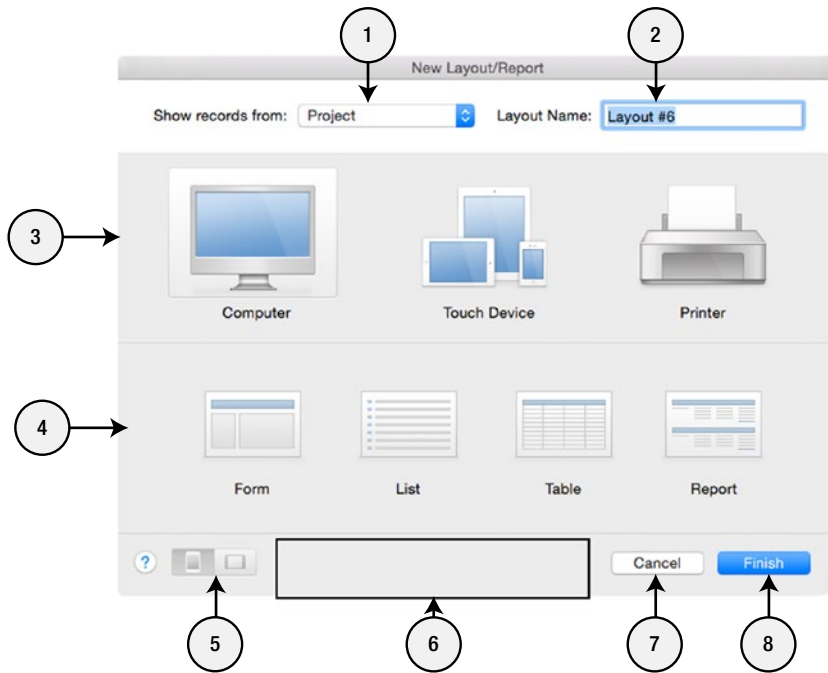


Figure 28-8. The New Layout/Report dialog

The New Layout/Report dialog contains the following controls:



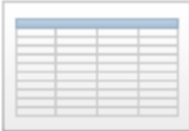
1. Show records from — This pop-up menu contains a list of every table occurrence defined in the database file. Choose the occurrence to indicate which records will be displayed on the layout and from what context within the relational model.
2. Layout Name — The default name for the new layout will be the layout number. However, the name in this text area can and should be modified to a more informative name that includes the table name and some detail about the function or format of the layout.
3. Device Type — Select the primary type of device for which the layout is intended. This selection will contextually change the available options on the remainder of the dialog. The Computer and Printer icons are buttons while the Touch Device icon in the center hides a pop-up menu with three choices: iPad, iPhone, and Custom Device.
4. Layout Type — Select from the available layout types. This will control the default configuration of the new layout and determine which other configuration dialogs will follow when clicking the Finish button.

5. **Touch Device Orientation** — If designing the new layout for a touch device, click on the portrait or landscape icon to indicate the desired orientation for the layout.
6. **Custom Device Measurements** — This area will display a width and height field for entering layout dimensions when designing for a custom touch device.
7. **Cancel** — Click this button to close the dialog without creating a new layout.
8. **Finish** — Clicking this button will continue the new layout configuration process. Certain layout types will open additional dialogs prior to the creation of the new layout.

Choosing a Layout Type




The New Layout/Report dialog contains six different types of layout choices, as defined in Table 28-1. Remember that these only determine which additional panels are presented in the process and which parts and objects will be included on the new layout.

Table 28-1. A list of layout choices when creating a new layout

Icon	Description
	<p>A <i>form layout</i> is generally used to view one record or find request at a time. They are so named since they resemble a paper form and are the primary layout format into which data is entered.</p>
	<p>A <i>list view</i> is used to display a list of multiple records in a found set. These are used to quickly scroll through records, sort them, compare values, and locate a desired record to which edits will be made after navigating to a form view.</p>
	<p>A <i>table view</i> is a kind of dynamic list view that displays records and fields in a format resembling a spreadsheet. Each record is a row and each field is a column. The user can rearrange and resize columns by dragging their headings or sort the records by clicking on a heading. New fields (columns) and new records (rows) can be created intuitively. Also, a layout actions pop-up menu is hidden within each field's heading and provides quick access to various features including sorting, summarization, field control, and view control. Table views aren't high on design and customized functionality but are a considerably more user flexible than plain lists and provide, in default functionality, a more familiar environment for those used to working with spreadsheets.</p>

(continued)

Table 28-1. (continued)

Icon	Description
	<p>A <i>report view</i> is a type of list view that is optimized for either viewing or printing summarized lists of data. Reports are similar to list views but generally have more complex grand and sub-summaries of data based on a sort order.</p>
	<p>A <i>label view</i> is a type of list or report view that is optimized for printing directly onto labels. These can be automatically created vertically or horizontally and sized to one of a list of preconfigured Avery or Dymo label formats, or developer-entered custom measurements.</p>
	<p>An <i>envelope view</i> is type of list or report view that is optimized for printing directly onto envelopes.</p>

Specifying Settings for Certain Layout Types

When creating a new layout, certain selections within the first dialog will result in one or more additional type-specific dialogs containing options. These dialogs are provided as a convenience, and any setting they apply can be applied or modified manually to an existing layout.

Additional Options for New Table Layouts

When creating a new Table layout, the Finish button on the New Layout / Report dialog will open two dialogs, one atop the other, as shown in Figure 28-9.

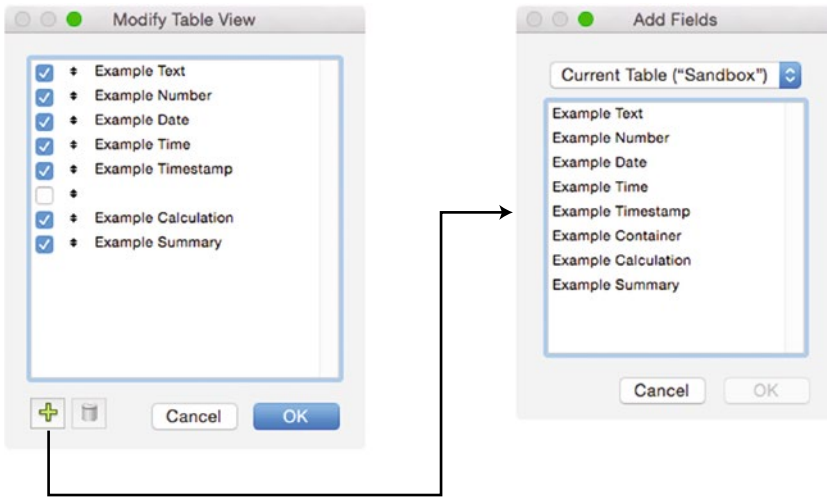


Figure 28-9. The *Modify Table View* and *Add Fields* dialogs for selecting which fields are options on a table view

On the top will be an *Add Fields* dialog, which allows the selection of one or more fields to be included on the new Table layout's list of available fields. The other, behind it and accessible when the first one is closed, is the *Modify Table View* dialog, which lists the fields already assigned to the layout and allows fields to be added or removed and activated or deactivated on the layout.

■ **Note** Any field physically placed on the layout will automatically be assigned *and* visible in the table. Any field only assigned through the *Modify Table View* interface will *only* be visible if enabled and can be disabled by users.

The Modify Table View Dialog

The *Modify Table View* dialog is opened automatically as part of the layout creation process behind the *Add Fields* dialog. It can be opened for existing layouts from the pop-up menu that is hidden in the heading of each field displayed on a Table layout. To do so, place the cursor over the heading, click on the small arrow that appears, and then select *Table > Modify* from the menu options.

The *Modify Table View* dialog contains the following controls:

- A list of all the fields that have been assigned to the layout from within the *Add Fields* dialog. The adjacent check box indicates fields that are actively displayed on the layout in browse mode while unchecked fields are attached to the layout but not currently visible.
- The + button is used to open the *Add Fields* dialog and add additional fields to the list.
- The Delete icon will delete the selected field(s) from the list if those fields were added through the *Add Fields* dialog. A lock icon will appear next to any field that was manually placed directly on the layout, indicating that they can't be deleted through this interface.

The Add Fields Dialog

The Add Fields dialog is opened automatically as part of the layout creation process but can be reopened later by clicking on the + button on the Modify Table View dialog.

This dialog allows the selection of one or more fields from the selected table occurrence and will add them to the list of assigned fields for the layout's Table view. The pop-up menu lists all table occurrences in the database file with a default selection of the table occurrence specified at the start of the New Layout / Report dialog process or the one assigned to the layout (if existing). The remaining occurrences listed in the menu can be selected to include fields from related tables into the Table view.

■ **Caution** When viewed in Layout mode, a Table view layout will not be displayed as a table, but as an arrangement of standard layout parts and manually inserted fields and other objects. The table formatting and features are superimposed when the layout is viewed in Browse mode. This allows a developer to offer users the option of viewing a layout in either a standard full-design Form or List view or, if they choose, as a Table. This can be helpful in some cases to allow a user to choose a more familiar or data-centric focus with other design elements obscured.

Fields in the list can be selected in a continuous sequence by clicking one at the start of the sequence and one at the end of the sequence while holding down the Shift key. Selecting a non-continuous group of fields can be accomplished by clicking on individual fields while holding the Command (macOS) or Windows (Windows) key.

Once finished selecting the desired field(s), click the OK button to close the dialog and save the selected fields into the available field list of the other dialog. Clicking the Cancel button instead will close the dialog without adding the select fields to the available field list.

Additional Options for New Report Layouts

New report layouts have a dizzyingly complex sequence of up to *eight* separate dialogs that step through the options for preconfiguring the parts, content, summarization, sorting, and other options for a new report layout. However, the options offered in these dialogs are an enormous convenience as it would be complex and laborious to manually configure them directly on a layout. To help illustrate the use of these, let's work through an example of an invoice report.

Defining Report Parts

A report layout can include many different parts, including multiple leading and trailing sub-summaries and grand summaries to create a complex hierarchical display of information. The illustration in Figure 28-10 dissects a hypothetical Invoice report into its component parts that, when sorted by Year and Month, summarize records into groups with subtotals.

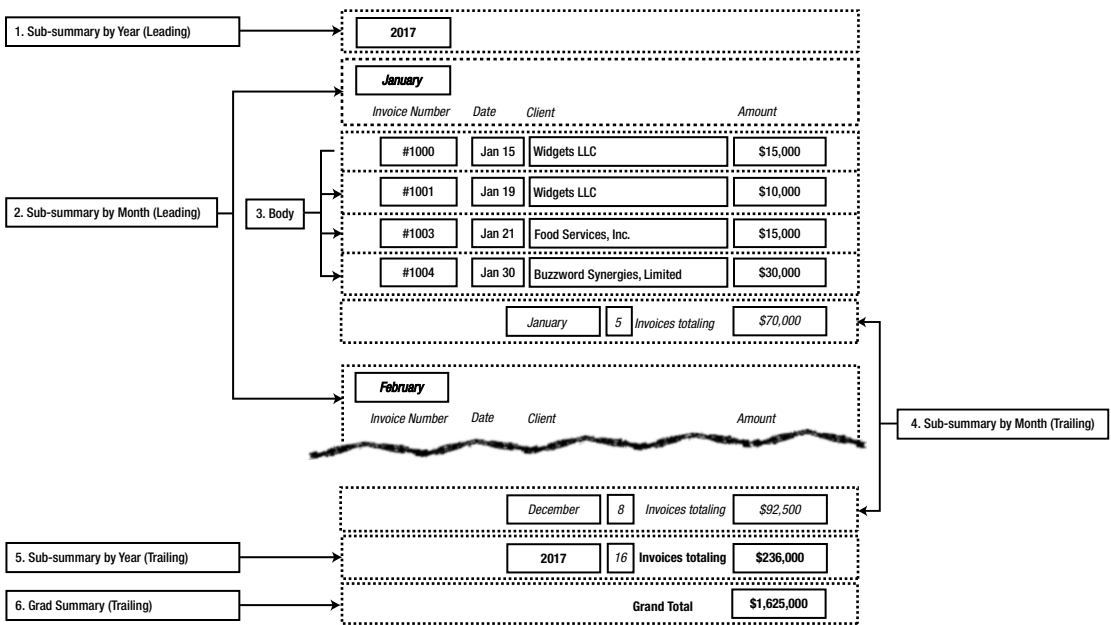


Figure 28-10. An illustration of the parts of a hypothetical invoice report summarizing sales by year and then by month

This report is made up of the following layout parts:

1. *Sub-summary when sorted by Year (Leading)* — when records are sorted by an Invoice Year field, this part will be repeatedly displayed once *before* each group of records where that field contains a unique value. It acts as a divider emphasizing the fact that the records to follow fall into a new year.
2. *Sub-summary when sorted by Month (Leading)* — when records are sorted by an Invoice Month field, this part will be repeatedly displayed once *before* each group of records where that field contains a unique value. It acts as a divider emphasizing the fact that the records to follow fall into a new month. Since this part appears immediately above the individual records is summarizes, field column labels are added here.
3. *Body* — repeats once for every invoice record in the found set, displaying the invoice number, date, client, and amount for each record.
4. *Sub-summary when sorted by Month (Trailing)* — when records are sorted by an Invoice Month field, this part will be repeatedly displayed once *after* a group of records with the same month and before starting a new batch of records with a different value. Summary fields placed here will automatically calculate a subtotal of the counts and totals for the group of records summarized for the month.
5. *Sub-summary when sorted by Year (Trailing)* — when records are sorted by an Invoice Year field, this part will be repeatedly displayed once *after* a group of records with the same year and before starting a new batch of records with a different value. Summary fields placed here will automatically calculate a subtotal of the counts and totals for the group of records summarized for the year.

6. *Grand Summary (Trailing)* — At the end of the report, after all the Invoice records and various sub-summaries have been displayed for the entire found set, a *trailing grand summary* part will appear. Summary fields placed here will automatically calculate a grand total of the counts and totals for the entire found set.

The Invoice Count Summary and Invoice Total Summary are the same field in both sub-summaries and the trailing grand summary. These are summary fields that generate a count of the Invoice Number field and a total of the Invoice Total fields. Since they are placed within a summary part type, they automatically summarize the values for the records in the group they subsume.

■ **Tip** When creating a new report record, most dialogs include a small color coded graphical representation of the report being built. This allows you to visually understand which part(s) of the layout are being referred to by the interface controls in the setup assistant.

Preparing for the Invoice Report Demonstration

To construct the layout illustrated in Figure 28-10, an Invoice table must be defined containing some entry, summary, and calculation fields, as shown in Figure 28-11.

Invoice Date	Date	
Invoice Amount	Number	
Invoice Customer	Text	
Invoice Number	Text	
Invoice Amount Summary	Summary	= Total of Invoice Amount
Invoice Count Summary	Summary	= Count of Invoice Number
Invoice Year	Calculation	= Year (Invoice Date)
Invoice Month	Calculation	= Month (Invoice Date)
Invoice Month Name	Calculation	= MonthName (Invoice Date)

Figure 28-11. The field definitions for the report layout demonstration to follow

The first four fields are used for data entry to store the date, amount, customer, and number of an invoice. The two summary fields will be used to display the total count and the total dollar amount of invoices in the various sub-summary and trailing grand summary parts. The three calculation fields are used to sort records and display values in summary fields.

To begin creating a new report layout, open the New Layout/Report dialog and follow these steps:

1. Select the Invoice table.
2. Enter a Layout Name of “Invoice Report.”
3. Click on the Computer icon.
4. Click on the Report icon.
5. Click the Continue button.

This will begin a sequence of eight steps of the setup assistant for report layouts.

Step 1 — Include Subtotals and Grand Totals

The Include Subtotals and Grand Totals dialog shown in Figure 28-12, contains two check boxes providing the option to Include subtotals and to Include grand totals.

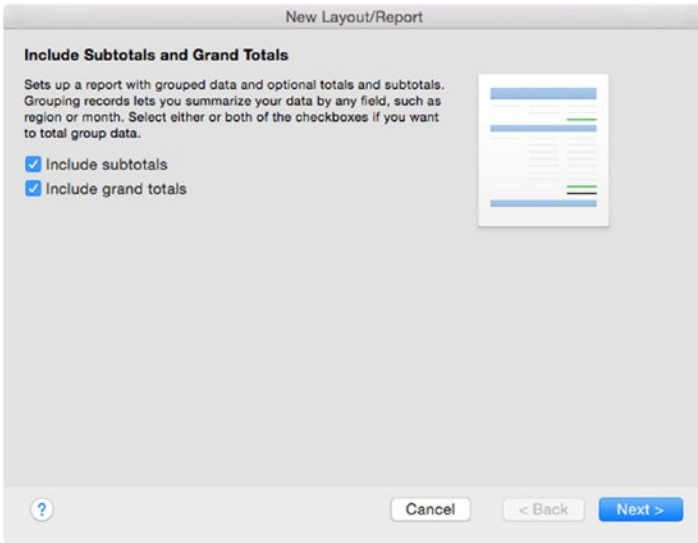


Figure 28-12. The first step in the sequence for creating a new report layout is the Include Subtotals and Grand Totals dialog

Select both options so that the Specify Subtotals dialog (Step 6) and Specify Grand Totals dialog (Step 7) are included in the configuration process. Click the Next button to continue the layout setup process.

Step 2 — Specify Fields

The next dialog in the sequence is the Specify Fields dialog, shown in Figure 28-13, with the fields necessary to build the Invoice Report. This dialog allows the selection and ordering of the fields that will be placed somewhere on the report.

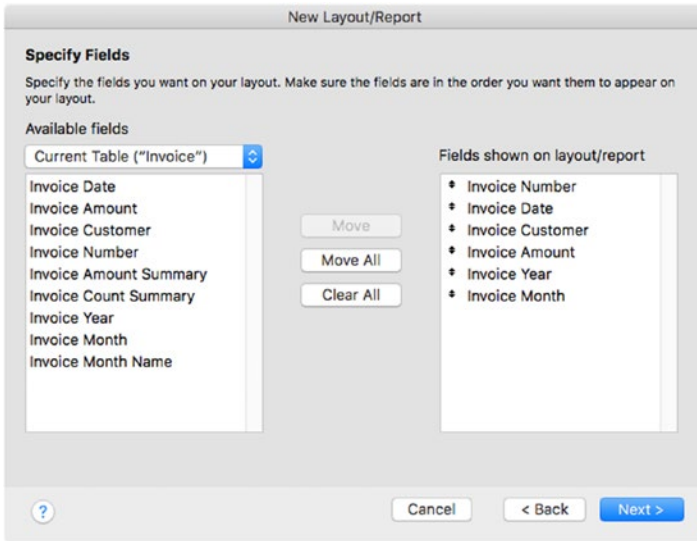


Figure 28-13. The second step in the sequence for creating a new report layout is the Specify Fields dialog

To add fields to the list, select the appropriate table occurrence from the pop-up menu and then use one of these options:

- Click the Move All button to copy all the available fields over to the layout list.
- Select one or more available fields and click the Move button to copy them over to layout list.
- Double-click directly on individual available fields to quickly copy them over to the layout list.

Once added, dragging fields up or down in the list will determine the order in which they are added to the new report layout.

When the desired fields are selected, click the Next button to continue to the next step.

Step 3 — Organize Records by Category

The next dialog in the sequence is the *Organize Records by Category* dialog, shown in Figure 28-14, in which you can specify one or more sort fields that act as grouping criteria for the report's sub-summaries.

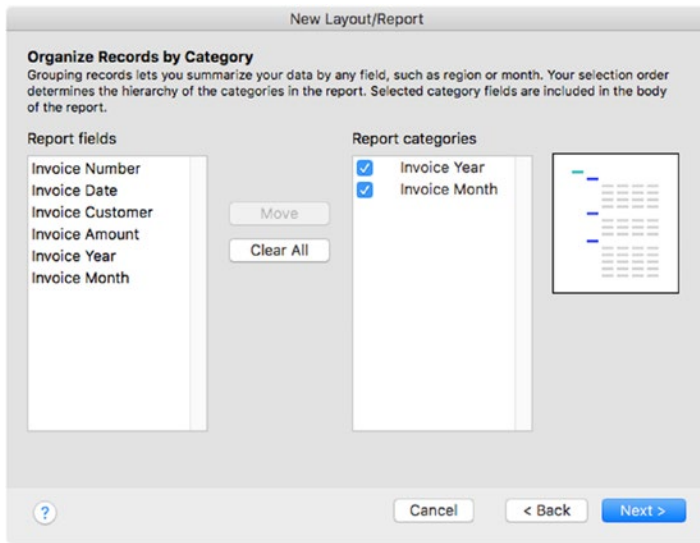


Figure 28-14. The third step in the sequence for creating a new report layout is the *Organize Records by Category* dialog

Report fields selected in Step 2 are displayed in the left-hand list. These can be added to, removed from, or enabled in the Report categories list as follows:

- To *add* one or more fields into the Report categories list, double-click directly on them in the Report fields list or select them and click the Move button.
- To *remove one* field from the Report categories list, double-click on it in list or select it and click the Clear button.
- To *remove all* fields from the Report categories list, click the Clear All button.

Once in the list, a field will be included as a summarizing category for groups of sorted records. Enable the adjacent check box to include a field in *both* the sub-summary layout part and the body of the report.

In our invoice report example, the Invoice Year and Invoice Month fields are both added as categories because these will be used to summarize groups of Invoice records.

Once the desired options are selected, click the Next button to continue to the next step.

Step 4 — Sort

Next in the sequence is the Sort dialog, shown in Figure 28-15, in which you can specify a sort order from the list of fields added to the report layout. This dialog contains a typical sort interface (Chapter 4, “Sorting Records in the Found Set”).

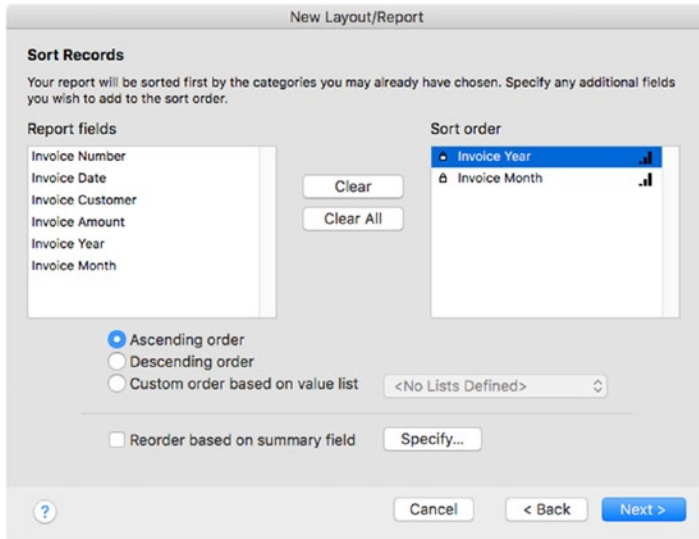


Figure 28-15. The fourth step in the sequence for creating a new report layout is the Sort dialog

The Report fields list on the left contains a list of every field added to the report. The Sort order list on the right contains a list specifying how the records will be sorted when the report is generated. Any fields added as Report categories in Step 3, will automatically appear in the sort list and be locked at the top of the sort order, as they are required to establish the summarized groups.

One or more fields from the Report fields list can be added to the Sort order list, each with the ability to be sorted in Ascending order, Descending order, or in a Custom order based on value list. There is an option to Reorder based on a selected summary field. Sort fields will be used in several forthcoming steps.

In our invoice report example, the Invoice Year and Invoice Month fields are both added as categories because these will be used to summarize groups of Invoice records. They must be placed in the same order as their corresponding sub-summary parts on the layout: year first, month second.

Once the desired options are selected, click the Next button to continue to the next step.

Step 5 — Specify Subtotals

The Specify Subtotals dialog, shown in Figure 28-16, will only appear if the Specify subtotals check box was selected in Step 1. This dialog allows the selection of one or more summary fields that can be displayed above or below groups of records sorted and categorized by selected fields.

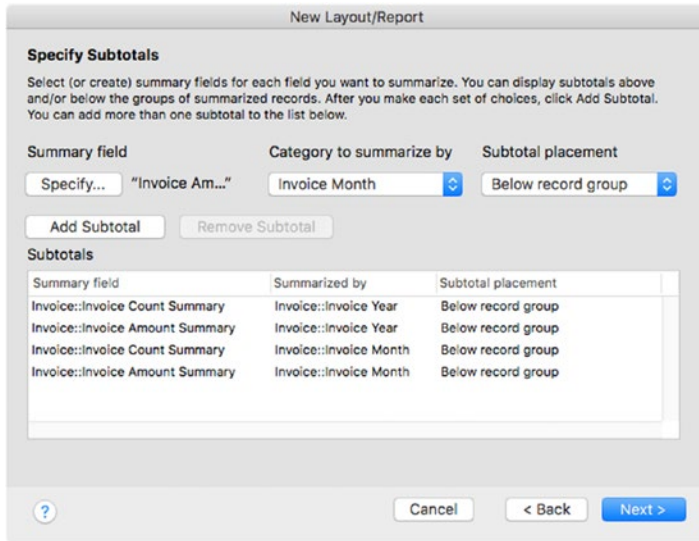


Figure 28-16. The fifth step in the sequence for creating a new report layout is the Specify Subtotals dialog

Across the top, a summary field can be selected for inclusion on the report by clicking the Specify button. Next, a field can be selected from the Category to summarize by pop-up menu, which lists all the fields added as report categories in Step 3. The Subtotal placement pop-up menu specifies where the subtotal will appear in relation to the group of records that it is summarizing, with the following choices:

- Above record group
- Below record group
- Above and below

Once these choices are made along the top of the dialog, click Add Subtotal to insert them into the list below. Multiple summaries can be added to the list to build up a more robust report with additional levels of summarized hierarchy. To remove one of the Subtotals from the list, select it and click the Remove Subtotal button or type the Delete key.

In our invoice report example, the Invoice Count Summary and Invoice Amount Summary fields are both added as subtotals that will appear below the record group, summarized by *both* Invoice Month and Invoice Year so that they appear in both sub-summaries.

Once the desired options are selected, click the Next button to continue to the next step.

Step 6 — Specify Grand Totals

The Specify Grand Totals dialog, shown in Figure 28-17, will only appear if the Specify grand totals check box (Step 1) was selected. This dialog works like the subtotal options in Step 5 but uses summary fields to display a total of *all records* on the report rather than a subset of them.

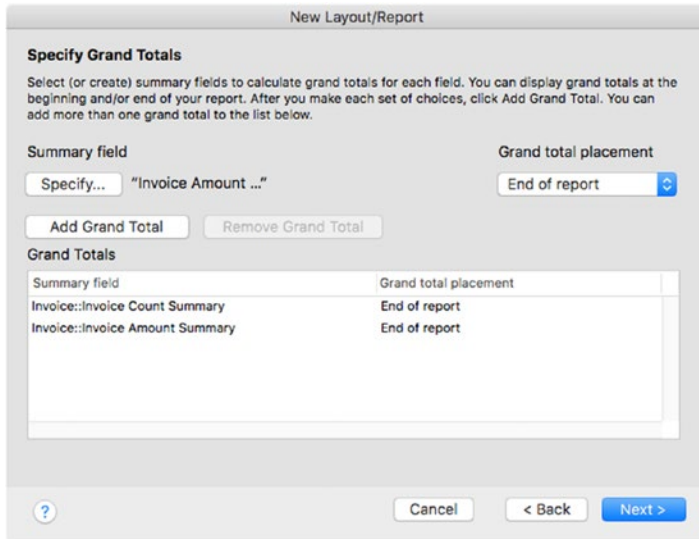


Figure 28-17. The sixth in the sequence for creating a new report layout is the Specify Grand Totals dialog

A summary field can be selected and added to the list of grand totals with the Grand total placement pop-up allowing a choice of a location for the grand total relative to the entire report, with the following options:

- Beginning of report
- End of report
- Beginning and end

Once these choices are made along the top of the dialog, clicking Add Grand Total will insert them into the list below. Multiple summaries can be added to the list to create a more robust report. To remove a Grand Total from the list, select it and click the Remove Grand Total button or type the Delete key.

In our invoice report example, the Invoice Count Summary and Invoice Amount Summary fields are both added as grand totals that should appear at the end of the report.

Once the desired options are selected, click the Next button to continue to the next step.

Step 7 — Header and Footer Information

The Header and Footer Information dialog, shown in Figure 28-18, allows the selection from a menu of standard information that can be automatically placed on the new layout in six different locations: the left, center and right on the Header (top) and Footer (bottom).

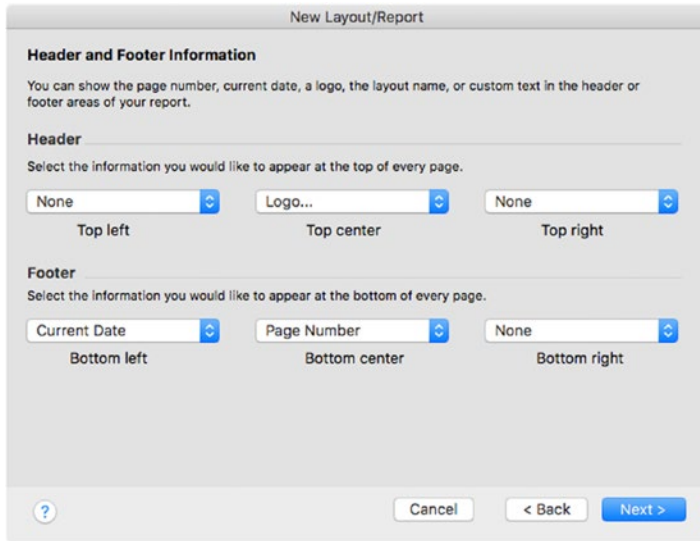


Figure 28-18. The seventh step in the sequence for creating a new report layout is the Header and Footer Information dialog

The pop-up menus are the same for all six options and include the following values:

- None — Nothing will be added to this region on the new layout.
- Page Number — This choice will place the `{{PageNumber}}` placeholder on the layout in the desired position. This symbol will be dynamically replaced with the current page number when printing the report or viewing in Preview mode.
- Current Date — This choice will place the `{{CurrentDate}}` placeholder on the layout in the desired position. This symbol will be dynamically replaced with the current date when printing the report or viewing in Preview mode.
- Layout Name — This choice oddly places a literal text string containing the layout name on the layout in the desired position instead of the dynamically updating `{{LayoutName}}` placeholder.
- Large Custom Text — This choice displays a Custom Text dialog, shown in Figure 28-19, into which any custom text string can be entered, which will be added to the layout in the desired position in a large font size.

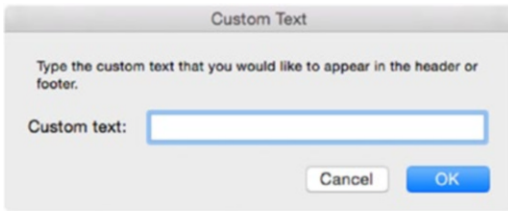


Figure 28-19. The Custom Text dialog for specifying literal text to add to the newly created layout

- Small Custom Text — This choice displays a Custom Text dialog, into which any custom text string can be entered, which will be added to the layout in the desired position in a small font size.
- Logo — This choice displays an Insert Picture file selection dialog, which allows the selection of a picture file, which will be added to the layout in the desired position.

Once the desired options are selected, click the Next button to continue to the next step.

Step 8 — Create a Script

The Create a Script for this Report dialog, shown in Figure 28-20, allows the automatic creation of a FileMaker script to generate the report.

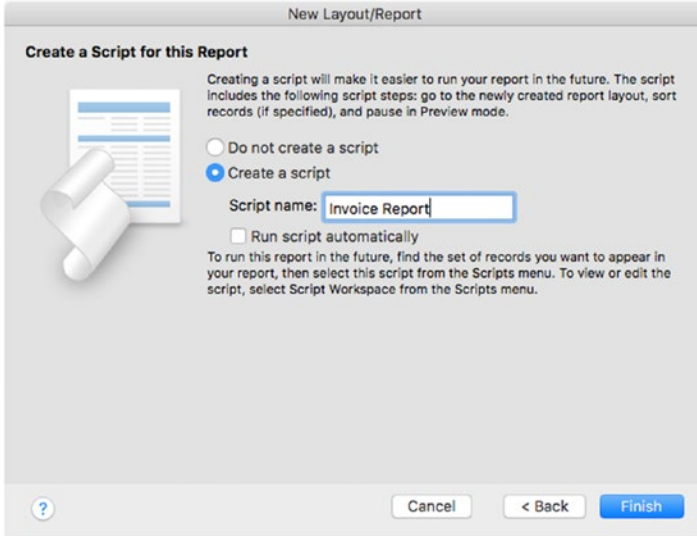


Figure 28-20. The eighth step in the sequence for creating a new report layout is the Create a Script for this Report dialog

With the Do not create a script radio button selected, the setup process will not create a script for the report. The user must perform a find, navigate to the report, sort records, and print or preview the report manually, unless a developer manually creates a script to do so.

Clicking the Create a Script radio button will prompt the report setup process to automatically create a script. Enter a name for the new script in the adjacent Script name text area. The name will default to the same name as the layout being created, but can be modified to be more descriptive if desired. The script will include up to three steps automatically configured depending on other options in this process:

- Enter Browse Mode
- Go to Layout
- Sort Records

The Sort Records script step will only be included if sort fields were specified in Step 4. Selecting the Run script automatically check box, will automatically assign an OnLayoutEnter Script Trigger (Chapter 37) that will be configured to run the new script whenever a user or another script navigates to the layout being created.

Once the desired options are selected, click the Finish button to continue to complete the creation of the new layout.

Refining the Report Layout

Once the report layout is created, it can be further refined and customized as needed. The example shown in Figure 28-21 shows the resulting Invoice Report layout based on the options selected in the setup assistant and helps to illustrate the need for further refinement.

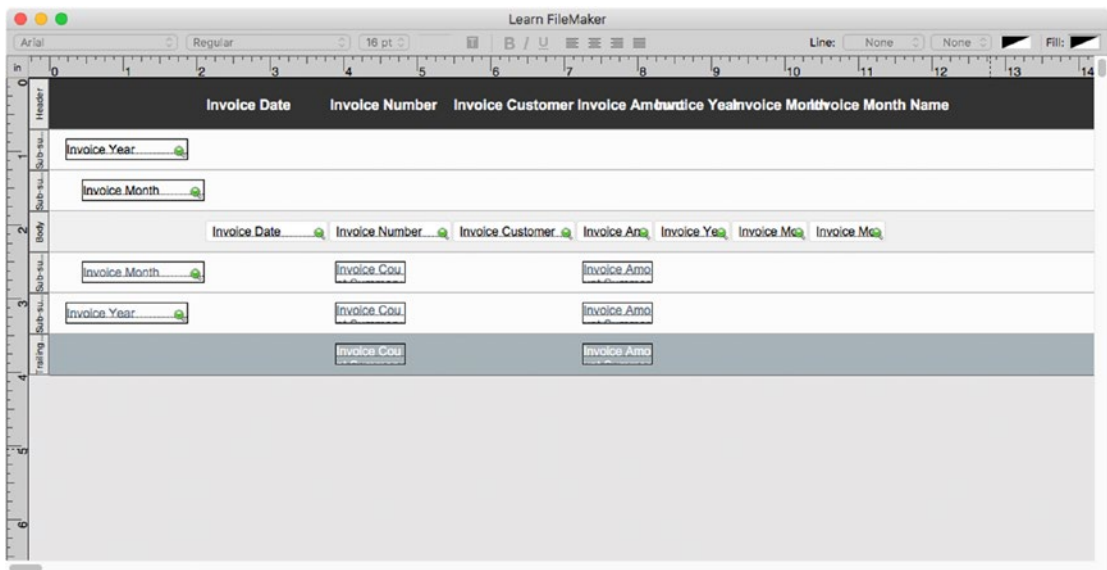


Figure 28-21. The layout automatically created based on the options selected in the setup assistant dialogs

Some of the issues with the default report configuration include:

- The height of various parts might need to be tightened up to save space.
- The background color choices will be based on your current theme and may not be appropriate for a printed report.
- If a logo was included, the size and position might require adjustment.
- Additional text can be added to the Header such as a company and report name.
- The font, size, and position choices for the fields may not suit your preferences.
- The field labels for the body are automatically added into the Header. These can be moved down to the sub-summary just above the body part so they are repeated directly above each group of fields. Also, the field labels automatically show the full name of the field and may overlap as shown. These can be edited and adjusted in size to make them fit more appropriately.

Additional Options for New Label Layouts

When creating a new layout for printing labels, there are two additional dialogs that assist in the configuration of the layout.

Step 1 — Specify Label Layout

The Specify Label Layout dialog, shown in Figure 28-22, provides an intuitive interface for specifying label dimensions and assist in creating a new layout for printing labels.

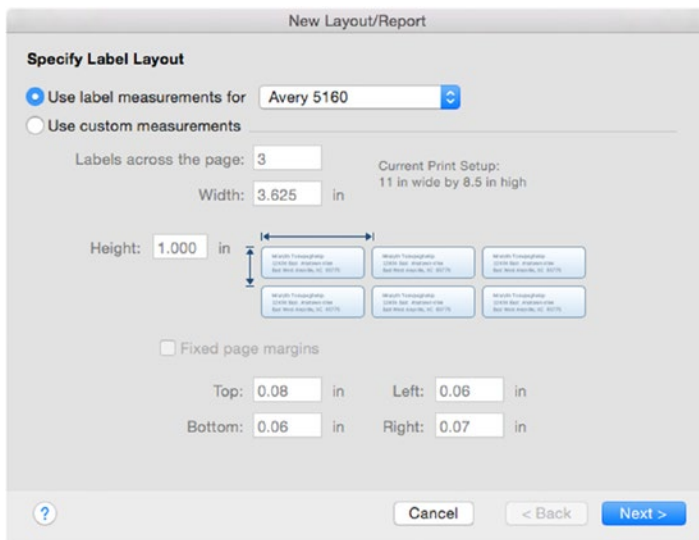


Figure 28-22. The Specify Label Layout dialog

The dialog contains two primary choices:

- Use `label measurement` for — Choosing this option activates a pop-up menu listing popular Avery and Dymo label sizes. Once selected, the dimensions and configuration of labels on the page are automatically entered.
- Use `custom measurements` — This option enables the label measurement settings in the lower portion of the dialog. Use this to enter custom configuration and dimensions for labels not available using standard labels. Here you can enter the number of labels across a page, width, and height, and optionally, edit fixed page margins for Top, Bottom, Left, and Right.

Step 2 — Specify Label Contents

The second step when creating a label layout is the `Specify Label Contents` dialog, as shown in Figure 28-23. This dialog allows the quick construction of a single sequence of merge fields that will be automatically placed on the layout when it is created.

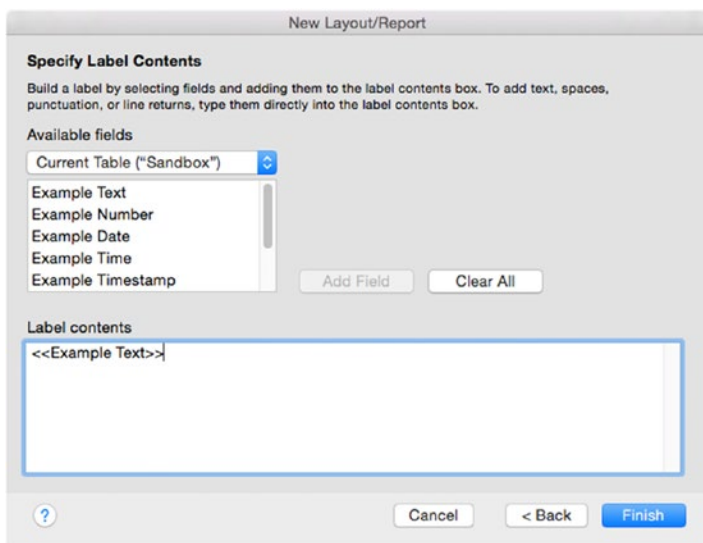


Figure 28-23. *The Specify Label Content dialog*

This dialog allows the selection and insertion of merge field tags into the `Label contents` text area. Mentioned Chapter 27 and 30, a *merge field* is the name of a field enclosed in a set of left-shift (<<) and right-shift (>>) symbols that, when present on a layout, will be rendered in non-Layout modes as actual field contents for the current record. Merge fields can be intermixed with any other text contents such as static values and paragraph carriage returns to create a formula for generating multi-line label content.

To add a field to the `Label contents` area, either double-click it in the list or select it in the list and click the `Add Field` button. You can selectively edit the `Label contents` by highlighting some or all the text and typing the `Delete` key. To completely clear all data entered so far, click the `Clear All` button.

Additional Options for New Envelope Layouts

When creating a new layout for printing envelopes, there is a single dialog that assists. The Specify Envelope Contents dialog, shown in Figure 28-24, enables the quick construction of a sequence of merge fields that will be automatically placed on the layout when it is created.

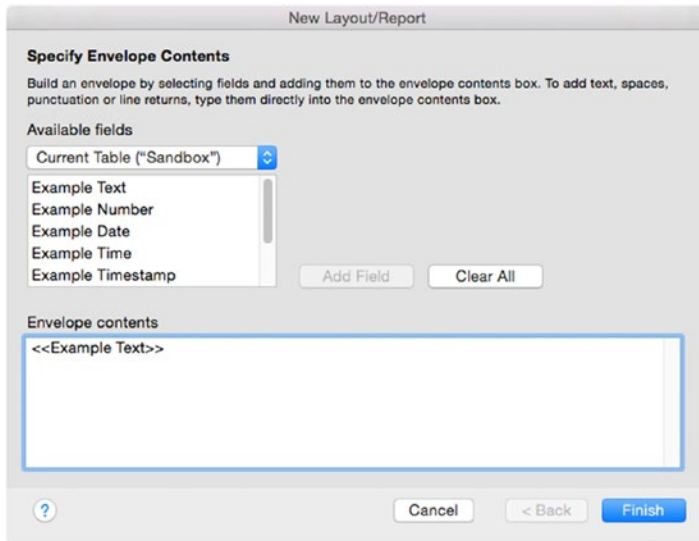


Figure 28-24. The Specify Envelope Content dialog

This dialog allows the selection and insertion of merge field tags and other text in the exact same manner as the Specify Label Contents dialog.

Duplicating an Existing Layout

Duplicating an existing layout has the advantage of precisely replicating the position and size of every layout part and object from the original onto the duplicate. This is a benefit when designing many complex layouts with a uniform design.

There are two methods to duplicate a layout:

- Navigate to the layout that you want to duplicate and select the **Layouts > Duplicate Layouts** menu.
- Open the **Manage Layouts** dialog from the **Layouts** pop-up menu in the toolbar and use the duplicate feature there (see “Managing Layouts” later in this chapter).

Either method will immediately create a duplicate copy of the layout and display it as the current layout. The new layout will have the same name as the original with the word “Copy” appended to it. Everything else about the layout will be the same as the original including all parts, theme, objects, formatting, and settings as well as the assigned table occurrence. Once created, the new layout can be renamed and edited as needed.

Configuring Layouts

Once a layout is created, there are many layout configuration settings that customize its features and behavior.

Exploring the Layout Setup Dialog

The Layout Setup dialog contains options that control the behaviors, appearance, and functions of the layout. This dialog can be opened by any of the following methods:

- Select the Layouts > Layout Setup menu.
- Click the Layout Setup icon in the lower portion of the toolbar next to the Layout pop-up menu.
- From the Manage Layouts dialog, double-click a layout or select a layout and clicking the Edit button.

The dialog is divided into four tabs:

- General
- Views
- Printing
- Script Triggers

General

The General tab of the Layout Setup dialog, shown in Figure 28-25, contains general settings of the current layout.

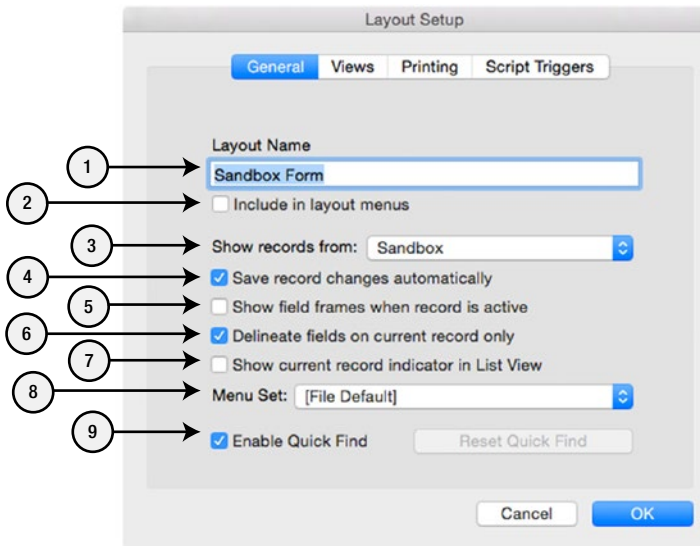


Figure 28-25. The General tab of the Layout Setup dialog, showing default values for check boxes

The controls on the General tab of the Layout Setup dialog include:

1. **Layout Name** — This text area contains the name of the layout and can be modified as needed. Although FileMaker will allow you to create two layouts with the same name, to avoid confusion, this should be avoided.
2. **Include in layout menus** — Check this box to have the layout name appear in the toolbar Layout pop-up menu and the View > Go to Layout menu when viewing the database in any non-Layout mode.
3. **Show records from** — This pop-up menu contains a list of every table occurrence in the database. The selected occurrence establishes both the relational context for the layout and determines which table's records will be displayed on the layout when browsing. If the selection is changed for a layout, the relationship context for any fields will need to be changed.
4. **Save record changes automatically** — Check this box to automatically save changes anytime a user or script commits a record by clicking on the background away from any field, navigating to a new record or closing the file. If left unchecked, when performing any of these actions, a dialog will appear asking if the user wants to save changes, giving them the following options:
 - **Save** — closes the record and saves changes.
 - **Cancel** — returns to edit mode within the record without saving.
 - **Don't Save** — reverts to the previous values in every field that was modified during the user's editing session.
5. **Show field frames when record is active** — When editing a record, the field with focus will be styled uniquely according to its layout settings. Checking this box will cause the borders of every other field on a layout to be displayed with dotted lines, emphasizing that the record has editing focus.
6. **Delineate fields on current record only** — Enabling this check box will change the way field borders and fills are displayed in Browse mode when viewing List layouts. When checked, only the current record will display field border and fill settings. While fields on other records will still draw their content, they will not display any border or fill settings at all if this box is enabled. Any empty fields will seem to disappear.
7. **Show current record indicator in list view** — When this box is checked, a small vertical black bar will appear along the left side of the currently selected record in a list view. This provides a somewhat outdated way of indicating which record has active focus. The *Use active row state* check box on the Part Definition dialog for the Body part provides a preferable modern method of identification of the current record.
8. **Menu Set** — This pop-up menu lists all menu sets defined in the file and an option to open the Manage Custom Menus dialog (Chapter 33).
9. **Enable Quick Find** — This check box allows enabling or disabling of the Quick Find feature (Chapter 4) for the entire layout. When the feature is enabled and some fields have had their Quick Find feature turned off in the Layout Inspector (Chapter 29), clicking the Reset Quick Find button here will return all fields back to their default setting.

Views

The Views tab of the Layout Setup dialog, shown in Figure 28-26, controls which Content Views will be available to users for the current layout (Chapter 3, “Defining Content Views”).

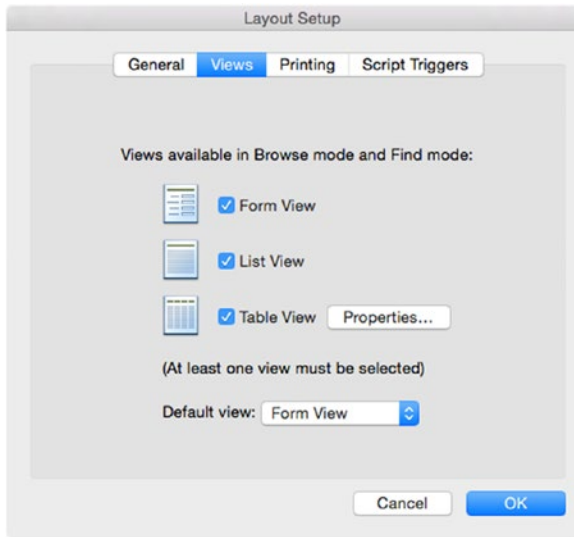


Figure 28-26. The Views tab of the Layout Setup dialog, showing default values for check boxes

This dialog contains a set of three check boxes that determine which content view types are accessible to a user through the toolbar and which is the default when a user or script first navigates to a layout.

When the Table view check box is selected, the adjacent Properties button becomes enabled. This button will open the Table View Properties dialog, shown in Figure 28-27, which provides control over how a layout appears in Browse mode when viewed in Table view.

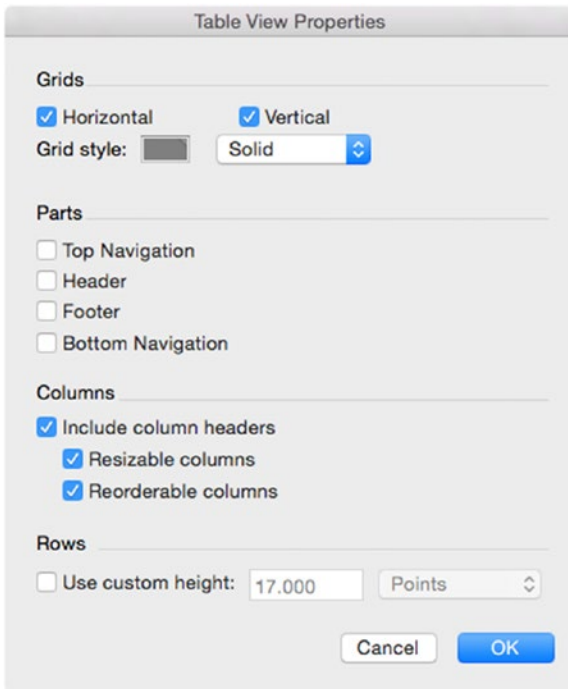


Figure 28-27. The Views tab of the Layout Setup dialog, showing default values for check boxes

This dialog has the following controls for Grids:

- **Horizontal** — Check this box for the view to display grid lines horizontally between records.
- **Vertical** — Check this box for the view to display grid lines vertically between fields.
- **Grid style** — The color well contains the current color selection for grid lines. Click here to open a color palette from which you can select and assign a new color.
- **Line Style** — The pop-up menu selection controls the line style and contains the following values: None, Solid, Dashed, and Dotted.

The next section of the dialog contains four check boxes that determine if those Parts, when present in the layout's defined structure, will be visible in other non-layout modes. The choices are:

- Top Navigation
- Header
- Footer
- Bottom Navigation

The third section of the dialog contains choices that control column headings. The options are:

- **Include column headers** — Check this box to make column headers visible. When selected, the two other check boxes become enabled.
- **Resizable columns** — Check this box to allow a user to resize the columns for their view.
- **Reorderable columns** — Check this box to allow a user to reorder the columns by dragging and dropping them between other columns.

The final section of the dialog controls the height of the record rows. Check the box to activate custom row height and enable the adjacent controls. Then you can enter a row height number in the text area and choose a unit of Inches, Centimeters, or Points.

Printing

The Printing tab of the Layout Setup dialog, shown in Figure 28-28, contains controls for print settings of the layout's columns and page margins. These should not be confused with settings for print and paper size that are accessible through the File > Page Setup and File > Print menus. Instead, these focus on spacing and columns of the *layout* as a means of controlling the available printable area in which to place fields and other layout objects.

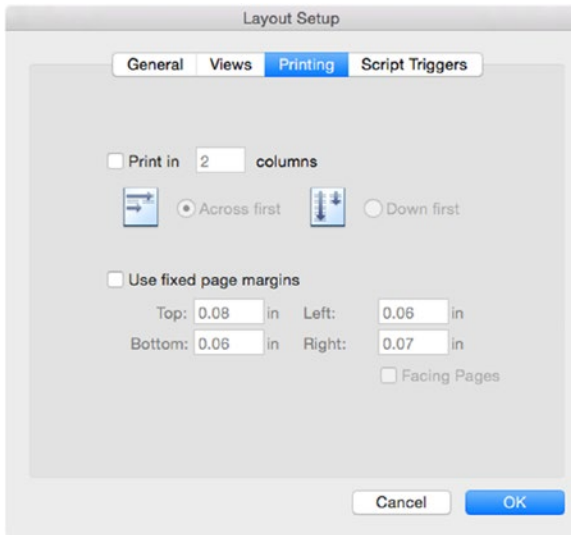


Figure 28-28. The Printing tab of the Layout Setup dialog

Specifying Multiple Columns

Selecting the `Print In` check box will cause the layout to be divided into the number of columns specified in the adjacent text area. When selected, the radio buttons immediately below become enabled and allows a choice of the following two directional flows for the records:

- `Across first` — This option will cause records to flow in order from left to right in as many columns as specified before starting a new row and continuing this process down the page.
- `Down first` — This option will cause records to flow from top to bottom until reaching the end of the page and then moving to the next column, repeating this for the specified number of columns before starting a new page.

When configured to print in columns, certain layout parts will appear shaded in `Layout mode`. This indicates non-editable areas that will become additional columnar instances of the unshaded areas.

Specifying Fixed Page Margins

When selected, the `Use fixed page margins` check box specifies fixed page margins in the text areas provided.

By default, you can enter a measurement for the `Top`, `Bottom`, `Left`, and `Right` of the page. The units for these text areas will match the current unit selection in the window's ruler. When the `Facing Pages` check box is selected, the `Left` and `Right` measurement are renamed and repurposed for `Inside` and `Outside` measurements.

Script Triggers

The `Script Triggers` tab of the `Layout Setup` dialog, shown in [Figure 28-29](#), contains controls for triggering scripts in response to layout events ([Chapter 37](#)).

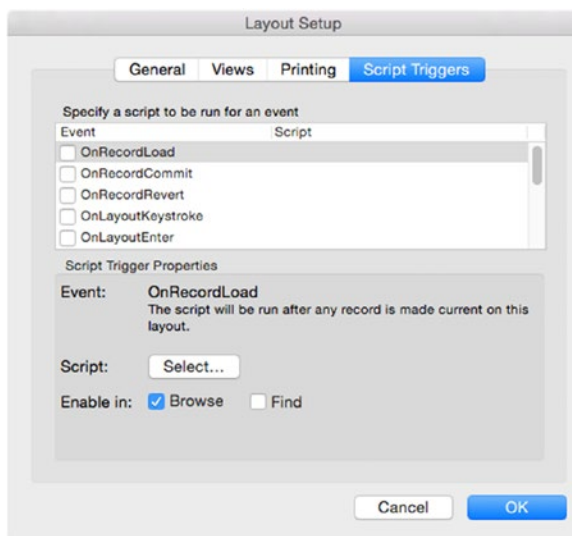


Figure 28-29. The `Script Triggers` tab of the `Layout Setup` dialog

Deleting Layouts

Deleting an unused layout can be performed one of two ways:

- Navigate to the layout that you want to delete and select the **Layouts > Delete Layouts** menu item.
- Open the **Manage Layouts** dialog from the **Layouts** pop-up menu in the toolbar and use the delete feature there.

FileMaker will present a dialog confirming that you want to delete the layout. Click **Delete** to permanently delete the layout from the file or **Cancel** to stop the deletion process.

Using the Manage Layouts Dialog

At the top of the **Layout** pop-up menu in the second row of the toolbar is a **Manage Layouts** option that opens a dialog of the same name, shown in Figure 28-30.

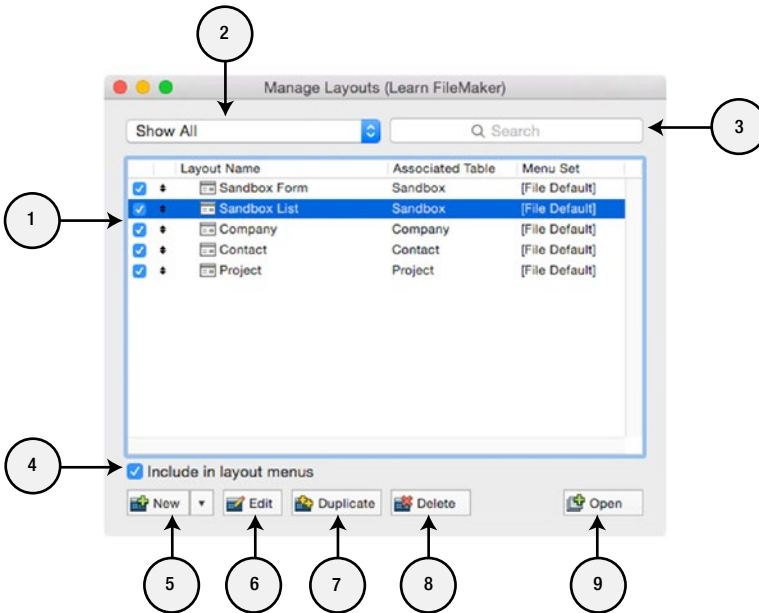


Figure 28-30. The *Manage Layouts* dialog

The Manage Layouts dialog integrates into one simple dialog the ability to quickly create, view, locate, edit, duplicate, delete, and open layouts. It also provides the exclusive functions of reordering, adding separator lines and grouping layouts into folders. The following controls are available on this dialog:

1. **Layout List** — The central area contains a list of every layout defined in the file, showing columns containing the Layout Name, Associated Table, and Assigned Menu Set. Drag a layout, folder, or separator line up and down the list to reorder, or drag them inside of folders to re-group.
2. **Quick Filter Popup** — This pop-up menu contains a list of every layout folder defined in the list below. Selecting a folder from the list will filter the list down to include only those layouts that are contained within that selection. The default value, and the only option when there are no folders defined, is Show All.
3. **Search Filter** — Type text into the search field to filter the list of layouts to only include those whose name contains the text entered.
4. **Include in layout menus** — This check box and the ones above it next to each individual layout determine if the layout name will appear in the toolbar's Layout pop-up menu and the View > Go to Layout menu when in non-Layout modes such as Browse. Select multiple layouts to enable or disable this check box across a batch of layouts all at once.
5. **New** — This button and the attached pop-up menu create items in the list above. Click the button or select New Layout/Report from the adjacent menu to start the process of adding a new layout to the file, described earlier in this chapter. The menu includes options for creating a separator line and folder in the list, which are both useful methods for delimiting and organizing layouts.
6. **Edit** — Select a layout and click this button to open the Layout Setup dialog.
7. **Duplicate** — This button will duplicate the layout(s) selected in the list.
8. **Delete** — This button will delete the layout(s) selected in the list after presenting a confirmation dialog.
9. **Open** — This button will open a new window for the selected layout(s).

■ **Tip** As a database file's list of layouts grows, it is a good idea to organize them into folders. The easiest way to do this is by their associated table; however, you may decide to use some other organizational methodologies based on your own preferences. Folders can be nested hierarchically, so you can create sub-groupings based on various criteria, such as category of table or making a distinction between layouts intended for users vs. layouts intended for developers or hidden script access.

Optimizing Layout Performance

FileMaker provides some guidance for optimizing layouts for efficient performance. These include:

- In places where multiple records are displayed at once, like a List view or portal, try to minimize the number of layout objects displayed simultaneously.
- Within a single layout, try to minimize the number of advanced controls such as portals and panels.
- Divide complex layouts across multiple smaller layouts instead of trying to fit everything into a single view.
- Keep the quantity and size of shadows on objects to a minimum.
- Minimize the number of unstored calculations.
- Use script triggers sparingly, only when necessary.
- Use custom themes and custom styles instead of overriding formatting directly on objects without updating the theme.

Summary

In this chapter, we discussed planning an interface and creating, deleting, managing, and optimizing layouts.

CHAPTER 29



Exploring Layout Palettes

FileMaker provides three palettes for adding, manipulating, and configuring layout objects. In this chapter, we explore all the options available on these:

- Field picker
- Inspector
- Layout objects

A familiarity with the various tools and settings available on these palettes is essential when building layouts.

Exploring the Field Picker

The *Field Picker*, shown in Figure 29-1, is a palette that integrates many field functions including the selection and placement of fields onto a layout. The visibility of the Field Picker can be toggled using any of the following methods while in Layout mode:

- Select the View > Field Picker menu.
- Click the Field Picker button in the second row of the toolbar.

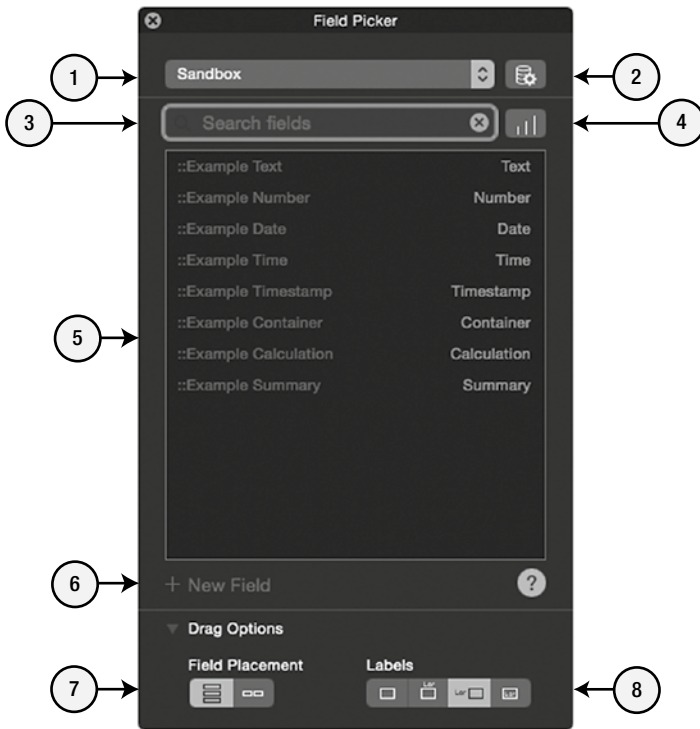


Figure 29-1. The Field Picker palette

The following controls are available on the Field Picker palette:

1. Table Occurrence — This pop-up menu contains a list of every table occurrence organized into two groups: those related to the current layout’s table and those not related. The default selection will always be the current layout’s table.
2. Manage Database — This button provides a shortcut to quickly open the Manage Database dialog (Chapters 7-9).
3. Search fields — When typing into this area, the list of fields below will be filtered to include only those whose name contains the text entered.
4. Sort fields — This icon contains a pop-up menu that offers the following sort options for the list of fields: Creation Order, Field Name, Field Type, and Custom Order.
5. Fields — This area lists the fields for the selected occurrence’s table, either every field, or those consistent with any filtering criteria entered. There is a deceptively large number of features integrated into this list. Double-click a field to edit its defined name. Click on the adjacent data type to reveal a hidden pop-up menu, used to change the defined format of the field. One or more fields can be selected and dragged onto the current layout where they will be added in a default size and format based on their data type and the Field Placement options selected at the bottom of the palette. Also, the field list’s contextual menu, contains the following two options:

- **Field Options** — Choose this item to directly open the Options for Field dialog for the selected field. This allows fast changes to field definitions.
 - **Delete Field** — Choose this item to permanently delete the field from the table after a warning dialog.
6. **New Field** — This button provides a shortcut for quickly creating a new field in the selected occurrence's table. The new field will appear in the list, highlighted so a name and data type can be quickly assigned before it is dragged onto the layout.
 7. **Field Placement** — These two buttons control whether fields dragged onto a layout will be placed side by side *horizontally* or stacked one above another *vertically*. The former is good for list layouts while the latter is better for form layouts.
 8. **Labels** — These four buttons control if a field label is automatically created when a field is dragged onto the layout and where that label is placed in relation to the field.

■ **Caution** The palette features that allow editing of a field's definition or the creation of new fields directly from the palette will *not* be available to a user with security access to edit layouts without access to define fields.

Exploring the Inspector

The *Inspector* is a palette packed full of configuration controls that are used to edit the format and behavior settings of a selected layout part or object. The options on the palette will dynamically enable, disable, and reconfigure to provide only the settings applicable to the object(s) selected.

The Inspector can be displayed or hidden by toggling the View > Inspector menu item. The Inspector palette will retain its last size and screen location when toggled on or off and even from one session to the next.

The controls on the Inspector are spread across the following four panels, each represented by an icon running along the top of the palette:

- Position
- Style
- Appearance
- Data

Each of these panels has various controls that are grouped into collapsible regions, which can be opened or closed in any combination.

■ **Tip** For fastest access to tools in the Inspector, open it to the maximum height with all the regions expanded.

It is possible to open multiple instances of the Inspector simultaneously and position them anywhere on the available display space. This makes it possible to have all four panels open and fully expanded to provide the fastest access to the variety of settings and tools within.

Panel One — Defining Position Settings

The *Position settings* are located on the first panel of the Inspector and are divided into five collapsible regions.

Position

The *Position* group, shown in Figure 29-2, contains options for naming and position.

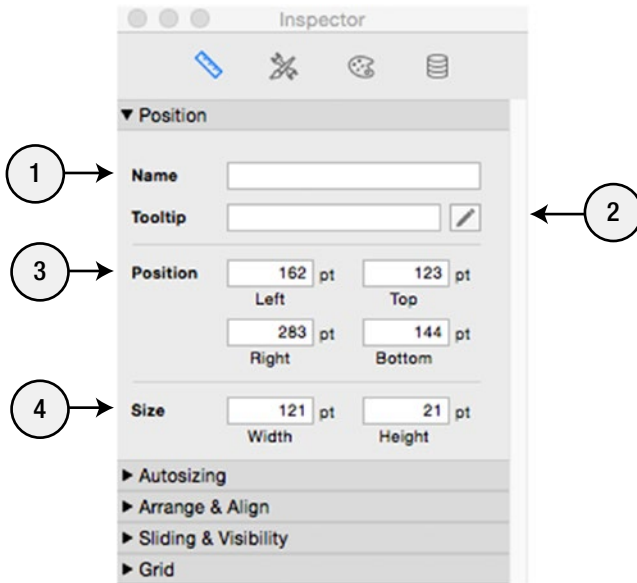


Figure 29-2. The *Position* subgroup of the *Position* panel of the *Inspector*

The following controls are available in this group:

1. **Name** — This text area allows the entry of an object name for the selected object (Chapter 31, “Naming Objects”).
2. **Tooltip** — This text area allows the entry of static or formula-generated text that will be displayed as a standard operating system style tooltip when the user positions the cursor over the object in Browse mode. The adjacent icon button will open a Specify Calculation dialog into which more complex formulas can be entered.
3. **Position** — These four text areas allow precise positioning of the object within the layout area by defining the *Left*, *Top*, *Right*, and *Bottom* values as an alternative to manually moving the object around the layout with the mouse. These numbers are locked in proportion, so changing one value will automatically adjust the opposing value to maintain the *Width* or *Height* values entered below.
4. **Size** — These two text areas allow precise sizing of the object within the layout area by defining a *Width* and *Height* value as an alternative to manually expanding or contracting the dimensions with the mouse.

■ **Tip** The position and size units will match the current ruler's settings but can be changed by clicking on the unit labels, which cycles through the options: centimeters (cm), points (pt), inches (in).

Autosizing

The *Autosizing* group, shown in Figure 29-3, contains anchoring controls that allow fine control over how an object moves or grows when a user resizes the window.

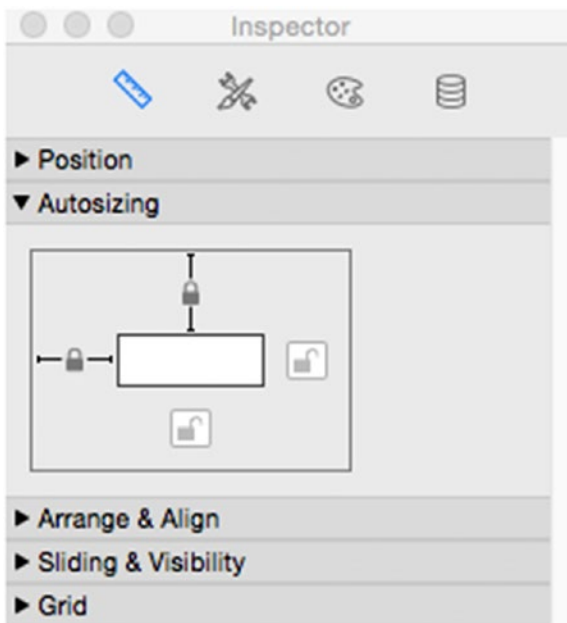


Figure 29-3. *The Autosizing subgroup of the Position panel of the Inspector*

The selected object(s) can be anchored to the Left, Top, Right, or Bottom of the layout by toggling the lock icon on or off on the respective side of the box. A lock on any side means that the distance between that side of the object and the corresponding edge of the layout will remain fixed as the window is resized while unlocked means that distance will vary. By using various combinations of locked and unlocked, an object can be made to either:

- Remain firmly in place at the same size by anchoring it to a corner such as locking Top and Left.
- Slide to the right or down as the window is resized by anchoring it just to the Bottom and/or Right sides.
- Grow in width and/or height as the window is resized by anchoring it to two opposing corners such as Left/Right or Top/Bottom.
- Float in the center of the window as it is resized by anchoring it to no side.

Arrange & Align

The *Arrange & Align* group, shown in Figure 29-4, contains numerous buttons that control object alignment, distributed space, relative size, group status, arrangement, and locked status.

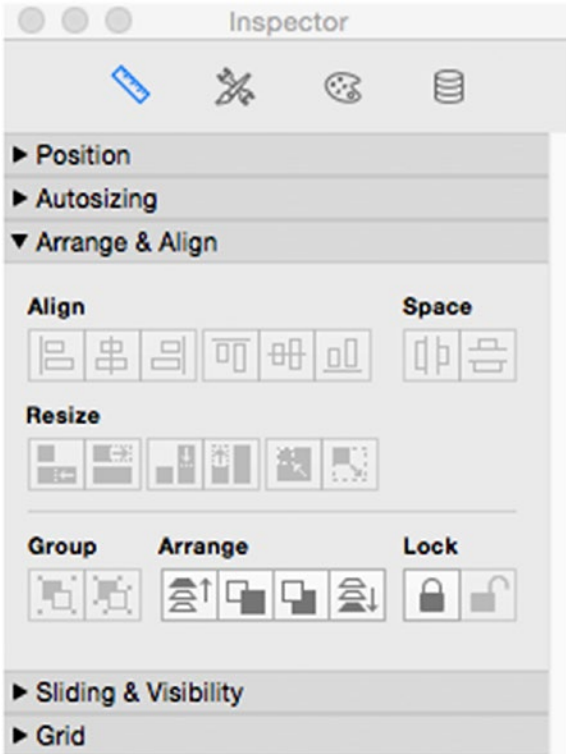


Figure 29-4. The Arrange & Align subgroup of the Position panel of the Inspector

The following controls are available in this group:

- **Align** — These six buttons allow quick alignment two or more selected objects by Left, Center, Right, Top, Middle, or Bottom.
- **Space** — These two buttons allow quick distribution by two or more selected objects evenly Horizontally or Vertically.
- **Resize** — These six buttons allow quick synchronization of the sizes of two or more selected objects to a uniform height and/or width based on the smallest or largest among them.
- **Group** — These two buttons allow you to quickly Group or Ungroup one or more selected objects.
- **Arrange** — These four buttons allow quick changes to the stacked arrangement of one or more selected objects in relation to other objects on the layout: Bring to Front, Bring Forward, Send Backward, Send to Back.
- **Lock** — These two buttons allow you to quickly Lock or Unlock one or more selected objects.

■ **Tip** Each of these settings are available in the Arrange menu and can therefore be invoked from the keyboard using a menu's corresponding Command key combination.

Sliding & Visibility

The *Sliding & Visibility* group, shown in Figure 29-5, contains several check box options that control how objects behave when printing a layout or viewing it in Preview mode.

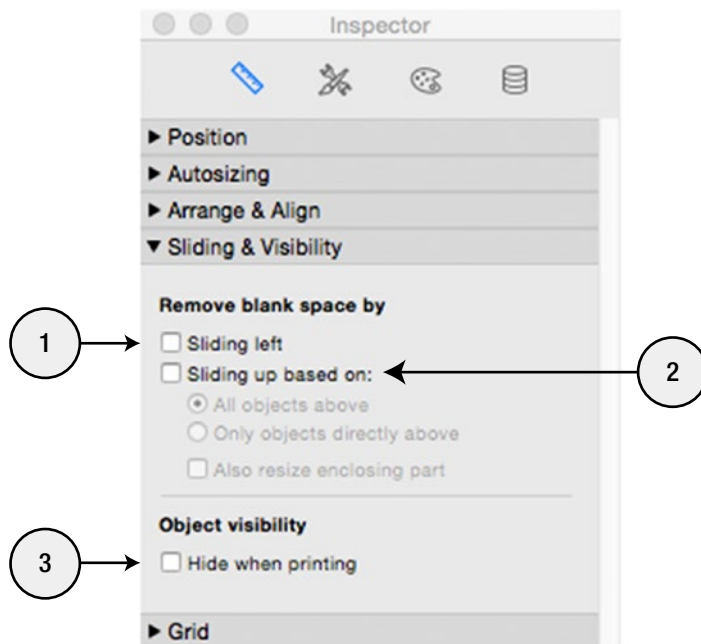


Figure 29-5. The *Sliding & Visibility* subgroup of the *Position* panel of the *Inspector*

The following controls are available in this group:

1. **Sliding left** — Enable this check box to cause the selected object(s) to shrink its width to the minimum required to display its content and to slide left as far as possible based on the other objects within the same horizontal plane.
2. **Sliding up based on** — Enable this check box to cause the selected object(s) to shrink its height to the minimum required to display its content and to slide up as far as possible based on the other objects in the same vertical plane. When this object is selected, other options below will become enabled:
 - a. **All objects above** — Choose this option to cause the selected object(s) in a common horizontal plane to slide up *together* based on the amount the objects above have been resized.

- b. **Only objects directly above** — Choose this option to cause the selected object(s) in a common horizontal plane to slide up independent of each other based on the amount the objects above have been resized.
 - c. **Also resize enclosing part** — Enable this check box to cause the layout part enclosing the selected object(s) to also shrink vertically based on the amount that the selected objects shrink.
3. **Hide when printing** — Enable this check box to cause the selected object(s) to be hidden when printing or previewing the layout.

Grid

The *layout grid* is a sequence of evenly spaced horizontal and vertical lines overlaid on a layout background that form a regular grid of minor and major areas, shown in Figure 29-6. This is only visible in Layout mode and can assist with precision manual placement and spacing of objects.

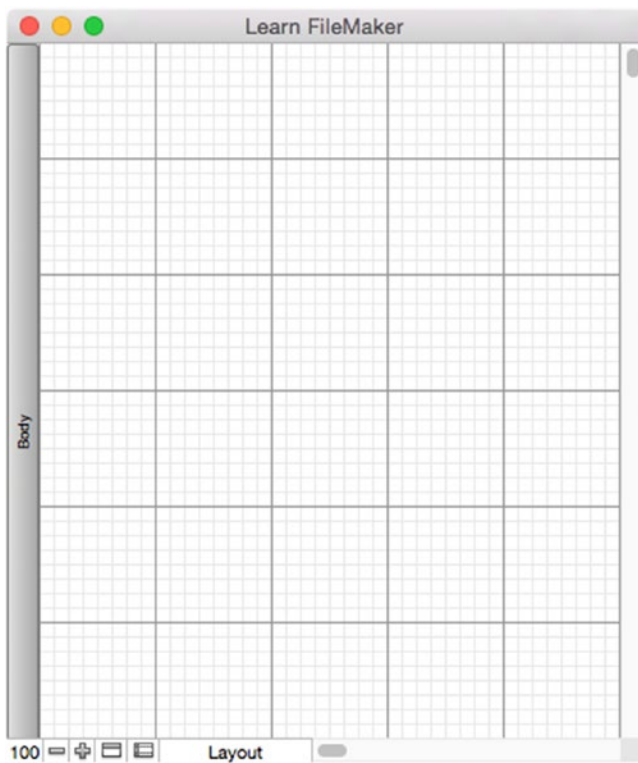


Figure 29-6. The grid visible on the background of a layout

The *Grid* group, shown in Figure 29-7, contains settings that allow control over the layout grid for the entire file.

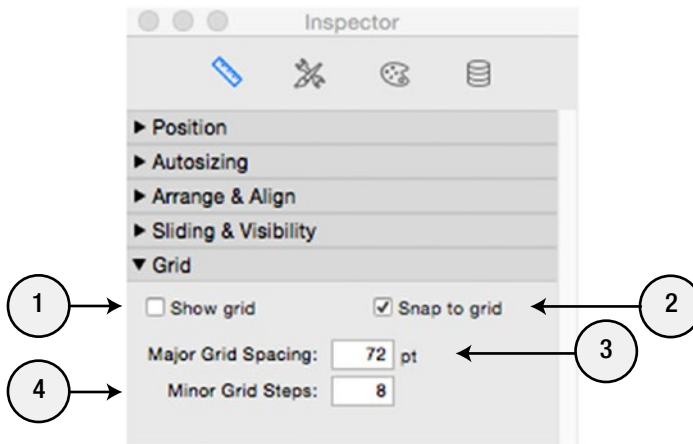


Figure 29-7. The *Grid* subgroup of the *Position* panel of the *Inspector*

The following controls are available in this group:

1. **Show grid** — Enable this check box to make the grid visible on all layouts within the file. This can also be toggled by selecting the `View > Grid > Show Grid` menu item.
2. **Snap to grid** — Enable this check box to cause layout objects to always fall on a grid lines as you drag them around a layout.
3. **Major Grid Spacing** — This text area contains the major grid spacing and can be overridden with a custom value.
4. **Minor Grid Steps** — This text area contains the number of minor steps within each major grid area and can be overridden with a custom value.

Panel Two — Style Settings

The *Style settings* are located on the second panel of the Inspector and shown in Figure 29-8.

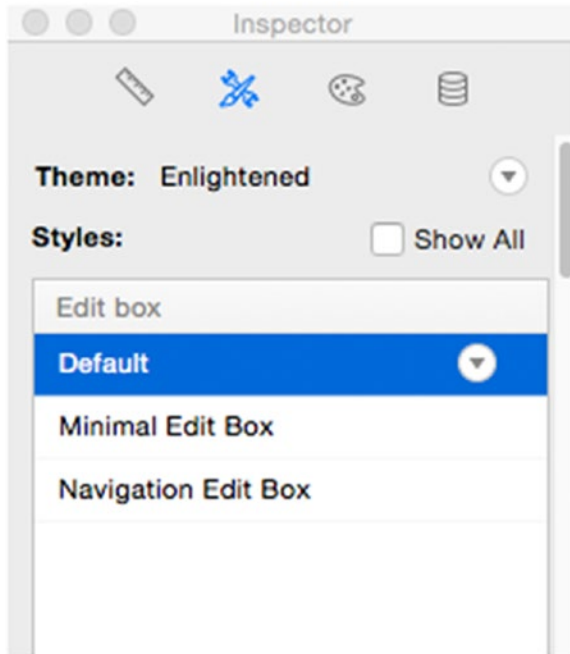


Figure 29-8. The Style panel of the Inspector

This panel provides information about the layout's assigned theme and allows saving theme changes. Depending on the state of the Show All check box, the list displays either a list of *all* defined styles organized by object type or *only* those applicable to the selected object type. The selected object's assigned style is always highlighted. Click on a style to apply it to the selected object.

■ **Note** For more information about Themes and Styles, see Chapter 32.

Panel Three — Appearance Settings

The *Appearance settings* are located on the third panel of the Inspector and are divided into six regional categories, all but one of which can be collapsed or expanded.

Theme and Style

The *Theme and Style* group, shown in Figure 29-9, is non-collapsible and displays information about the theme and style of the layout and selected object(s).

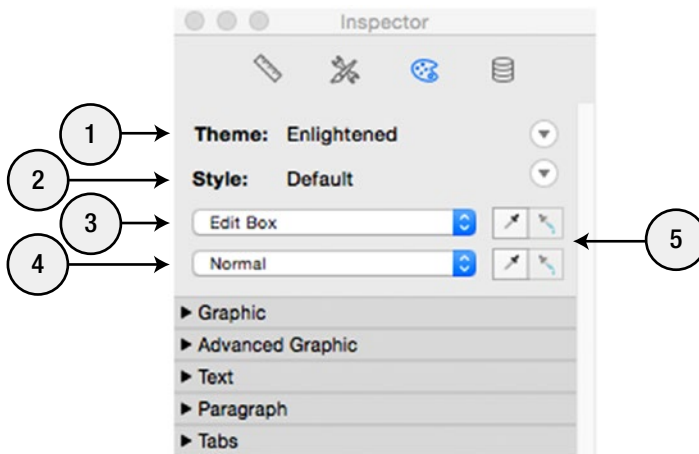


Figure 29-9. The Theme and Style subgroup of the Appearance panel of the Inspector

The following controls are available in this group:

1. Theme — This area displays the name of the theme assigned to the current layout. The adjacent icon hides a pop-up menu that allows saving, renaming, or reverting of theme changes.
2. Style — This area displays the name of the style of the currently selected object(s). The adjacent icon hides a pop-up menu that allows saving, renaming, or reverting of style changes.
3. Object Type — This pop-up menu automatically displays the object type of the currently selected object(s). For object types that are made up of a hierarchy of components, this list will contain all available component types for the select object type. For example, when a `portal` is selected, the list will include `Portal` and `Portal:Row`, since each of these can have different appearance formatting.
4. Object State — This pop-up menu displays a list of available object states for the selected object. Each state can have different format settings so the selection from this list indicates which state is being edited with the controls in the rest of the Appearance panel. While some objects have additional states available, an *object state* is generally one of four possible conditions with respect to the user's interaction with the object:
 - a. Normal — The state of the object when it is displayed on a layout without active interaction.
 - b. Hover — The state of the object when the user places the cursor over it without clicking.
 - c. Pressed — The state of the object when the user has pressed the mouse button onto the object and not yet released it.
 - d. In focus — The state of the object when it is active. For example, a field is considered in focus when the cursor is within it as the user is entering data.

5. Copy Paste Style — These two sets of buttons allow the selected object type or state styling to be copied and then pasted onto other objects.

■ **Note** For more information about an object's Theme, Style, Type, and State, see Chapter 32.

Graphic

The *Graphic* group, shown in Figure 29-10, contains controls over the graphical settings of the chosen state of the selected layout object.

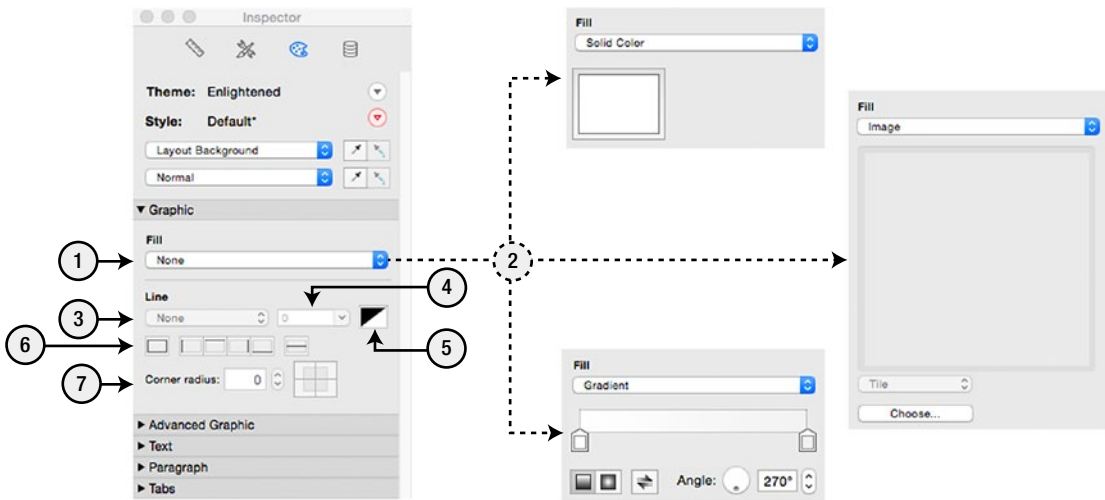


Figure 29-10. The *Graphic* subgroup of the *Appearance* panel of the *Inspector*

The following controls are available in this group:

1. Fill Type — This pop-up menu contains four options for the fill of an object:
 - a. None — The object will be transparent.
 - b. Solid Color — The object can be assigned a color by selecting it from the color well in the Fill Settings area below the menu.
 - c. Gradient — The object can be assigned gradient settings in the Fill Settings area below the menu. Two or more gradient color stops can be defined to establish the range and variety of color. Buttons allow for the selection of gradient, linear, or radial; the direction can be reversed and the angle customized.
 - d. Image — The object can be assigned an image that will be displayed as its background fill. The pop-up menu allows selection of how the image is handled with the following options: Original Size, Scale to Fit, Scale to Fill, Slice and Tile.

2. **Fill Settings** — This area changes to present settings specific to the selected Fill Type.
3. **Line Style** — This pop-up menu offers a choice of line style for the selected object's border: None, Solid, Dashed, and Dotted.
4. **Line Thickness** — This combination text area and drop-down menu allows the entry or selection of a number representing the thickness of the line.
5. **Line Color** — Clicking this color well allows the selection of a line color from a palette.
6. **Line Border** — These icon buttons allow toggling of specific line border settings: All borders, Left, Top, Right, Bottom, and between field repetitions of repeating fields.
7. **Line Corner Radius** — This combination of text area and four icon buttons allows control over the radius of corners. Enter the percentage of desired rounding on the object corners into the text area and click buttons that correspond with the corner(s) to which this will apply.

Advanced Graphic

The *Advanced Graphic* group, shown in Figure 29-11, contains controls over the advanced graphical settings of the chosen state of the selected layout object.

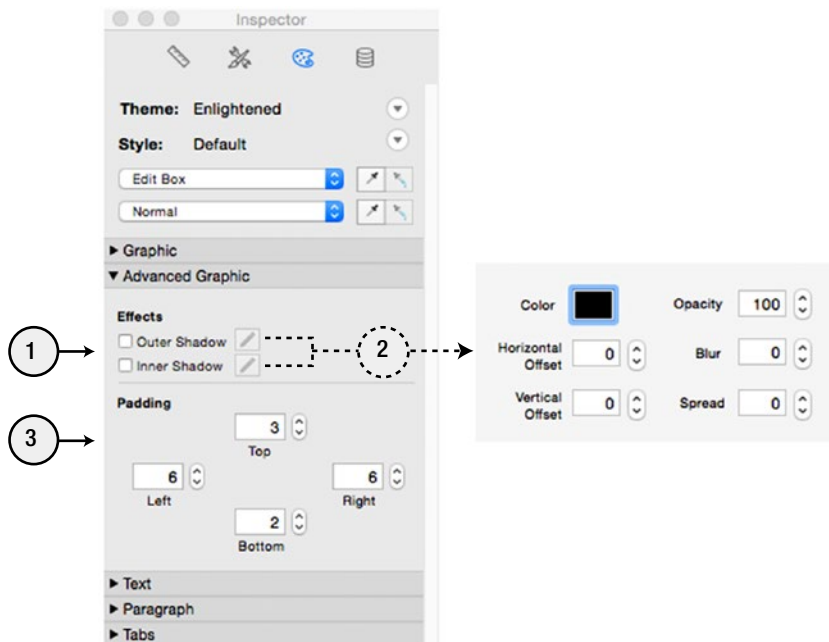


Figure 29-11. The *Advanced Graphic* subgroup of the *Appearance* panel of the *Inspector*

The following controls are available in this group:

1. **Effects** — Two check boxes allow the application of an outer and inner shadow effect to the selected objects. Once selected, the adjacent icon button becomes enabled allowing specific configuration of the effect settings.
2. **Effect Settings** — Each button opens a pop-up panel with the following settings for the effect:
 - a. **Color** — The color well shows the selected color of the effect. Clicking it will open a color palette from which a new color can be assigned.
 - b. **Horizontal Offset** — The number within this text area dictates how far the shadow effect extends to the right of the object (**Outer Shadow**) or in from the left side of the object (**Inner Shadow**).
 - c. **Vertical Offset** — The number within this text area dictates how far the shadow effect extends beneath the object (**Outer Shadow**) or in from the top of the object (**Inner Shadow**).
 - d. **Opacity** — The number within this text area indicates the percentage of opacity of the shadow effect.
 - e. **Blur** — This number indicates the amount of blur applied to the shadow effect.
 - f. **Spread** — This number indicates the amount of spread applied to the shadow effect. The *spread* is the amount of height and width applied to the shadow beyond the size of the original object.
3. **Padding** — An object's *padding* is the amount of distance between the border of the object and the content contained within it. The four text areas allow specific distance to be specified between the content and the **Top**, **Left**, **Bottom**, and **Right** sides of the object. These settings may compete with those of **Indentation** in the **Paragraph** group and may cause confusion if used in simultaneously.

Text

The *Text* group, shown in Figure 29-12, contains controls of the text and style settings of the chosen state of the selected layout object.

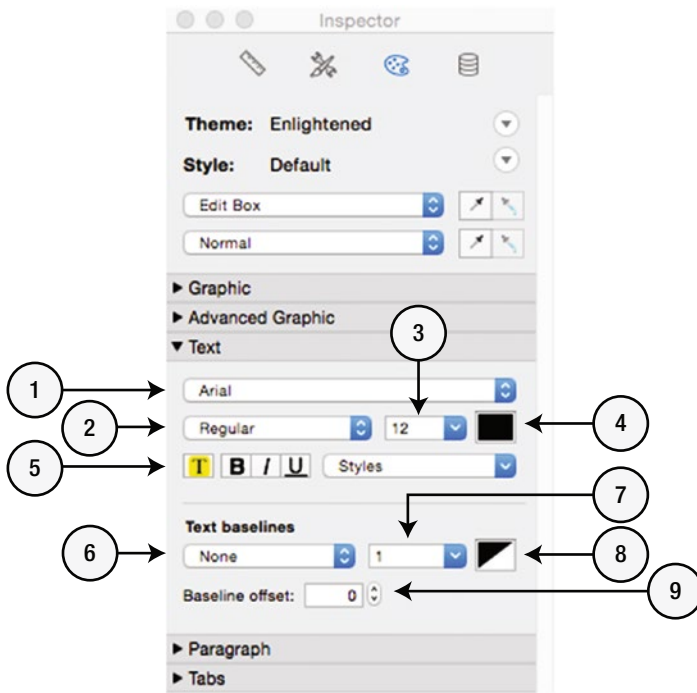


Figure 29-12. The Text subgroup of the Appearance panel of the Inspector

The following controls are available in this group:

1. **Font Family** — This pop-up menu lists every font installed on the local computer and is used to change the font assigned to the selected object. FileMaker automatically summarizes fonts into families where possible and lists the available styles in the **Font Style** pop-up menu below.
2. **Font Style** — If the selected **Font Family** contains multiple styles, those are available in this pop-up menu.
3. **Font size** — This combination text area and pop-up menu allows the entry or selection of a size for the selected object's text.
4. **Text color** — This opens a text color selection palette.
5. **Text style** — This row of buttons and adjacent pop-up menu are used to apply text styles to the selected object. Between the two, the available options are the following: Highlight, Bold, Italic, Underline, Plain, Word Underline, Double Underline, Strike Thru, Condense, Extend, Small Caps, Title Case, Uppercase, Lowercase, Superscript, and Subscript. When a selected **Font Family** contains style options that FileMaker automatically separates into the **Font Style** pop-up menu, the interface elements on this row will synchronize with that menu.
6. **Text baseline** — This pop-up menu allows the selection of one of the following baseline types for the text within a field: None, Solid, Dashed, or Dotted.

- 7. **Baseline thickness** — This combination text area and drop-down menu allows the entry or selection of a number representing the thickness of the text baseline for the selection field.
- 8. **Baseline color** — The color well shows the selected color of the text baseline. Clicking it will open a color palette from which a new color can be assigned.
- 9. **Baseline offset** — This text area contains a number indicating the amount of offset for the baseline in the selected field.

Paragraph

The *Paragraph* group, shown in Figure 29-13, contains controls over the paragraph alignment and spacing settings of an interface state of the selected element.

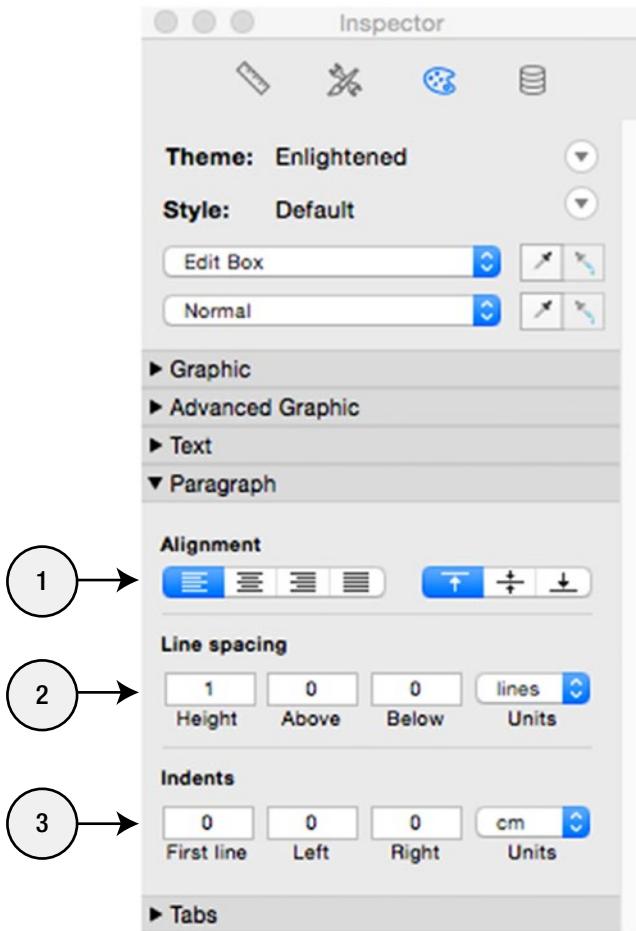


Figure 29-13. The Paragraph subgroup of the Appearance panel of the Inspector

The following controls are available in this group:

1. **Alignment** — This row of buttons control the horizontal and vertical alignment of text within the object. The horizontal settings are Left, Center, Right, Full Justification. The vertical settings are Top, Middle, Bottom.
2. **Line spacing** — This row of settings allows adjustment to the line spacing of the text of the object. You can adjust the number for Height, space Above and Below, as well as select the type of Units from the pop-up menu, choosing between the default lines and a unit of measure: centimeters, inches, or points.
3. **Indents** — This row of settings allows adjustment to the indentation of the text in the object. You can adjust the number for indenting the First line, Left, or Right as well as select a preferred unit of measure: centimeters, inches, or points.

■ **Caution** The Line Spacing and Indents settings may compete with those of Padding in the Advanced Graphic group and may cause confusion if used simultaneously.

Tabs

The *Tabs* group, shown in Figure 29-14, provides control over the tab settings of a selected object.

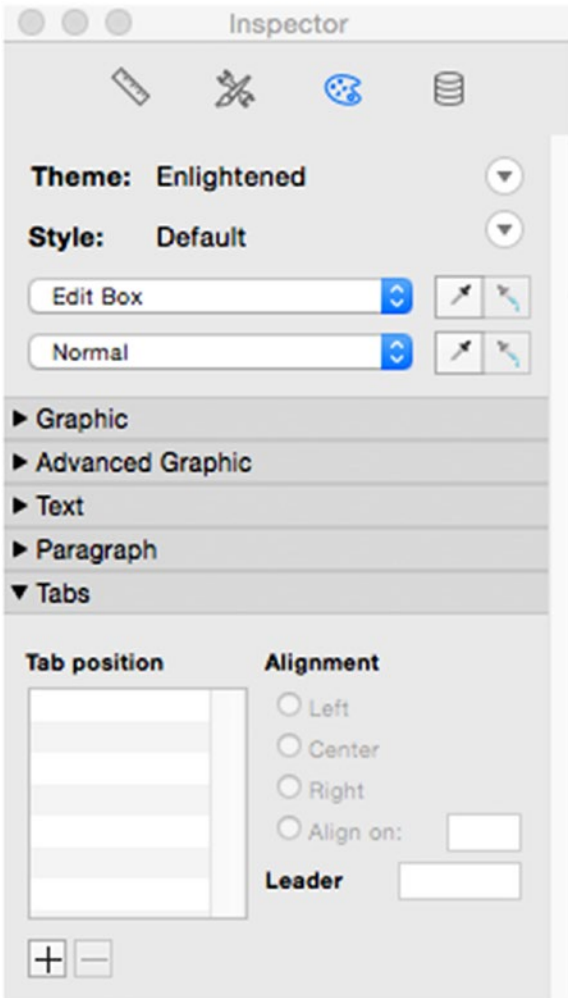


Figure 29-14. The *Tabs* subgroup of the *Appearance* panel of the *Inspector*

Click the + or - buttons to add or remove *Tab position* from the list. Click on a *Tab position* to edit the measurement and to choose an *Alignment* and *Leader* character.

Panel Four — Data

The *Data settings* are located on the fourth and final panel of the *Inspector* and are divided into three collapsible regions.

Field

The *Field* group, shown in Figure 29-15, contains controls over the settings of a selected field.

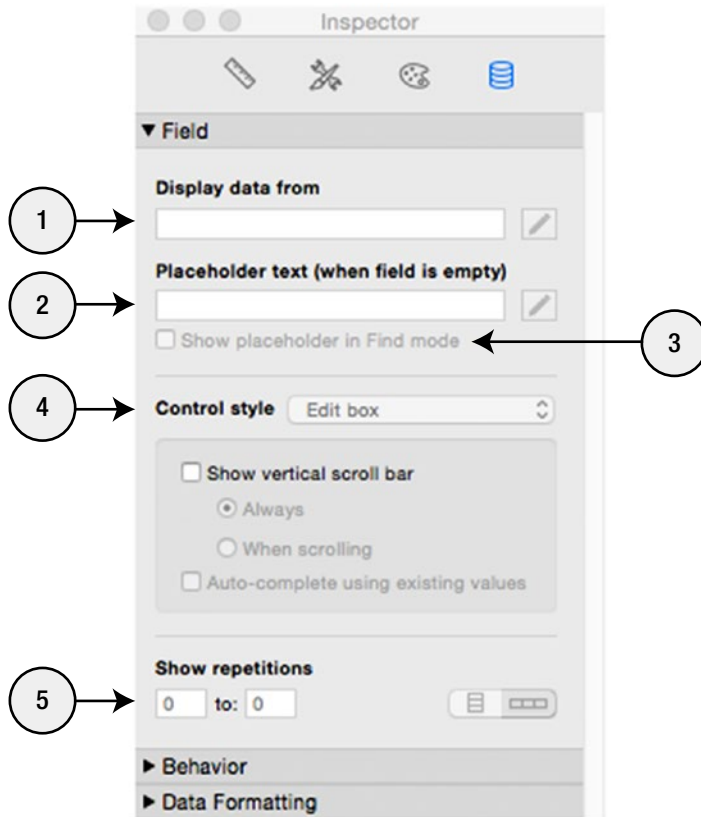


Figure 29-15. The *Field* group of the *Data* panel of the *Inspector*

The following field controls are available in this group:

1. **Display data from** — This text area controls which table field is assigned to a selected field object on the layout. The adjacent button allows reassignment of the field object to a different field through a *Specify Field* dialog.
2. **Placeholder text** — This text area contains static text or a formula that evaluates to text, which will be displayed in a field when it is empty. This can replace field labels or be an additional helpful hint about the field's intended content.
3. **Show placeholder in Find mode** — This check box's state determines if a field's placeholder text is displayed in Find mode or not. Alternatively, the actual formula can contain conditional clauses that evaluate to a different result depending on the current window mode.

4. **Control style** — This pop-up menu selects a field control style for the selected field. The settings in the shaded area below it will vary depending on the style selected. Control styles include:
 - **Edit box** — The field will be formatted as a standard field area, with or without a scroll bar, into which a user can click to begin data entry.
 - **Drop-down list** — The field will be formatted as an edit field with an optional attached value list, allowing a user to select or type a value. Depending on the settings, the list will appear instantly when the user enters the field or when they click an optional attached arrow icon. It can be configured to present potential matching values as they begin typing, which can auto-complete the entry.
 - **Pop-up menu** — The field will take the form of a pop-up menu, displaying a value list. Users can choose or paste in a value but can't type directly into the field.
 - **Check box set** — The field will display value list items as a set of check boxes. Users can select or unselect values with mouse clicks or by pasting. The field will contain a carriage-return delimited list of selected or entered values but will only display on a layout check boxes for those defined in the assigned value list.
 - **Radio button set** — The field will display a value list as a set of radio buttons allowing a single value selection.
 - **Drop-down calendar** — The date field will be formatted as an edit field into which the user can type a date with an optional drop-down graphical calendar to facilitate faster data entry.
 - **Concealed edit box** — The field will be formatted as a standard field area but all data typed will be obscured with bullets (but not encrypted).
5. **Show repetitions** — These settings allow control over which repetitions of a field are displayed on the layout and the orientation in which they are drawn.

■ **Note** Control Style options are discussed in more detail in [Chapter 30](#).

Behavior

The *Behavior* group, shown in Figure 29-16, contains controls that affect various object behaviors on the layout such as visibility, mode accessibility, and other features.

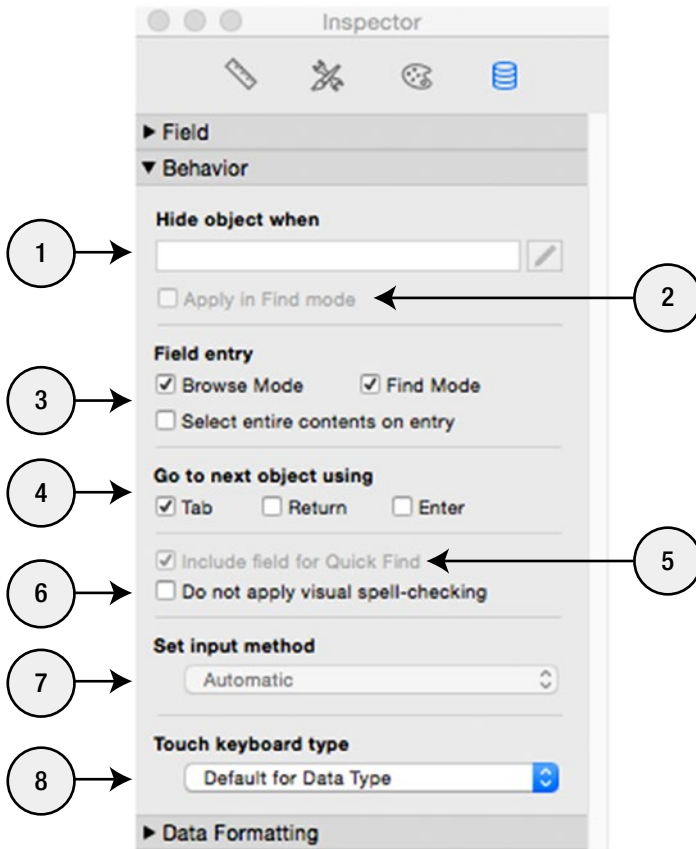


Figure 29-16. The *Behavior* group of the *Data* panel of the *Inspector*

The following field controls are available in this group:

1. **Hide object when** — This text area contains a calculation formula that, when evaluated to a result of true, will cause the object to be hidden in any non-Layout mode. A formula can be typed or pasted directly into the field, or the adjacent button can be used to open a *Specify Calculation* dialog for creating more complex formulas.
2. **Apply in Find mode** — Enable this check box to apply the *Hide object when* formula in Find mode. Alternatively, the actual formula can contain conditional clauses that evaluate to a different result depending on the current window mode.

3. **Field entry** — These check boxes control when a user can click into the selected field or not and what happens when they do:
 - a. **Browse mode** — Enable this check box to allow a user to click into the field in Browse mode.
 - b. **Find mode** — Enable this check box to allow a user to click into the field in Find mode.
 - c. **Select entire contents on entry** — Enable this check box to automatically select the entire contents of the field when a user clicks into it. This is useful when a value is likely to change completely every time a user makes an edit.
4. **Go to next object using** — These check boxes control if the Tab, Return, and Enter keys jump to the next field or are entered into the field as literal text. By default, FileMaker uses a Tab keystroke to jump between fields and an Enter keystroke to commit a record. These check boxes allow that behavior to be overridden.
5. **Include field for Quick Find** — Enable this checkbox to allow the selected field to be included in Quick Find feature (Chapter 4; Searching with Quick Find).
6. **Do not apply visual spell checking** — Enable this check box to exclude the selected field from visual spell checking.
7. **Select input method** — This pop-up menu contains options for choosing a specific input language method for the selected field in situations where certain foreign languages are used such as Japanese.
8. **Touch keyboard type** — This pop-up menu contains a choice over which keyboard type will be available to the user when they click into the selected field on an iOS device. For layouts that will be used on such devices, this allows a specific keyboard to be specified that relates to the field's data type (text, number) or its intended content (email, phone, URL).

Data Formatting

The *Data Formatting* group provides control over the formatting overlay based on a selected field's specific data type. When a field has editing focus, it will show the information as entered. However, these settings allow a field to display that information with special formatting characteristics depending on its type.

The top portion of the group, shown in Figure 29-17, displays an icon indicating the data type of the selected field: decimal, date, time, or container. The Format pop-up menu contains the formatting options based on that data type and the area beneath this contains controls that vary by the selection made within this menu.

■ **Caution** When exporting records, the actual value in the field will be exported instead of the formatting imposed by these settings, unless the Apply current layout's data formatting to exported data check box is selected (Chapter 5).

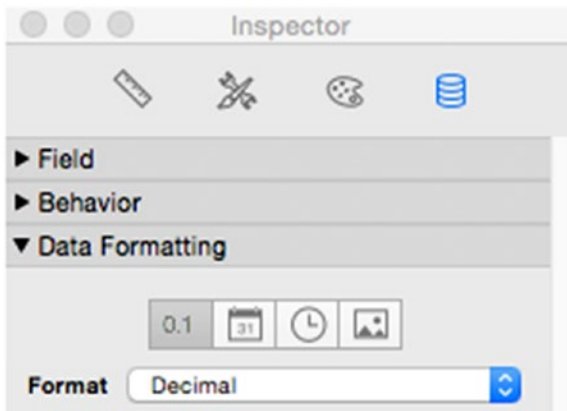


Figure 29-17. The Data Formatting group of the Data panel of the Inspector

Data Formatting — Decimal

When a number field is selected, the Format pop-up menu contains the following options, each subsequently displaying contextual controls beneath:

- **General** — This choice will display the numeric value as it was entered with no formatting options available. The value will be rounded or expressed in scientific notation as needed to fit within the boundaries of a field on a layout.
- **As entered** — This choice will display the numeric value exactly as it was entered with no formatting options available. If the value contains more digits than can be displayed in the field on a layout, it will display a question mark instead.
- **Boolean** — This choice allows the entry of a value that should replace zero and non-zero values. The default is Yes and No, but these can be replaced with True or False or any other values.
- **Decimal** — This choice presents options for how the number should be formatted, including number of decimals, notation, negative value formatting, choice of separators, and more. For example, using this option, entering a numeric value of 1003.7568 into a field, could be automatically displayed as 1,003.75 or #1003 depending on the selected settings.
- **Currency** — This choice presents options like Decimal for how the number should be formatted as currency. For example, using this option, entering a numeric value of 5.75 into a field, could be automatically displayed as \$5.75, \$5, £5.00, etc., depending on the selected settings.
- **Percent** — This choice presents options like Decimal for how the number should be formatted as a percentage. For example, using this option, entering a numeric value of .62 into a field, could be automatically displayed as 62% or 62.00% depending on the selected settings.

Data Formatting — Date

When a date field is selected, the Format pop-up menu contains a variety of literal preconfigured date formats and an option to enter a custom format using the controls shown in [Figure 29-18](#).

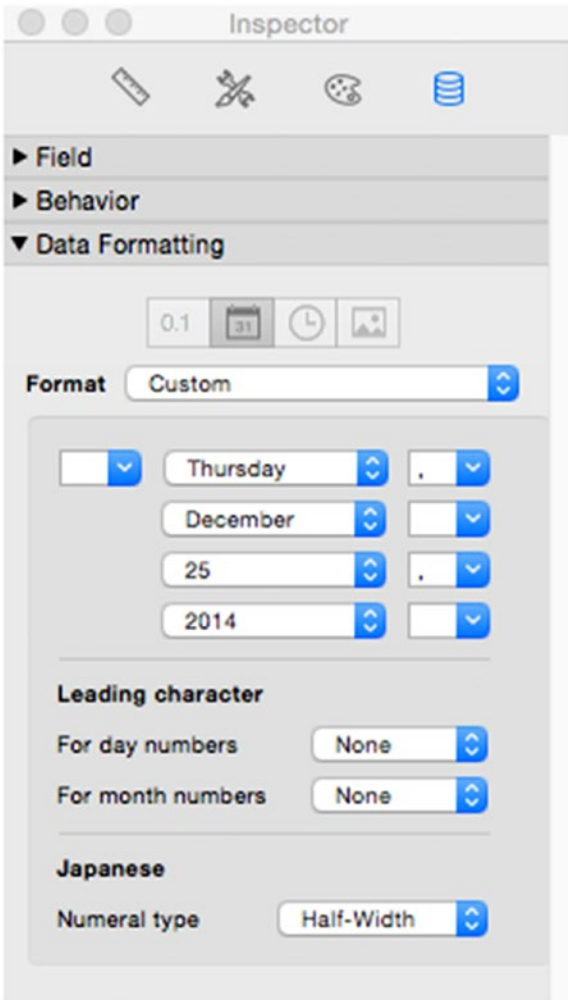


Figure 29-18. The custom date options for the Data Formatting group of the Data panel of the Inspector

The Custom date format presents a flexible modular set of controls that allow the selection of any date component arranged in any order with any delimiter placed between them and with leading zero options for day and month numbers. Using these controls, a date entered as 1/15/2017 can be displayed in any number of different combinations including:

- January 15, 2017
- 01.15.2017
- 01.15.17
- 15 Jan 2017
- Sunday Jan 15, 2017
- 01/15/2017 (Thursday)

Data Formatting — Time

When a time field is selected, the Format pop-up menu contains a variety of literal preconfigured time formats and other optional control over separate, military vs. civilian time, leading zeros for hours and minutes, and more.

Data Formatting — Container

When a container field is selected, the Format pop-up menu contains the following options that control how an image should resize for display purposes only (the actual file contents will not be affected):

- **Crop to frame** — Choosing this option will leave the image sized as entered, cropping it to fit the frame if it exceeds the space available.
- **Reduce image to fit** — Choosing this option will cause the displayed image to be reduced until it fits within the frame.
- **Enlarge image to fit** — Choosing this option will cause the displayed image to be enlarged so that it fits within the frame.
- **Reduce or enlarge image to fit** — Choosing this option ensure that the displayed image fits exactly within the frame, reducing or enlarging it as needed.

Many settings are available in the area below the Format menu, shown in [Figure 29-19](#).

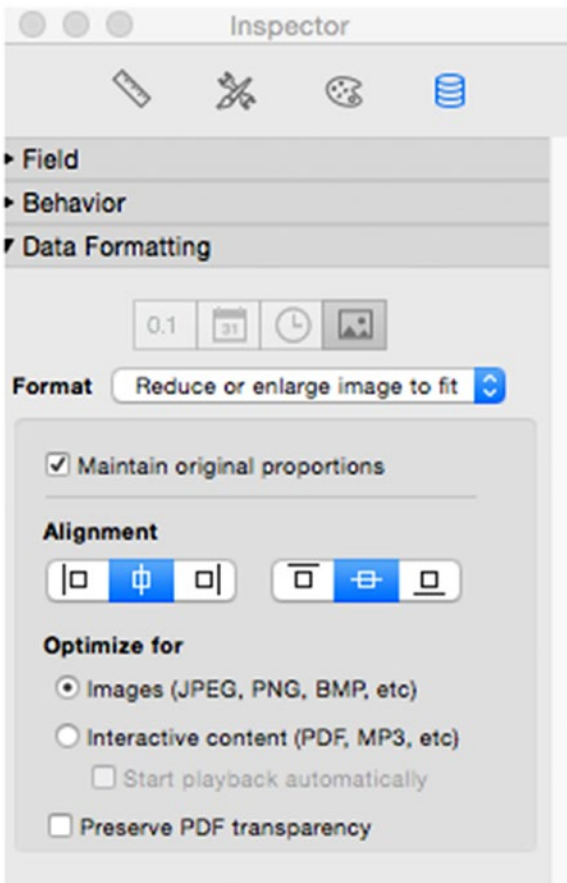


Figure 29-19. The container options for the Data Formatting group of the Data panel of the Inspector

These settings include:

1. **Maintain original proportions** — Check this box to keep the image’s aspect ratio fixed when it is reduced or enlarged for display within the selected field.
2. **Alignment** — These button icons allow control over how an image will be aligned within the selected field.
3. **Optimize for** — These options control how the container contents are displayed, with options including:
 - a. **Images** — Optimize content for images, causing an image file in the container to be displayed as an image rather than a file.
 - b. **Interactive content** — Optimize content for PDF, MP3, and movie files. Users will be able to view and interact with these types of media as if they were viewing them in an application optimized for manipulating them. Checking the start playback automatically will cause an audio or video file contained in the field to begin playing as soon as a record displaying it on a layout is loaded.

Exploring the Layout Objects Palette

The Layout Objects palette, shown in Figure 29-20, is used to locate, select, reorder, and hide layout objects. The palette can be opened by selecting the View > Layout Objects menu item and will remain open automatically any time you return to Layout mode unless you explicitly close it.

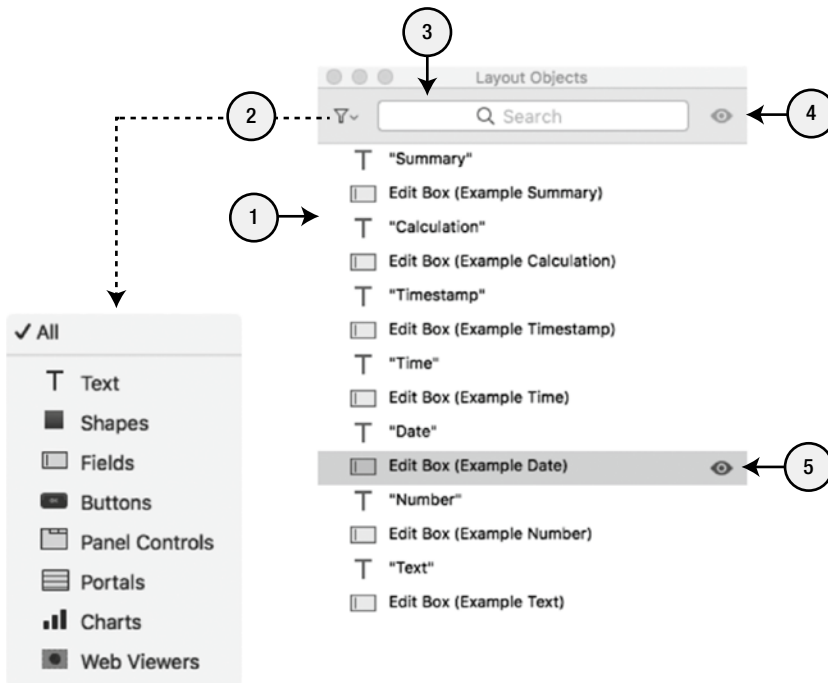


Figure 29-20. *The Layout Objects palette*

The Layout Objects palette contains the following controls:

1. **Objects List** — The palette lists every object on the current layout. Each object will be displayed with a generic name based on the object type and/or field name unless a custom name has been assigned. Items in the list can be selected to perform many different functions:
 - Selecting an object in the list selects the corresponding object on the layout. Conversely, selecting on object on the layout while the palette is open, selects the object there as well.
 - Multi-layered objects such as portals, tab controls, and slide controls can be toggled to show or hide the nested object structure they contain.

- Dragging an object up or down in the list performs a function like the arrangement features, bringing the corresponding object forward or backward on the layout. Reordering only works when viewing all objects on the layout in a non-filtered list.
 - Objects can be temporarily hidden on the layout to declutter the view when editing complex layered objects. This hides the object when viewed in Layout mode only and does not hide the object in Browse mode.
2. *Search Filter Menu* — Clicking the icon at the top of the palette will open a menu of layout types. Select a type to filter the list in the palette to include only objects of the selected type.
 3. *Search Filter* — The list of objects can be filtered by typing search criteria into this text area.
 4. *Hide Group* — This button becomes enabled when one or more objects are hidden on the layout. Click it to restore all hidden objects to their normal visible state.
 5. *Hide Object* — This button will toggle the selected object(s) hidden status.

■ **Tip** For faster selection and editing of objects, open the Layout Objects palette and Inspector(s) side by side. Select an object from the Layout Objects palette and edit its settings in the Inspector.

Using the Shortcuts in the Layout Objects Palette Contextual Menu

The palette's contextual menu contains several convenient options and shortcuts, as shown in Figure 29-21.

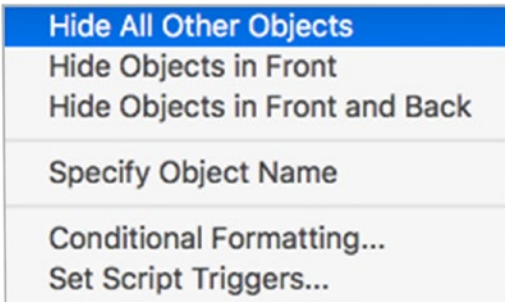


Figure 29-21. The contextual menu for an item within the Layout Objects palette

The following commands are available in the contextual menu depending on the type of object selected:

- **Hide All Other Objects** — This will hide every object on the layout except for the object(s) currently selected in the Layout Objects palette. This provides a quick way to isolate the selected object(s) and remove all other distractions from the layout.
- **Hide Objects in Front** — This will hide every object that is layered in front of the selected object(s). This allows you to quickly see an object that is hidden behind other overlapping elements.
- **Hide Objects in Front and Back**— This will hide every object that is layered in front of and behind the selected object(s).
- **Specify Object Name** — By default object names will appear in the palette generically by object type and/or field name. This command reveals an edit box in place of this generic name into which you can enter a custom name for the object, as shown in Figure 29-22. Object names are not widely used in FileMaker but are important in some instances such as when using the Go to Object script step, which requires an object name as an identifier rather than a field name or other reference.

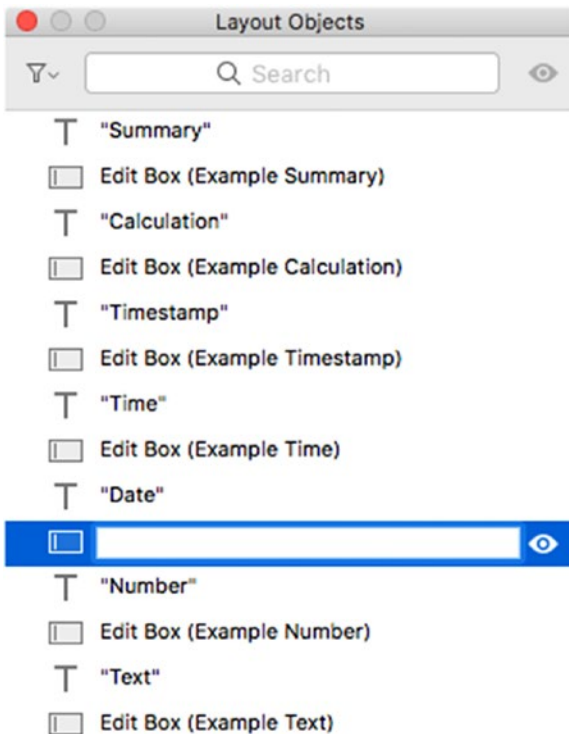


Figure 29-22. The Edit Object Name text area

- **Conditional Formatting** — This item provides a shortcut to open the Conditional Formatting dialog for the selected object(s). For more on conditional formatting, see Chapter 31.
- **Set Script Triggers** — This item provides a shortcut to open the Set Script Triggers dialog for the selected object(s). For more on script triggers, see Chapter 37.

■ **Tip** If you use the Layout Objects palette a lot, assigning object names will greatly improve your efficiency.

Summary

In this chapter, we introduced the Layout mode palettes and explored their various tools and settings.

CHAPTER 30



Creating Layout Objects

Over the last three chapters, we introduced Layout mode, layout parts, managing layouts, and the palettes available for working with objects. Now, we can begin creating layout objects.

This chapter discusses each type of object that can be placed on a layout, including:

- Introducing layout objects
- Working with field objects
- Working with text
- Working with button controls
- Working with panel controls
- Working with portals
- Working with web viewers
- Working with charts

■ **Note** The illustrations of layout objects pictured in this chapter are designed with default formatting. However, as indicated previously, FileMaker's design tools allow for a much more professional design than is shown in these simple examples.

Introducing Layout Objects

A *layout object* is an interface element used to display information, allow user entry, or provide access to a script function. The basic layout object types are:

- *Fields* — are used to display a field from the current record in the layout's table or from tables related to that table, each displayed as either editable or read-only.
- *Text* — can be static or dynamic, and help to communicate information and to identify other objects.

- *Buttons* — make script functions accessible to users and hide expandable interface panels.
- *Panels* — make efficient use of space by alternating between different groups of objects within the same space when the user clicks a tab or swipes.
- *Portals* — can display a limited set of fields in a list of records from another table that are related to the record being viewed in the window.
- *Web Viewers* — display web pages, directly from the Internet or dynamically generated by a formula using information from the database.
- *Charts* — display data graphically with charts and graphs.

Inserting an Object onto the Layout

Generally, a new instance of any available object type can be inserted onto a layout, using the following methods:

- Click the toolbar icon corresponding to the type of object, then click and drag anywhere on the layout to define the outline of the object. When you release the cursor, the object will appear in the size and position indicated.
- Select the desired object type from the corresponding Insert menu item to quickly place a new instance at a default size and position on the layout.
- Duplicate an existing object to a new instance by selecting it and choosing the Edit > Duplicate menu item.
- Use the Copy and Paste functions to create a new copy of an existing object on the same or on a different layout.

■ **Note** Some object types have alternative insertion methods that will be noted throughout the chapter.

Configuring an Object

Once a new object instance is inserted on a layout, it will be in a raw state ready to be formatted and configured. Depending on the type, an object can be configured using some or all of the following methods:

- Double-clicking on most object types will open a configuration dialog, usually focused on data configuration options rather than formatting or other behavioral control.
- When an object is selected:
 - the Format and Arrange menus contain many styling and positioning options.
 - the Inspector contains many configuration options (Chapter 29).

Working with Field Objects

The primary purpose of most a layout is to allow users to view and interact with data stored in fields. Therefore, fields are likely to be the most common object on most layouts and they have the most varied set of configuration options.

This section covers many aspects of using and configuring fields, including:

- Clarifying field terminology
- Field basics
- Choosing a field control style
- Exploring the benefits of pop-up menus for two-field value lists
- Bypassing value list entry restrictions
- Using placeholders

Clarifying Field Terminology

Within the context of a discussion about layouts, it is important to first clarify the often-blurred difference between the following four uses of the term, each illustrated in Figure 30-1:

- A table's field definition
- An instance of a field for a record
- A field on a layout being designed
- A field rendered on a layout being viewed by a user

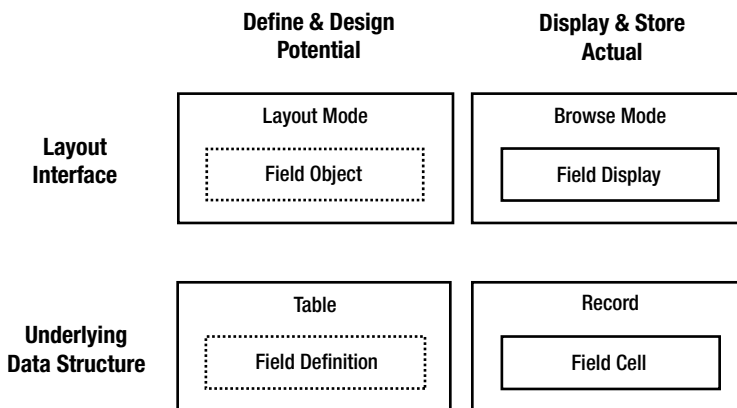


Figure 30-1. An illustration of the various uses of the term “field”

These are all generally referred to as a “field” and, while quite common and acceptable, it might be confusing to newcomers.

Elaborating on the material discussed in previous chapters, a *field definition* for a table is an individual unit of storage *potential* that establishes the type of data and certain informational behaviors of that data. These will be replicated as a *field cell* for every record created within that table, making an *actual* unit of storage that will contain information entered by a user or automatically by a calculation or script process.

■ **Tip** Remember the spreadsheet metaphor: a field definition is like a column; a record is like a row; and an instance of a field for a record is like a cell, the intersection of a column at a row.

Because a database user always interacts with the data structure *through* a layout, the metaphorical spreadsheet described above is completely hidden from view when thinking of the interface. When designing a layout, a *field object* is a graphical element that defines the position, behavior, and appearance of where and how a referenced field cell should be displayed. In Layout mode, the field object is assigned a field reference, pointing to a field definition. When viewed in Browse mode, the layout object is rendered as a *field display* showing information stored in the field cell for the record currently viewed: formatted and behaving in accordance with the settings of both the field definition and the layout field object.

It isn't crucial that one remembers all the terminological details here. However, when thinking or reading about something referred to as a "field," just remember that the usage context should be kept in mind to avoid confusion.

Field Basics

Working with field objects requires a minimum understanding of a few basics.

Adding Fields to a Layout

More than any other object type, a field object can be added to a layout by one of *numerous* methods. Some will place the object and then ask which defined field the field object should represent. Others don't ask because a field must be specified as part of the process of adding the object. Finally, one method varies in this regard depending on the specifics. Let's work through all the methods for adding a field object to a layout.

Specifying the Field After Adding the Object

Any of these methods will add an empty field object to the layout and then present a dialog asking for a field selection:

- Drag the Field tool from the toolbar down onto the layout.
- Use the Field/Control toolbar menu to set the cursor into a mode that allows you to draw a field by clicking and holding the cursor on the layout and dragging down and to the right, defining the area for the field before releasing to create the field. This tool can be used one of two ways:
 - Click once on the icon to auto-select the Edit box control type for the new field.
 - Click and hold to activate the hidden menu from which you can select a field control type for the new field.
- Select the Insert > Field menu.

Specifying the Field Before Adding the Object

These methods of adding fields require a field to be specified as part of the process of using them:

- Drag one or more fields from the `Field Picker` palette onto the layout.
- When viewing a layout in `Table view`, use the hidden menu that appears in the upper-right corner of the column heading when the cursor is placed over it. Choose the `Table View > Modify` option.
- Copy and paste one or more existing fields from any layout in the same or another database file.
- Add fields through one of the interface dialogs during the process of creating a new layout (Chapter 28, “Creating a New Layout”).
- Drag and drop a field from a layout in one open window onto a different layout open in another window by holding the `Option` (macOS) or `Windows` (Windows) key. You can even drag and drop a field to a layout in another FileMaker database file, although it will vary in the receiving file depending on the presence of a table and field named the same:
 - If the receiving file has a field defined with the same name as the field being dragged in a table with the same name, the field object will be placed referencing that field.
 - If the receiving file does not have a field with the same name but does have a matching table name, the field object will be placed with a `< Field Missing >` reference and will need to be manually pointed to an existing field.
 - If the receiving file does not have a matching field or table name, the field object will be placed as a blank field object and will need to be manually pointed to an existing field.

Adding a Field by Duplication

This final method varies regarding a specify field prompt:

- You can duplicate one or more existing fields to the same layout by selecting the `Edit > Duplicate` menu item. This method will ask you to specify a field when duplicating a single field. However, when duplicating multiple selected fields as once, it will not ask and will instead create exact duplicates pointing to the same field(s) as the originals.

Exploring the Specify Field Dialog

Adding a field to a layout using the methods that do not require a pre-selected field will place a blank field object on the layout, as shown in Figure 30-2.

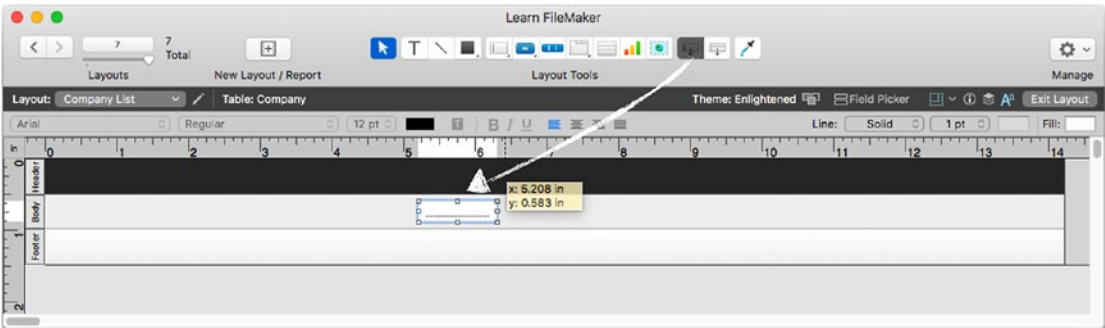


Figure 30-2. Dragging a field onto a layout with the toolbar icon adds a blank field object on the layout before presenting a Specify Field dialog

Once the object is positioned and the cursor released, the blank field object will be tentatively placed at the drop point and become the currently selected object. Since a new field object must contain a defined field reference, a Specify Field dialog, shown in Figure 30-3, will automatically open.

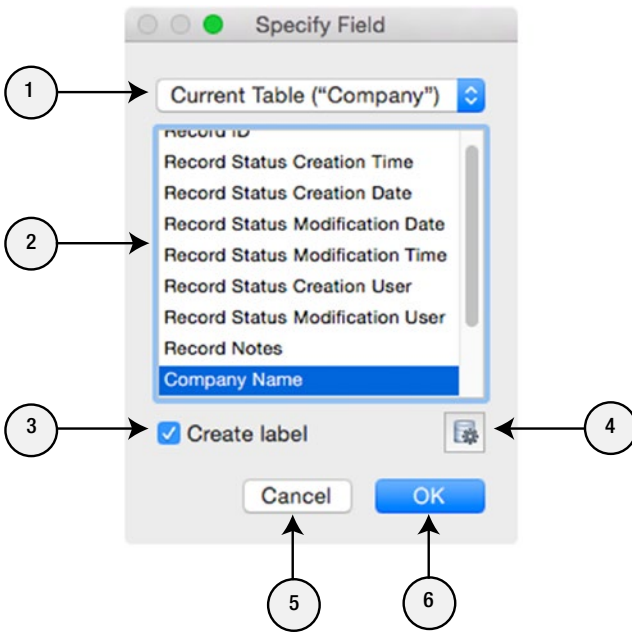


Figure 30-3. The Specify Field dialog is used to establish a reference to a field in a selected table occurrence

The Specify Field dialog is used to identify the field definition assigned to a field object on the layout. The dialog includes the following controls:

1. **Table Occurrence** — This pop-up menu lists every table occurrence defined in the database with the default selection being the layout's assigned occurrence. At the bottom of the menu is a Manage Database shortcut.
2. **Fields** — This area lists every field defined within the selected table occurrence. Click to select the field you want to assign to the selected field object.
3. **Create label** — Check this box to have FileMaker automatically create a field label next to the field object on the layout.
4. **Manage Database** — This button provides an additional shortcut into the Manage Database dialog to quickly create or edit fields without leaving the Specify Fields dialog.
5. **Cancel** — Click this button to close the dialog without saving any changes. If this button is clicked during the automatic process of adding a field to a layout, the field will not be added to the layout and its temporary blank instance will be removed. However, if this dialog is opened to edit an existing field, the field object on the layout will remain unchanged.
6. **OK** — Click this button to close the dialog and update the field object on the layout.

Once finished, the field object will remain selected and will display the name of the selected field definition that it represents, as shown in Figure 30-4. If the Create label check box in the Specify Field dialog was selected, a text object will be present to the left of the field and contain the full name of the field.

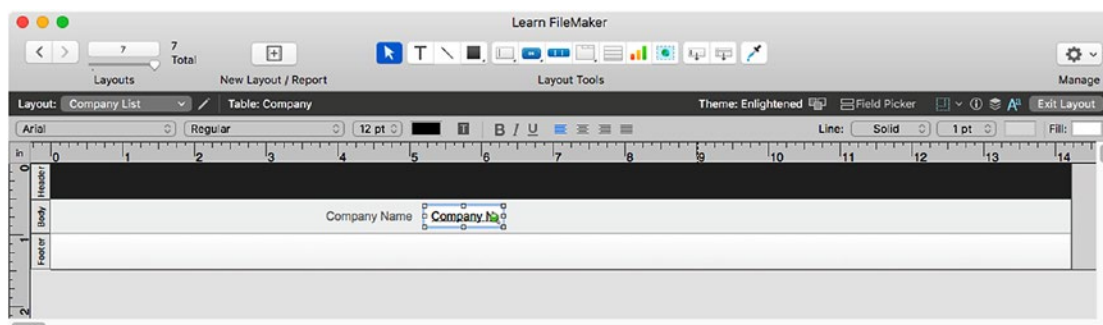


Figure 30-4. The newly added field object selected on the layout with a corresponding label

Editing a Field Object's Reference

There are many occasions where an existing field object may require reassignment to a different field reference. Correcting a mistaken field assignment might be rare, but it happens. Modifying a layout to reflect structural changes to field definitions might be required if a field moves to another table or if a new field is created to replace an older one. Finally, re-configuration of field objects that were created by duplication of existing fields or entire layouts is a more common practice.

To reassign the reference for a field object, open the `Specify field` dialog using any of these methods:

- Double-click on the field object.
- Bring up the object's contextual menu and choose the `Specify field` option.
- Select the field and then click the pencil icon next to the `Display data` area of the `Field group` on the `Data` panel of the `Inspector`.

Once open, the `Specify field` dialog will have the current table and field assignment selected, which can each be modified to point to another table occurrence and/or field. Then, click `OK` to save the change.

Working with Field Labels

A *field label* is a text object placed adjacent to a field object that helps a user identify the assigned field definition and, therefore, the type of information that should be entered or will be displayed within.

A label can be created manually by typing into a text object or automatically using the `Create label` check box in the `Specify Field` dialog. Once placed on a layout, labels can be moved, formatted, and manipulated like any other text object.

Creating a label automatically has one advantage over manual labels. If the text of the label hasn't been manually edited since it was automatically created, the label remains *dynamically linked* to the field. This means that, if the field's defined name is modified, the label automatically updates to reflect the new name. Labels that are automatically created but later manually edited will no longer be linked to the field and will not automatically update.

■ **Note** When a field's name is technically descriptive or arranged to accommodate sorting in the development environment, the layout instance will require an edited label. For example, a field defined as `Contact Name First` might have a layout label of `First Name` or just `First`. If dynamically linked field labels are desired, the field's defined name *must* match whatever text one wants displayed on layouts.

A label that has been modified and unlinked from the defined field can be deleted and re-created as a dynamically linked label by reopening the `Specify Field` dialog for the field and selecting the `Create label` check box before clicking `OK`. The label will reappear as a dynamically linked value.

Configuring Field Control Style

A *control style* is a field object setting that influences the general appearance of and entry options available to a field on a layout. Fields can be assigned one of the following seven control styles:

- Edit box (default)
- Pop-up menu
- Drop-down list
- Checkbox set
- Radio button set
- Drop-down calendar
- Concealed edit box

To change the control style, select a field and open the Field group on the Data panel of the Inspector. Click the Control style pop-up menu, shown in Figure 30-5, and select the desired style.

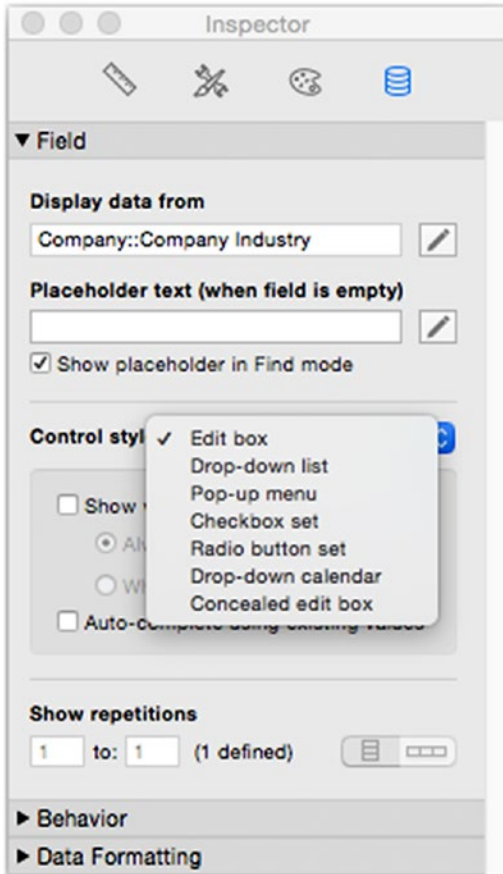


Figure 30-5. The Control style pop-up menu is used to change the appearance, options, and behavior of a field

Once a control style is selected, the options in the shaded area of the Inspector beneath the pop-up menu will change to reflect the choice, enabling only those options that are applicable.

■ **Note** The ability of a user to view or edit any field can depend on many factors including the field's configuration and what access privileges the user is granted based on their login credentials. The assumption throughout this chapter is that a user has full access, which may not be the case when a database uses security settings.

Edit Box

An *Edit box* is the default field control style that renders a field as a box into which the user can type, paste, or drag information. These can vary in size to accommodate data ranging, including these shown in Figure 30-6:

- A single line able to contain a short piece of text.
- Multiple lines or paragraphs of text.
- A scroll bar to allow entries longer than the size of the field object.

Company Name

Company Website

Record Notes

Figure 30-6. Examples of various edit style boxes

Any field data type *can* be formatted as an edit box even though the content doesn't allow for multiple lines. For example, a date field can be formatted with multiple lines and a scroll bar even though, by its nature as a date container, it can only contain a date.

■ **Note** Container fields are automatically locked to an Edit box control style.

Control Options for Edit Boxes

The shaded option area below the Control style pop-up menu in the Layout Inspector, shown in Figure 30-7, will update to show only those options available to an Edit box.

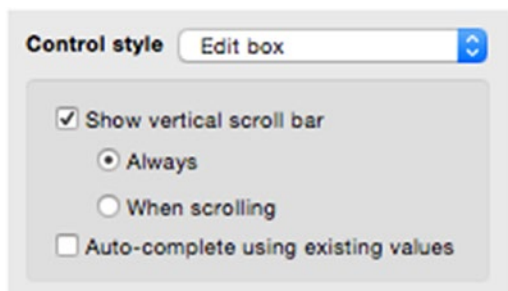


Figure 30-7. Examples of various edit style boxes

The following control options are available for Edit boxes:

- **Show vertical scroll bar** — Use this box to add or remove a scroll bar on the selected field. When the box is checked, a radio button choice below will become enabled with two options:
 - **Always** — When selected, the scroll bar will be visible on the field all the time, even when the field does not have focus. The user can enter the field and scroll by swiping on a touch-enabled device, like a trackpad, or by clicking and dragging up or down on the scroll bar.
 - **When scrolling** — This option will only display the scroll bar when the field has focus and the user is actively scrolling when the content exceeds the height of the field dimensions. There will be no scroll bar to click and drag using this option.
- **Auto-complete using existing values** — Check this box to cause the field to automatically make suggestions based on similar past entries. The suggestion will appear after the text already typed and will be updated as the user is typing. To accept a suggestion, a user can type the Enter or Return keys to automatically complete the entry using that value.

Pop-Up Menu

A *pop-up menu* is a field control style that renders the field as a menu that pops up a list of choices when the user clicks into the field in Browse mode, as shown in Figure 30-8. The list of options contained in the menu is provided by a value list and allows a user to enter a value into a field, replacing any previously entered value, with a simple click of an item from the menu. This entry option helps to save time and maintains consistency by avoiding mistakes for commonly used values.

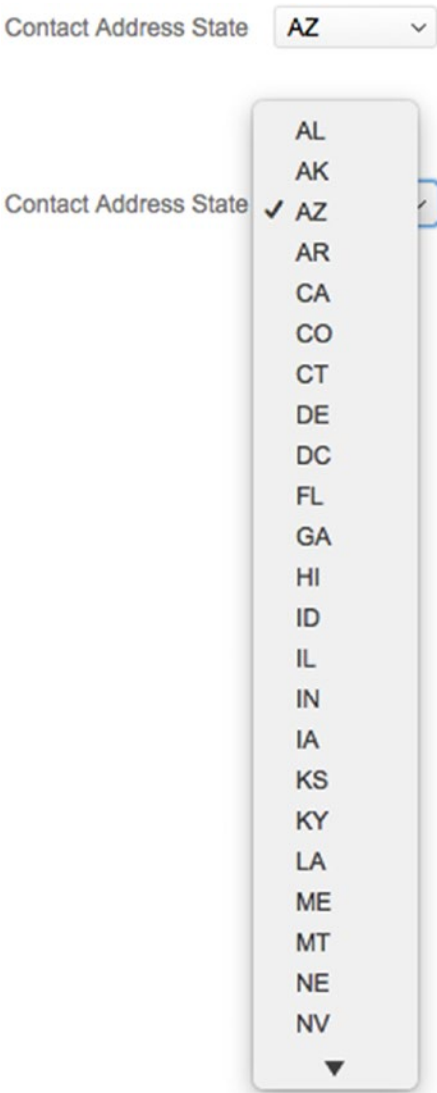


Figure 30-8. Example of a pop-up menu in both a resting state (top) and open with active focus (bottom), the current field value indicated with a check mark and a scroll arrow at the bottom indicating the presence of additional values below.

If the entire list can't fit on the screen, it will truncate and display an up or down arrow icon indicating the presence of additional values out of visual range in one or both directions.

A user can select a value from the list using any of the following methods:

- Position the cursor onto any item in the list and click.
- Begin typing several letters of the desired value to “key scroll” down. Once the desired value is highlighted, type the Enter key to choose that value and enter it into the field.
- Use the Up or Down arrow keys to navigate up or down the list until you reach the desired value. Then type the Enter key to choose that value.

Once the user selects a value, it becomes the field value. When the menu is reopened, that value will have a check mark preceding it in the list, indicating its status as the current value.

Control Options for Pop-Up Menus

The shaded option area below the Control style menu, shown in Figure 30-9, will automatically update to reflect relevant choices based on the Pop-up menu selection.

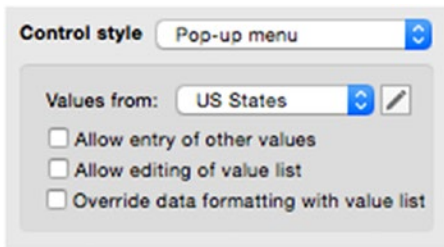


Figure 30-9. The pop-up menu control options

The following control options are available for a Pop-up menu:

- Values from — This pop-up menu contains every value list defined in the current database file, with the selected list providing the values for the field’s menu. Click the adjacent pencil icon to open the Manage Value Lists dialog allowing direct access to create, edit, or select value lists (Chapter 11).
- Allow entry of other values — Check this box to automatically add an extra menu item of “Other” to the end of the values displayed in the field’s pop-up menu. When a user chooses Other in the menu, a small dialog window opens allowing them to type in *any* text value, as shown in Figure 30-10.

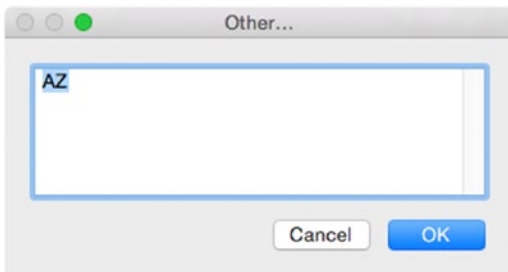


Figure 30-10. The Other dialog in which a user can type any value into the field, bypassing the assigned value list

- Allow editing of value list — Check this box to automatically add an extra menu item of “Edit” to the end of the values displayed in the field’s pop-up menu. When a user chooses Edit in the menu, a small dialog window opens, allowing them to edit the actual contents of the value list, as shown in Figure 30-11. This option is only enabled when the value list items are generated using custom values (Chapter 11, “Using Custom Values”).

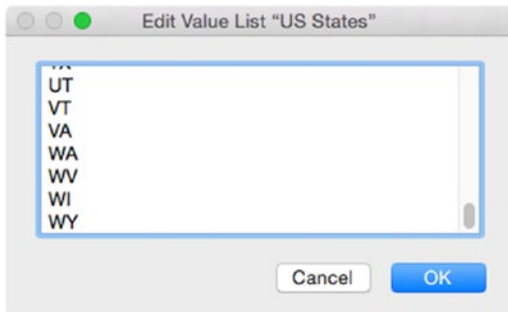


Figure 30-11. The *Edit Value List* dialog in which a user edits the actual items available in the assigned value list

- Override data formatting with value list — This box performs a rather subtle function that is useful when the value list assigned to the field is generated from two fields of differing data types. Without this function, the two values will be displayed correctly in the menu but, once placed into the field, will only display one value. With this option selected, the difference in data types between the two fields will be remedied, and the field will display both values properly.

Drop-Down List

A *drop-down list* is a field control style that renders a field as a combination of an edit box with an attached menu of values, as shown in Figure 30-12. This allows a user a choice of typing a unique value or choosing a preexisting value from a list.

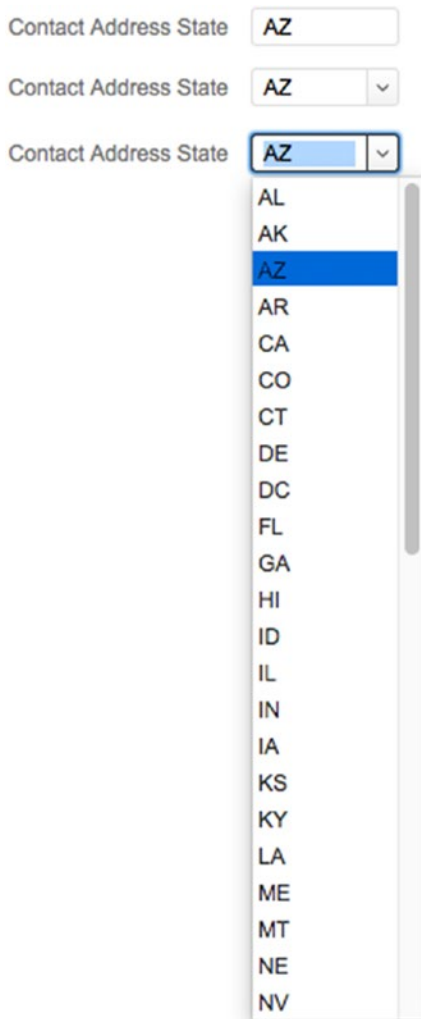


Figure 30-12. Example of a drop-down list in both a resting state (with and without optional arrow icon) and open with active focus

The behavior of the drop-down list is like a pop-up menu. When the list of values exceeds the space available, a scroll bar appears indicating the presence of additional values. A user can use the cursor or arrow keys to scroll the list up or down.

Selecting a value from the list can be achieved using any of the following methods:

- Position the cursor onto any item in the list and click.
- Begin typing several letters of the desired value to “key scroll” down. Once the desired value is highlighted, type the Enter key to choose that value and enter it into the field.
- Use the Up or Down arrow keys to navigate up or down the list until you reach the desired value. Then type the Enter key to choose that value.

Once the user selects a value, it becomes the field value. When the menu is reopened, that value will be highlighted in the list, indicating its status as the current value.

Control Options for Drop-Down Lists

The shaded option area below the Control style menu, shown in Figure 30-13, will automatically update to reflect relevant choices based on the Drop-down list selection.

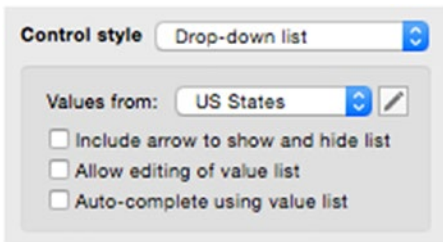


Figure 30-13. The drop-down list control options

The following control options are available for a Drop-down list:

- **Values from** — This pop-up menu contains every value list defined in the current database file, with the selected list providing the values for the field’s menu. Click the adjacent pencil icon to open the Manage Value Lists dialog allowing direct access to create, edit, or select value lists (Chapter 11).
- **Include arrow to show and hide list** — By default, with this box unchecked, when the user enters the field, the drop-down list immediately appears. They have the option to ignore it and type or to select an item. By enabling this check box the field will be rendered on the layout with a visible icon attached to its right side. A user can click into the field and type without distraction and must click the icon to show the list.
- **Allow editing of value list** — Like the Pop-up menu control style, checking this box adds an “Edit” menu item to the end of the values displayed in the field’s drop-down list. When a user chooses “Edit” from the list of available values, a small dialog window opens allowing them to edit the actual contents of the value list. This option is only enabled when the value list items are generated using custom values (Chapter 11, “Using Custom Values”).
- **Auto-complete using value list** — Enable this check box to cause the field to automatically make suggestions as the user types, based on past entries matching the text they are typing. The suggestion will appear after the text the user typed and will keep updating as they continue typing. To accept a suggestion, a user can type the Enter or Return keys to automatically complete the entry using that value.

Checkbox Set

A *checkbox set* is a field control style that renders the field as a set of boxes, each with a text label, as shown in Figure 30-14. These allow a user to select or unselect individual options from the list with a click of the mouse.

Company Industry	<input type="checkbox"/> Accommodations	<input type="checkbox"/> Electronics	<input checked="" type="checkbox"/> Music
	<input type="checkbox"/> Accounting	<input type="checkbox"/> Employment	<input type="checkbox"/> Pharmaceutical
	<input type="checkbox"/> Advertising	<input type="checkbox"/> Energy	<input type="checkbox"/> Public Administration
	<input type="checkbox"/> Aerospace	<input type="checkbox"/> Entertainment & Recreation	<input type="checkbox"/> Public Relations
	<input type="checkbox"/> Agriculture & Agribusiness	<input type="checkbox"/> Fashion	<input type="checkbox"/> Publishing
	<input type="checkbox"/> Air Transportation	<input type="checkbox"/> Financial Services	<input type="checkbox"/> Real Estate
	<input type="checkbox"/> Apparel & Accessories	<input type="checkbox"/> Fine Arts	<input type="checkbox"/> Retail
	<input type="checkbox"/> Auto	<input type="checkbox"/> Food & Beverage	<input type="checkbox"/> Service
	<input type="checkbox"/> Banking	<input type="checkbox"/> Health	<input type="checkbox"/> Sports
	<input type="checkbox"/> Beauty & Cosmetics	<input type="checkbox"/> Information	<input type="checkbox"/> Technology
	<input type="checkbox"/> Biotechnology	<input type="checkbox"/> Information Technology	<input type="checkbox"/> Telecommunications
	<input type="checkbox"/> Chemical	<input type="checkbox"/> Insurance	<input type="checkbox"/> Tourism
	<input type="checkbox"/> Communications	<input type="checkbox"/> Journalism & News	<input type="checkbox"/> Transportation
	<input type="checkbox"/> Computer	<input type="checkbox"/> Legal Services	<input type="checkbox"/> Travel
	<input type="checkbox"/> Construction	<input type="checkbox"/> Manufacturing	<input type="checkbox"/> Utilities
	<input type="checkbox"/> Consulting	<input type="checkbox"/> Media & Broadcasting	<input type="checkbox"/> Video Game
	<input type="checkbox"/> Consumer Products	<input type="checkbox"/> Medical Devices & Supplies	<input type="checkbox"/> Web Services
	<input type="checkbox"/> Education	<input type="checkbox"/> Motion Pictures & Video	

Figure 30-14. Example of a checkbox set for the *Company Industry* field

Unlike a pop-up menu or drop-down list, if given enough layout space, a checkbox set displays *every item* in the list all at once. This makes it much easier for a user to quickly find and select an item. Instead of scrolling up and down a lengthy list, users can visually scan the entire list, locate the item(s) they are looking for, click to select, and enter the value into the field.

Checkbox sets are specifically intended to facilitate the entry of *multiple items* into a single field. As the user checks boxes, the corresponding values are inserted into the field as a list of carriage return-delimited values.

Control Options for Checkbox Sets

The shaded option area below the Control style menu, shown in Figure 30-15, will automatically update to reflect relevant choices based on the Checkbox set selection.

Control style:

Values from:

Allow entry of other values

Icon:

Figure 30-15. The *Checkbox set* control options

The following control options are available for a `Checkbox` set:

- `Values from` — This pop-up menu contains every value list defined in the current database file. Select one to assign as the source of values for the checkbox set. Clicking the adjacent pencil icon will open the `Manage Value Lists` dialog allowing direct access to create, edit, or select value lists (Chapter 11).
- `Allow entry of other values` — Like the same option for pop-up menus, enabling this box will automatically add an “Other” check box at the end of the values displayed in the field allowing the entry of one or more carriage return-delimited values through a dialog. From this dialog, the user can edit the values both those that are and are not on in the value list. The check boxes displayed in the field on the layout will automatically be checked or unchecked based on the actual contents of the field edited in this dialog. Any values in the field that are found in the value list will be checked. If any values are added through the dialogs that are *not* found in the value list, the “Other” check box will automatically be checked indicating the presence of values not visualized with check boxes.
- `Icon` — This pop-up menu contains two options for the style of the marking used to fill in a checkbox:
 - `X` — This option, which is the default, will mark selected check boxes by placing the same of an “X” within the box.
 - `Check` — This option will mark selected check boxes with a check mark.

Radio Button Set

A *radio button set* is a field control style that renders the field as a set of circles, each with a text label, as shown in Figure 30-16. These allow a user to select one option from the list with a click of the mouse.



Figure 30-16. Example of a radio button set for the `Company Industry` field

Like checkbox sets, the radio button control style allows a user to see all the available values at once. However, unlike check boxes, radio buttons are specifically intended to facilitate the entry of a *single item* into a field. As the user makes one selection, the corresponding value replaces the previous value in the field. While not obvious to users, bypassing this restriction is possible and will be discussed later in this section (“Bypassing Value List Entry Restrictions”).

Control Options for Radio Button Sets

The shaded option area below the Control style menu, shown in Figure 30-17, will automatically update to reflect relevant choices based on the Radio button set selection.

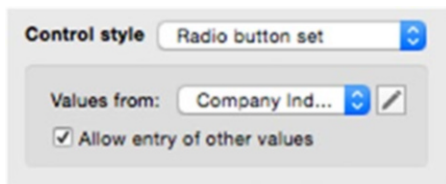


Figure 30-17. The radio button set control options

The following control options are available for a Radio button set:

- **Values from** — This pop-up menu contains every value list defined in the current database file. Select one to assign it to the selected field as the source of available values. Clicking the adjacent pencil icon will open the Manage Value Lists dialog allowing direct access to create, edit, or select value lists (Chapter 11).
- **Allow entry of other values** — Like the same option for Checkbox set, enabling this option will automatically add an “Other” radio button at the end of the values displayed in the field. When the user selects this option, an Other dialog will appear displaying any current selection from the field and allows them to add or replace it with a different value, including one not on the value list.

Unselecting Values

The Radio button format doesn’t provide an intuitive method for removing a selection and leaving the field empty. To clear all selections from a Radio button formatted field use any of these methods:

- To deselect an individual value, click on a selected value while holding down the Shift key.
- If the Other option is enabled, click that and edit the value(s) in the dialog.
- To clear the field completely, click anywhere within its borders and type the Delete key.

Drop-Down Calendar

A *drop-down calendar* is a field control style that renders the field as a combination edit box with an optional graphical calendar that drops down attached to the bottom of the field, as shown in Figure 30-18. This allows a user a choice between typing a date manually into an edit box or choosing a date using a familiar calendar graphic.

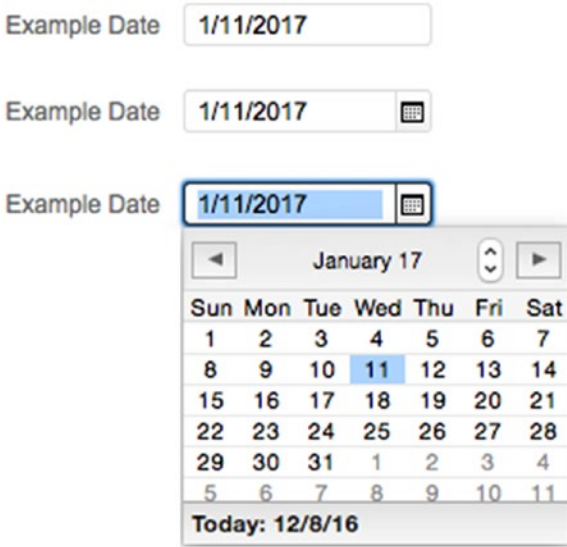


Figure 30-18. Example of a drop-down calendar in both a resting state (with and without optional icon) and open with active focus

The calendar panel has several controls, shown in Figure 30-19, that assist a user in quickly locating a desired date.

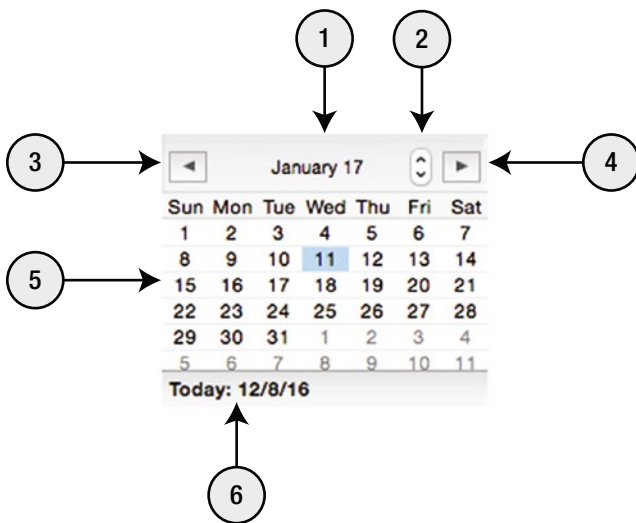


Figure 30-19. The calendar panel with controls highlighted

1. **Month & Year** — The heading displays the currently viewed month and year.
2. **Step by Year** — The small up and down arrows will step backward or forward one year at a time for the current month.
3. **Step back** — The left arrow will step backward to the previous month.
4. **Step forward** — The right arrow will step forward to the next month.
5. **Calendar** — Within the calendar area, a user can click on a date to select and enter it into the corresponding field. Using only the keyboard, the arrow keys will navigate around the days and the Enter key will select and enter the current date selected into the field.
6. **Today** — This area always displays the current date and, if clicked, will enter that into the corresponding field and close the calendar.

■ **Note** The Drop-down calendar control style can be used with non-date fields like a text or timestamp field. When using it with text fields, the selected value will be entered as a text string. When used with timestamp fields, the date selected will be entered with the current time.

Control Options for Drop-Down Calendars

The shaded option area below the Control style menu, shown in Figure 30-20, will automatically update to reflect relevant choices based on the Drop-down calendar selection.

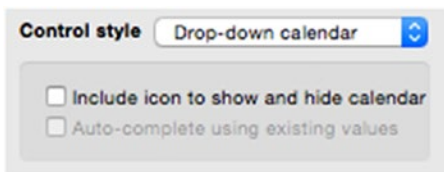


Figure 30-20. The drop-down calendar control options

The following control option is available for a Drop-down calendar:

- **Include icon to show and hide calendar** — With this box unchecked, the Drop-down calendar immediately appears when a user enters the field. They have the option to use or ignore it. Enabling this option will render the field with a visible icon attached to the right side that will open or close the calendar. A user can click into the field and type without distraction or choose to click the icon to show the graphical calendar.
- **Auto-complete using existing values** — This check box becomes enabled when a calendar is used with a text field only. Checking it causes the field to automatically make suggestions based on similar past entries updated as the user is typing like an Edit box. Typing the Enter or Return keys will automatically complete the entry using the suggested value.

Concealed Edit Box

A *concealed edit box* is a field control style that renders the field as an edit box with each character typed represented as a bullet to obscure the data, as shown in Figure 30-21. When the field has focus, it displays one bullet for every character entered but reverts to displaying eight bullets once focus shifts to another field or the record is committed.

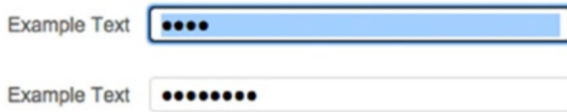


Figure 30-21. Examples of a concealed edit box, both with focus and without, each displaying the same entry

■ **Caution** Concealed edit boxes are intended to provide *minimal security* only. The data in the field is *not* encrypted. It is only obscured from view on layouts but continues to be accessible to scripts and calculations, which can access and manipulate the data as they would any other field.

Exploring the Benefits of Pop-Up Menus for Two-Field Value List

There are some important factors to consider when using a value list that is generated from field values for drop-down and pop-up menus (Chapter 11, “Using Values from a Field”).

To illustrate the issues involved, imagine a value list named `Company List`, as shown in Figure 30-22, which generates a list of every company record. The list would be assigned to the `Contact Company ID` field in the `Contact` table to allow a user to select which company a contact is connected.

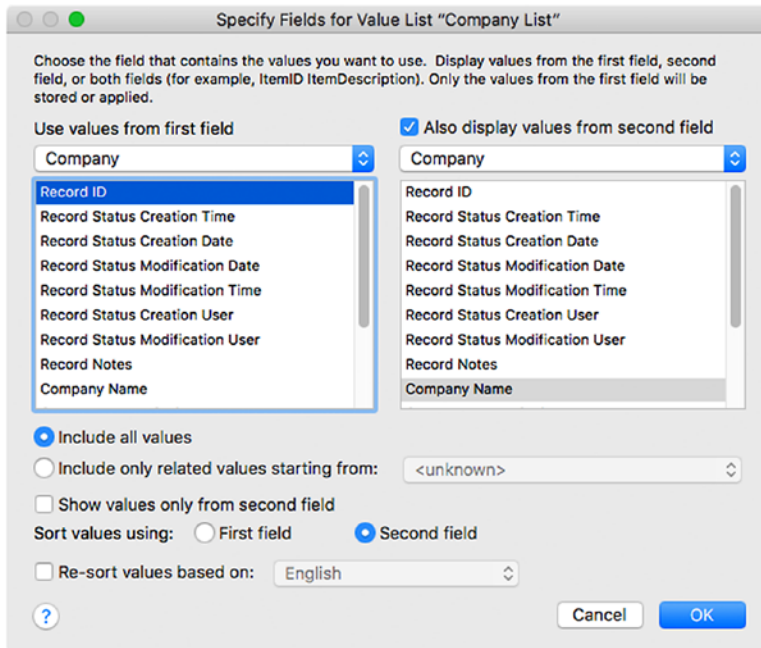


Figure 30-22. An example of a value list generated from the contents of two fields

A Contact record must contain the Record ID of a Company in the Contact Company ID field to form the relational link between the two. This mandates that the first field in the value list be the Company::Record ID.

However, with *only* this field, the value list will display a list of id numbers and be rather useless for most users. If a user doesn't know the id of the company, the list will be no help to them. If they do know the id of the company the list isn't required. So, this alone isn't adequate when using lists to aid users in entering a match field value used in a relationship.

The *Also display values from second field* option allows items in a value list to be pulled from the actual values of *two* fields to create a list of combined values: in this case, the *id* and the name of the Company record. The first value continues to provide the Record ID and the second value provides a human-friendly name that corresponds to that id. By adding the Company Name to the list, the user can select based on the second field while the first field is what is entered in the field.

This may seem clear in terms of generating a list. However, when applied to a drop-down list or pop-up menu layout object, there are subtle implications here that may influence your choice of object control type and the use of the *Show values only from second field* option in the value list.

Using Two-Field Value Lists with the Drop-Down List Control Style

When the dual-field value list is assigned to the Contact Company ID field as a drop-down list using that configuration, it will display a list of values containing the Record ID and Company Name fields, as shown in Figure 30-23.



Figure 30-23. The value list assigned as a drop-down list to the Contact Company ID field

When used in this configuration, the user can ignore the id portion of the values in the list and focus on selecting a company by name. Once they select a value, the id is entered in the field and the list will disappear. Since, the field will contain and display *only* the id, a second field will be required on the layout to display the Company Name. This can be any of the following depending on needs or preference:

- A *field object* referencing the Company Name field through a relationship.
- A *calculation field* locally created in the Contact table that pulls the Company Name field value through a relationship.
- A *text field* locally created in the Contact table that auto-enters the Company Name field value through a relationship.

While this method works fine, some find it a little visually cluttered to see both the id and the name in the list and prefer the id be hidden. This can be done by enabling the Show values only from second field check box on the value list. However, that results in possible confusions as the user sees a list of company names but their selection is placed into the field as an id, as shown in Figure 30-24.

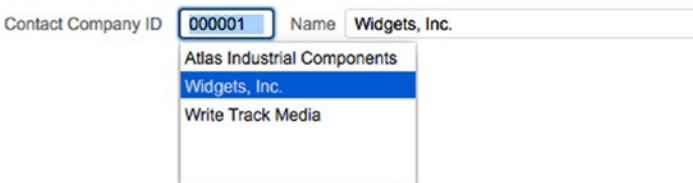


Figure 30-24. The value list with the id hidden shows only company names while still placing the id in the field

In the past, this led developers to pursue more creative solutions of overlapping the two fields so that the user only sees the company name in the list and the field without realizing that the id was driving the whole process. However, there is a much more professional and less contrived method of achieving the same end using the pop-up menu control style.

Using Two-Field Value Lists with the Pop-Up Menu Control Style

When using the dual-field value list with the pop-up menu control style, there is no difference between what a user sees in the list and what they see in the field in its resting state, as shown in Figure 30-25. In both cases, the user sees the actual value they selected. In this case, it is both the id and the name.



Figure 30-25. A dual-field value list assigned to a field using the pop-up menu control style displays both fields in its resting state

Since *both* fields will be displayed in the value list, this method eliminates the necessity of adding another field to the layout exclusively to display the second field in the value list. Further, by enabling the Show values only from second field check box on the value list, the field will display only the second field, the company name, *both* in the open menu and the field at rest, as shown in Figure 30-26.



Figure 30-26. A dual-field value list showing only the second field in the menu will continue displaying that value when the field is at rest

Using this method, the field still contains *only* the id of the related Company record while displaying only its name. This completely hides the functional aspect while providing a polished, user-friendly interface when selecting from the menu and viewing the field in its resting state.

Bypassing Value List Entry Restrictions

Each of the four control styles that allow entry from a value list has an interface mechanism that subtly encourages compliance with at least one of the following two rules, as shown in Table 18-1:

- Values entered must be found in the assigned value list.
- Entry is restricted to a single value.

Table 18-1. Showing the default implied restrictions for entry

Control style	Single Item	Value List Only
Pop-up menu	Yes	Yes
Drop-down list	Yes	—
Checkbox set	—	Yes
Radio button	Yes	Yes

These rules are not enforced except by implication in the method of interacting with the field through the control style interface elements. A pop-up menu and radio buttons both do not offer any obvious method to select multiple items or to make an entry not on the list. Similarly, a drop-down list seems to allow only a single value while a checkbox set without the setting to allow other values appears to only permit value list selections and no other values. However, these rules are easily bypassed if you know how and this can be either a benefit to allow users a more flexible environment or a developer's worst nightmare.

Disabling the Allow entry of other values setting *doesn't* stop someone from entering other values and there are ways to make multiple selections with each of the four control styles. It is important to understand that these restrictions are cosmetic only and you must take explicit steps to stop users from bypassing them if it's important.

Bypassing a Value List with Custom Values

Any user with access to edit a field's contents can enter custom values and bypass the value list completely using any of the following methods:

- Once the field has focus, a user can insert text from the clipboard using the Paste command or any of the functions under the Insert menu.
- A user can select text in another field or another application and use the operating system's Drag function to Drop the content into the field.
- A script can place any text into the field using many different script steps such as Set Field, Insert Text, Replace Field Contents, Import Records, and more.

To forbid these methods, you must explicitly restrict the ability of a user or script to bypass the value list.

Restricting Value List Bypass

For situations where you want to tighten up data entry to *absolutely* forbid a user from bypassing value list items, use any of these methods:

- Restricting field to a member of a value list
- Filtering bad values with an Auto-Enter calculation
- Using a script trigger to clean the entry

Restricting Field to a Member of a Value List

One method of forbidding an entry of any value not found in the assigned value list is to use the field validation option. To do so, enable the Member of value list check box on the Validation tab of the Options for Field dialog (Chapter 8) and then select the same value list used on the layout field object.

■ **Note** This is a field definition setting and will apply to *every* use of the field on any layout including those where a value list-driven control style is not used!

Filtering Bad Values with an Auto-Enter Calculation

When defining fields, enter a Calculated value in the Auto-Enter tab of the Options for Field dialog that compares the value(s) in the field to the items in the value list. The formula can automatically filter out any value entered that is not in the list without a warning as shown here:

```
Let (
    list.items = ValueListItems ( Get ( FileName ) ; "<value.list.name>" );
;
    Case (
        IsEmpty ( FilterValues ( list.items ; Self ) ) ; "" ) ;
        Self
    )
)
```

In the formula, the `ValueListItems` function puts a list of all the items available in the value list into a `list.items` variable within the `Let` statement. Then, a `Case` statement uses the `FilterValues` function to remove every item from `list.items` except what was entered into the field. If the result is an empty string, as detected by the `IsEmpty` function, we know that the entered value is not present in `list.items` and can replace it with an empty string. If the value is present, it can remain.

Validating multiple values in a field using an auto-enter calculation would require a recursive custom function (Chapter 25), which checks the values one by one.

■ **Note** This is a field definition setting and will apply to *every* use of the field on any layout including those where a value list-driven control style is not used!

Using a Script Trigger to Clean the Entry

Assigning an `OnObjectValidate` script trigger to the field (Chapter 37) to run a script (Chapter 34) that uses a `Loop` script step to remove one or more values not found on the value list or present an informative warning dialog as needed.

■ **Note** This is a layout object setting and will *only* apply the field where configured and not necessarily on every instance of the field on every layout.

Selecting Multiple Items

`Checkbox` sets are the only value list control style that intuitively allows the selection of multiple items from a value list; the user clicks one check box and then another. Other control styles always replace the existing value when a new value is selected and gives the appearance of the restriction of a single selection. However, it is possible to select more than one item from pop-up menus, drop-down menus, and radio buttons using any of the following methods:

- Click on multiple options while holding down the `Shift` key.
- Use the `Paste` command to enter a carriage return-delimited list of values into the field.
- Select a carriage return-delimited list of values in another field or another application and use the operating system's `Drag` function to `Drop` the content into the field.
- When the `Allow entry of other values` option is enabled, the `Other` dialog will allow a user to manually enter multiple values.
- Use an `OnObjectEnter` script trigger to store the last value in a global variable and then an `OnObjectSave` to save both the current and last values into the field together.
- A script can place multiple values into the field using many different script steps such as `Set Field`, `Insert Text`, `Replace Field Contents`, `Import Records`, and more.

Restricting Entry to a Single Value

To tighten up data entry to *absolutely* forbid the entry of more than one value, there are several approaches you might take, including:

- Using Auto-Enter to limit field to a single value
- Using a script trigger to limit field to a single value

Using Auto-Enter to Limit Field to a Single Value

One method for ensuring that a field contains a single value is to enter a `Calculated` value from the `Auto-Enter` tab of the `Options for Field` dialog. The formula can detect multiple values and automatically removes all except for one. In this example, only the last value will be retained:

```
Case ( PatternCount ( Self ; "¶" ) > 0 ; GetValue ( Self ; ValueCount ( Self ) ) ; Self )
```

■ **Note** This is a field definition setting and will apply to *every* use of the field on any layout including those where a value list-driven control style is not used!

Using a Script Trigger to Limit Field to a Single Value

Another method for ensuring that a field contains a single value is to assign an `OnObjectValidate` script trigger to the field (Chapter 37) that runs a script to detect the presence of multiple values and either automatically removes excess values or warns the user and halts the validation process until they correct the problem.

■ **Note** This is a layout object setting and will *only* apply to the field where configured and not necessarily on every instance of the field on every layout.

Using Placeholders

A field can be configured with a *placeholder text* value that will be displayed in the field whenever it is empty. A placeholder can be used to display the name of the field in lieu of an actual field label or it may contain a short prompt to action meant to augment the field label. This essentially gives you three options: no placeholder, a placeholder as field label, or a placeholder as a prompt to action, each of which is demonstrated in Figure 30-27.



Figure 30-27. An example of three instances of the same `Company Name` field: one with no placeholder text, one with the field name as the placeholder, and one with a prompt message as the placeholder.

This example uses three instances of the same `Company Name` field, emphasizing the fact that the placeholder setting is applied to the field as a *layout object* and not to the *field definition* itself. Therefore, multiple instances of the same field on the same or different layouts can have a completely different placeholder value and must be configured separately.

■ **Note** A field that is configured as a `Checkbox` set or `Radio` button can't have a placeholder text value.

Entering Placeholder Text

Placeholder text can only be entered from within the `Field` group on the `Data` panel of the Inspector. A static value can be typed directly into the `Placeholder` text box, shown in Figure 30-28. Clicking the pencil icon to the right of this box will open a `Specify` calculation dialog into which you can enter a formula that returns a text value for the placeholder text. This allows a placeholder to be generated by conditional logic based using field contents, the user's context, and more.

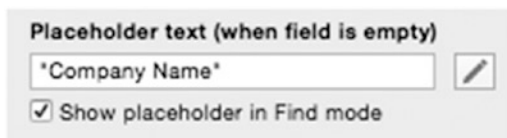


Figure 30-28. The controls for placeholder text from the `Field` group of the `Data` panel of the Inspector

Varying a Placeholder by Window Mode

Placeholder text is displayed when the window is in `Browse` and optionally in `Find` mode. To include the placeholder in `Find` mode, click the `Show placeholder in Find mode` check box below the entry box.

It is possible to dynamically vary the content of the placeholder text based on the window mode using a `Case` statement and the `Get (WindowMode)` function to return a different result for `Browse` and `Find` modes. For example:

```
Case (
    Get ( WindowMode ) = 0 ; "This is the browse mode placeholder" ;
    "This is the find mode placeholder"
)
```

Formatting Placeholder Text

By default, the text formatting of the placeholder is a light gray color. This helps to emphasize the fact that the value displayed is not *actually* the field content but is a piece of information or a prompt to action. If format settings are modified for the field's placeholder state, be sure to maintain a noticeable difference between the actual contents of the field and the placeholder text formatting.

Using Placeholder Text Instead of Display Calculation Fields

The Placeholder Text feature can display calculated information on layouts as an alternative to creating numerous calculation fields for this purpose. Instead of creating calculation fields to gather information from fields or about the context for interface display purposes only, consider creating a single calculation field that results in an empty string and uses global storage. When a layout needs to display a calculated result, such as a header or warning prompt, put that field in place and enter the specific information into the Placeholder Text formula. The formatting of the placeholder can be modified by modifying the style settings for the field's Object State (Chapter 32) so that it appears like a normal field or has formatting appropriate to its function.

Since the field uses global storage, this *single* field can be used anywhere in the database file to greatly reduce the number of fields required for display purposes.

■ **Caution** Since the data displayed in these fields is the placeholder value and not actually present in the field, there is no way to use this information for anything other than display. For example, it can't be exported and can't be referenced in other formulas or used by script steps.

Showing Field Repetitions

Fields that are defined with multiple repetitions (Chapter 8) can be displayed on a layout with some or all repetitions running vertically or horizontally, as shown in Figure 30-29.



Figure 30-29. A text field defined with five repetitions displayed on a layout in both vertical and horizontal arrangement

The Show repetition controls, shown in Figure 30-30, are accessed from the Field group of the Data tab of the Inspector. The number of defined repetitions for the field is displayed in parentheses. The unlabeled Start and End text areas allow control over which and how many repetitions are visible on the layout. The Vertical and Horizontal buttons allow control over the orientation of the repetitions.



Figure 30-30. The Show Repetition controls on the Inspector

Each instance of a repeating field on a layout can be configured to include any one or group of consecutive repetitions within the range defined by the field options. For example, each repetition can be displayed individually as if they were separate fields or as partial groups of repetitions, both shown in Figure 30-31.

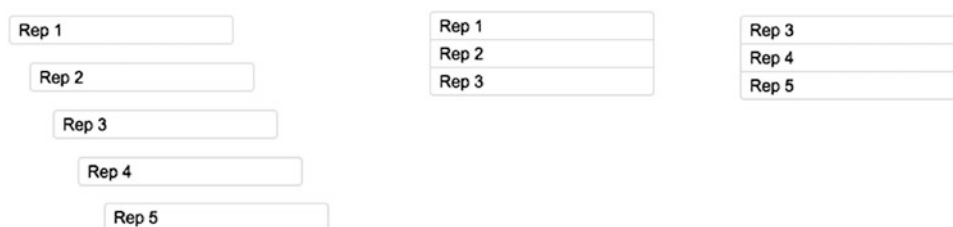


Figure 30-31. Several examples showing the flexibility of configuring repeating fields individually or in groups

■ **Note** Other field settings for Behavior and Data Formatting are located on the Data panel of the Inspector and were discussed in Chapter 29.

Working with Text

Text objects on a layout play an important role identifying the nature and content of other objects, informing users about proper data entry procedures, or prompting them to action.

In addition to field labels, which are discussed earlier in this chapter, there are four types of text that can be placed on a layout:

- Static text
- Dynamic symbols
- Merge fields
- Merge variables

Static Text

On a layout, *static text* is an object that contains an unchanging string of characters. Static text can be created one of two ways while in Layout mode:

- *Create a Text Object* — Choose the Text tool in the toolbar and then click and drag to define an area for the text object. When you release the cursor, the object will be created and have editing focus allowing you to type.
- *Convert a Field Label* — As mentioned earlier, a field label will dynamically update its content to match the name of its corresponding field until it is edited. Once modified, the dynamic label becomes a static text object.

Text objects can contain rich formatting and multiple paragraphs.

Dynamic Placeholder Symbols

A *dynamic placeholder symbol* is a text string placeholder that will be automatically replaced with a predetermined value that varies over time.

Symbols are made up of a keyword enclosed in double braces, following this pattern:

```
{{SymbolKeyword}}
```

For example, instead of typing a date on a layout as a piece of static text, you can insert the following symbol in a text object, indicating to FileMaker that it should be rendered with the current date:

```
{{CurrentDate}}
```

There are 100 different symbols available, each corresponding to a Get function (Chapter 22). Symbols may be combined with other symbols or static text in a single object. For example:

```
Report printed on {{CurrentDate}} at {{CurrentTime}} by {{AccountName}}
```

A symbol can be manually typed or added to a layout by selecting one from the Insert menu. When using the menu, the symbol will be inserted into an existing text object if there is one with edit focus currently active, otherwise a new text object containing the symbol will be created.

Merge Fields

A *merge field* is a symbolic tagging format that allows a field's text content to appear on a layout in a non-editable text object, optionally combined with static text.

Merge fields are made up of a field name enclosed in left-shift (<<) and right-shift (>>) symbols, following this pattern:

```
<<FieldName>>  
<<TableName::FieldName>>
```

When referencing a field local to the current layout's table, only the field name needs to be specified. A field from a related table must include the relationship name to indicate which table occurrence to pull through.

For example, a Contact name and Company name might be placed directly on a layout with the following text:

```
Contact <<Contact Name First>> <<Contact Name Last>> at <<Contact | Company::Company Name>>
```

Merge fields may be combined with other merge fields, other text and symbols. They can be added to a layout by selecting the Insert > Merge Field menu. As with symbols, the merge field tag will be inserted into an existing text object if there is one with edit focus currently active; otherwise a new text object containing it will be created.

Merge Variables

A *merge variable* is a symbolic tagging format that allows a global variable to be rendered within a text object on a layout.

Merge variables are made up of a variable name enclosed in left-shift (<<) and right-shift (>>) symbols, following this pattern:

```
<<$$VariableName>>
```

Merging variables on a layout works exactly like merge fields and dynamic symbols. They can be used in any combination with other merge variables, text, symbols, and merge fields. Use the Insert > Merge Variable menu to add one into the current text object with edit focus or a new object.

Working with Button Controls

A *button* is a layout object that embeds control of a function into an icon or object. Buttons work in Browse and Find mode and, depending on the type and configuration, can run a script, perform a script step, or open a popover interface element. FileMaker has three native button types and allows any layout object to be turned into a button:

- Button
- Popover button
- Button bar
- Object button

Working with Buttons

A *button* is a layout object in the form of a traditional computer interface push button, like that shown in Figure 30-32, which allows a user to perform a script or run an individual script step with a single click.



Figure 30-32. Examples of several push-style button objects

Buttons are configured using the Button Setup dialog, which automatically opens when a new button is created. This dialog can also be opened by any of the following methods:

- Double-click on an existing button.
- With a button selected, click the Format > Button Setup menu item.
- Open the button's context menu and select the Button Setup menu item.

Exploring the Button Setup Dialog

When a button is first created, the Button Setup dialog opens with limited default options, shown in Figure 30-33.

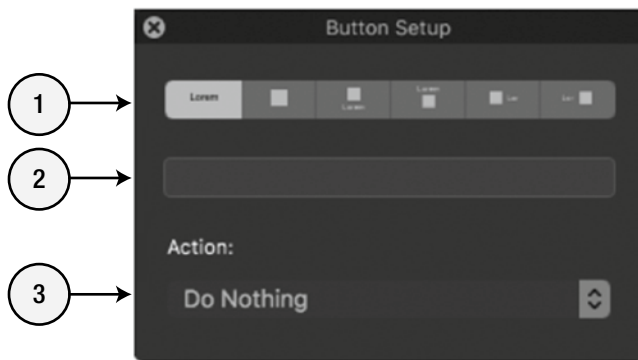


Figure 30-33. The default Button Setup dialog for a new button.

The Button Setup dialog is divided into three sections:

1. *Button Labeling Options* — Choose to have the button labeled with text, an icon, or both in a variety of orientations. The default selection is a text label with no icon. Available options include:
 - Text only (Default)
 - Icon only
 - Icon with text label below
 - Icon with text label above
 - Icon with text label on the right
 - Icon with text label on the left
2. *Button Labeling Specification* — This area dynamically updates the available controls depending on your choice of labeling option above. The default, as shown, allows entry of the button label by typing into the visible text area. However, this area will expand to allow selection of an icon and specification of its size (see the next section).
3. *Action Menu & Specification* — Choose an action type and this area will dynamically update available controls depending on your choice.

■ **Tip** You can create dynamic button labels using merge fields within the name of the field. For example, this button name will put a Contact record's first name field into the button name, creating a personalized button for each record:

Email <<Contact Name First>>

Configuring a Button with an Icon Label

After selecting a button labeling option that includes an icon, the labeling specification area will expand to include icon options, as shown in Figure 30-34.

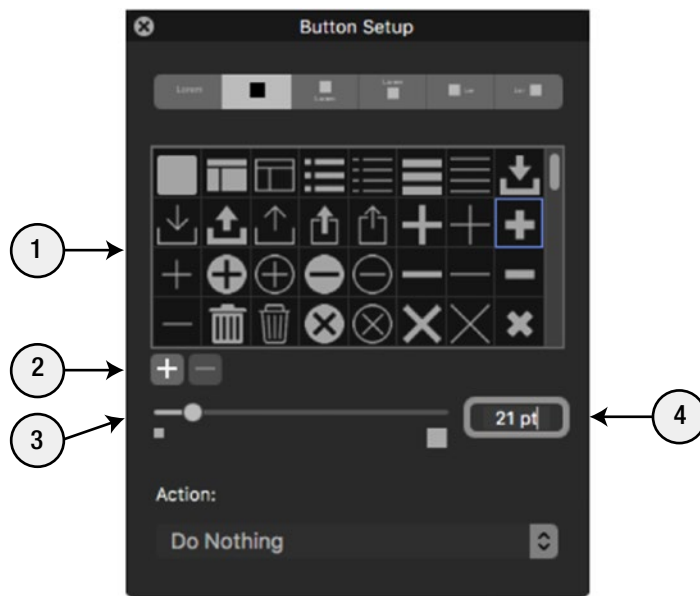


Figure 30-34. The *Button Setup* dialog with expanded icon options

The expanded controls are:

1. *Icon Selection* — Choose from a list of standard icons.
2. *Icon Size Buttons* — Use these buttons to add or remove custom icons to the list.
 - To add an icon, click the + button and then choose a PNG or SVG formatted file and click Insert (macOS) or Open (Windows).
 - To remove a custom icon from the list of available icons, select it and click the - button. Any buttons already using the icon will continue to do so until you reassign a new icon.

3. *Icon Size Slider* — Drag the slider left or right to shrink or enlarge the size of the icon or click either of the two squares to adjust the size one pixel at a time.
4. *Icon Size Setting* — This box displays the current size of the icon. You can type a new number to quickly set a new size.

As you adjust the selected icon and size, the button will update in real time on the layout, giving you a preview of how it will appear. Once it appears as desired, continue configuring other options or close the Button Setup dialog.

■ **Note** Unlike some measurement settings in the Layout Inspector, the Icon size will *always* be displayed in points regardless of units currently selected for the ruler.

Configuring a Button to Perform a Single Step

To assign a button to run a script step command, change the Action pop-up menu of the Button Setup dialog from Do Nothing to Single Step. This will immediately open the Button Action dialog, shown in Figure 30-35.

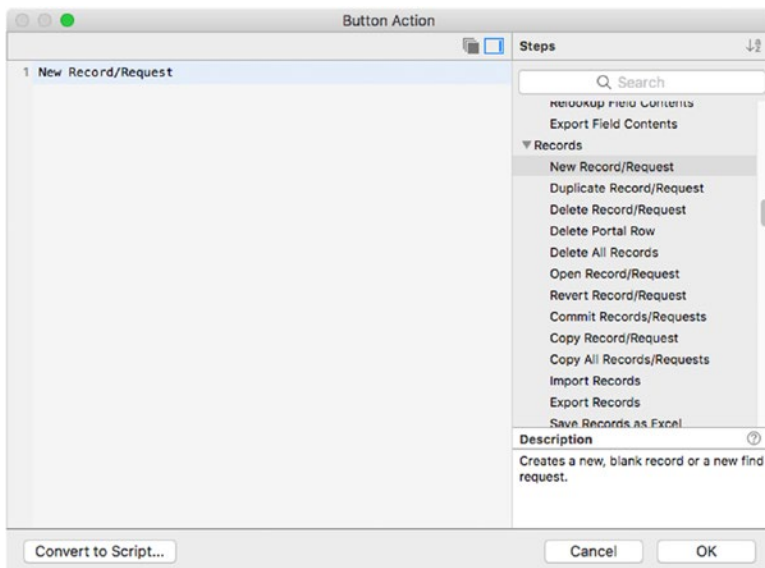


Figure 30-35. The Button Action dialog for selecting the script step that the button will perform

The Button Action dialog looks similar in appearance to a Script Workspace dialog (Chapter 34). Double-click the step from the list on the right to add it as the action for the button. Then click the OK button to save the selection.

■ **Note** When choosing this option, only a single command can be assigned to the button and any subsequent selection will replace the previous one. If more than one command is required for a button, choose *Perform Script* from the *Button Setup* dialog instead of *Single Step*, or click the *Convert to Script* button in the *Button Action* dialog. This convenient button will create a new script with a name you specify; add the selected command and assign the new script to the button all in one step.

Once a step has been assigned to a button, the *Action Specification* area at the bottom of the *Button Setup* dialog will display the command selected, as shown in Figure 30-36.

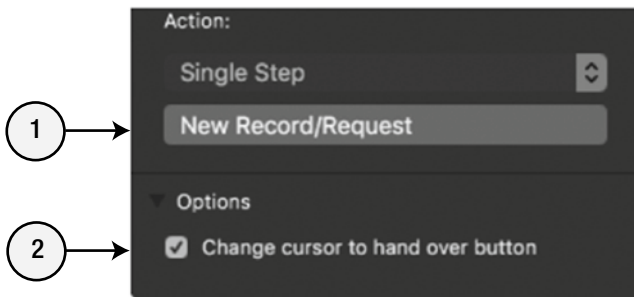


Figure 30-36. The lower portion of the *Button Setup* dialog configured to run a *Single Step*

The following two new controls will be available:

1. *Command Selection* — The selected command will be displayed here as a button. Click to open the *Button Action* dialog and assign a different command.
2. *Options* — When a button has an action assigned, the option to *Change cursor to hand over button* check box will appear. Check this box to make the user's cursor icon change into a hand when placed over the button to intuitively convey the fact that they have the option to click on the button.

Configuring a Button to Perform a Script

Changing the button action option to *Perform a Script*, will immediately open the *Specify Script* dialog, shown in Figure 30-37. This dialog is standard throughout the programming interface when selecting a script.

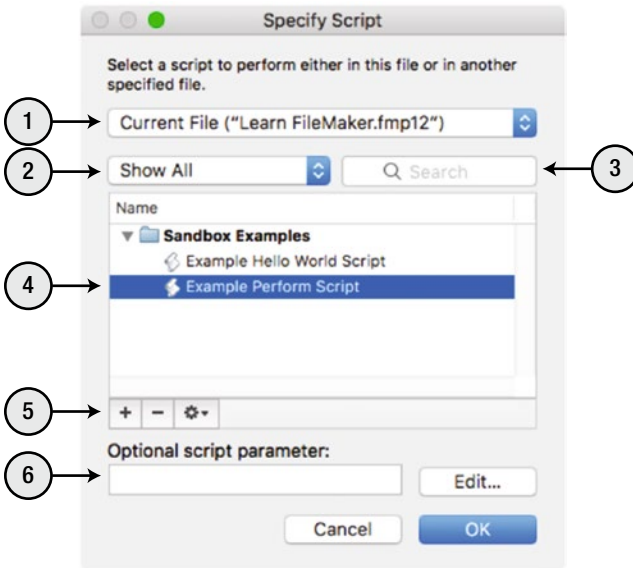


Figure 30-37. The Specify Script dialog

The following controls are available in the Specify Script dialog:

1. *File* — This pop-up menu allows the selection of a database file in which the desired script exists. The default selection at the top of the menu is the current file but the following other options are available:
 - *Existing Data Source List* — Choose any external data sources defined in the current database (Chapter 9, “Managing Data Sources”).
 - *Add FileMaker Data Source* — This option opens a standard Choose File dialog allowing the selection of a FileMaker Pro database. Once selected, clicking OK instantly adds the file as a data source and selects it in the menu.
 - *Add ODBC Data Source* — This option opens the Select ODBC Data Source dialog and allows the selection of a previously defined ODBC data source (Chapter 7, “Adding a Table from an ODBC Data Source”).
 - *Manage Data Sources* — This option opens the Manage Data Sources dialog (Chapter 9, “Introducing the Manage External Data Sources Window”).
2. *Folder Filter* — This pop-up menu contains a list of every script folder defined in the database and can filter the list of available scripts down to only those stored within the selected folder. The default choice of Show All presents an unfiltered list of every script.
3. *Search Filter* — Enter text in this area to filter the list of available scripts to only those whose name contains the value entered.
4. *Script List* — This list displays every script that meets any filtering criteria above, hierarchically arranged in folders. Click the disclosure triangle next to a folder to toggle it open or closed.

5. *Script Buttons* — These buttons allow convenient access to the following script functions:
 - a. *Create* — Click to create a new script and open it in the Script Workspace.
 - b. *Delete* — Click to delete the selected script(s).
 - c. *Edit* — Click to open the selected script in the Script Workspace.
 - d. *Duplicate* — Click to duplicate the selected script(s).
6. *Script Parameter* — Use this text area to enter any parameter information to send to the script or use the Edit button to open a Specify Formula dialog to enter a parameter calculation (Chapter 34, “Exchanging Data Between Scripts”).

Once finished, click OK to save the selection to the button. The options are at the bottom of the Button Setup dialog, as shown in Figure 30-38.

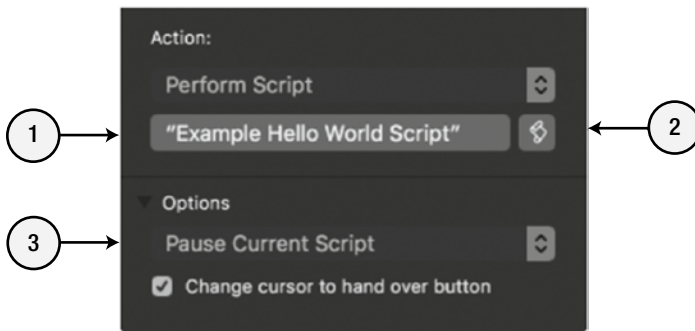


Figure 30-38. The options at the bottom of the Button Setup dialog configured to Perform Script

The following three new controls will be available when using the Perform Script action:

1. *Assigned Script* — This button displays the name of the selected script. Click to open and directly edit the script in the Script Workspace.
2. *Script Selection* — Click this button to open the Specify Script dialog and select a different script.
3. *Current Script Handling* — This pop-up menu controls how to handle a running script that is paused. After the button’s script is run, the available options are:
 - a. *Halt Current Script* — All running scripts can be halted.
 - b. *Exit Current Script* — The paused script will be exited. If another script called that script, control will be returned to it and it will continue running.
 - c. *Resume Current Script* — The paused script will resume running after the button script.
 - d. *Pause Current Script* — The paused script will remain paused.

Configuring a Button Directly from the Contextual Menu

FileMaker allows fast and direct configuration of a button’s action from a contextual menu submenu, shown in Figure 30-39.

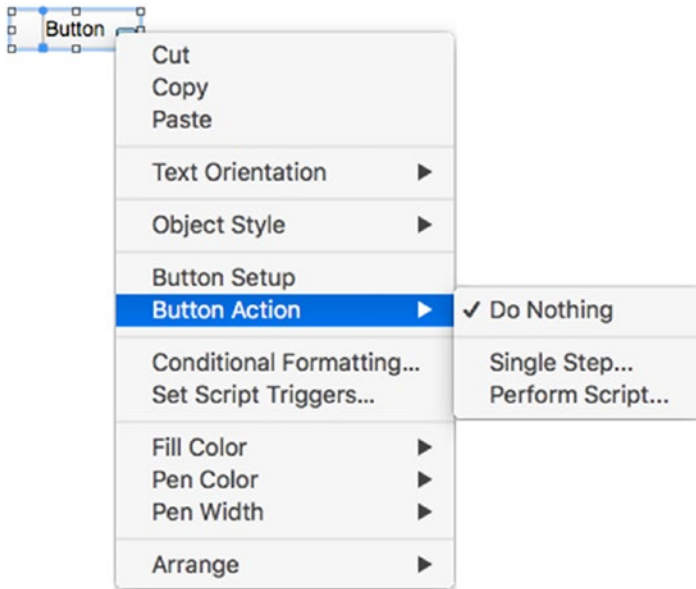


Figure 30-39. The Button Action submenu of a button’s contextual menu

Each option available under the Button Action submenu immediately configures the button and/or opens the relevant dialog while completely bypassing the Button Setup dialog.

- Do Nothing — Quickly removes any assigned action.
- Single Step — Directly opens the Button Action dialog.
- Perform Script — Directly opens the Specify Script dialog.

Popover Button

A *popover button* is a layout object that expands to make additional interface elements available on a miniature layout that temporarily overlaps a portion of the current window. While regular buttons can be configured to perform a specific action, a popover’s action is predetermined; it opens an adjacent, fully customizable interface from which a user can select from many actions or perform various data entry tasks. Popovers prevent visual clutter by allowing interface controls to be grouped and compactly stored out of view while remaining easily accessible with a click when needed.

The expanded pop-up button interface is a small layout that can contain any number and variety of interface elements just like a regular layout. These have many uses including:

- *Button menu* — Allows an array of buttons to be hidden until the user clicks to expand open a menu of all available choices. For example, a menu of print report options, as shown in Figure 30-40.

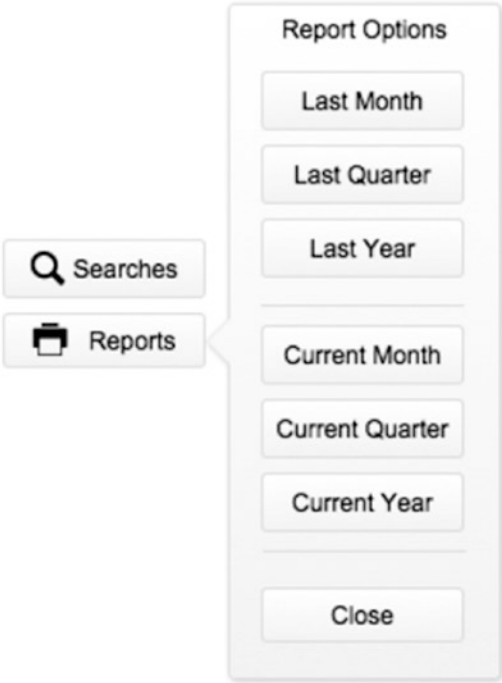


Figure 30-40. An example of an expanded popover button containing an array of buttons for various printing options

- *Task wizard* — An interface that walks a user through a multi-option process with expanded instructional text that would otherwise be too much clutter to include on a main layout.
- *Detailed entry* — A calculated summarization of text from multiple fields that is displayed on a main layout that pops open to make accessible and editable each individual field.

Popovers are created by choosing the Insert > Popover Button menu or clicking on the Button tool in the toolbar and choosing Popover Button from the drop-down menu, as shown in Figure 30-41.



Figure 30-41. The toolbar control used to add a new popover button

Defining the Basic Popover Interface

Once created, a popover button appears on the layout like a new standard button with a default-sized popover interface box open, attached to it, as shown in Figure 30-42.

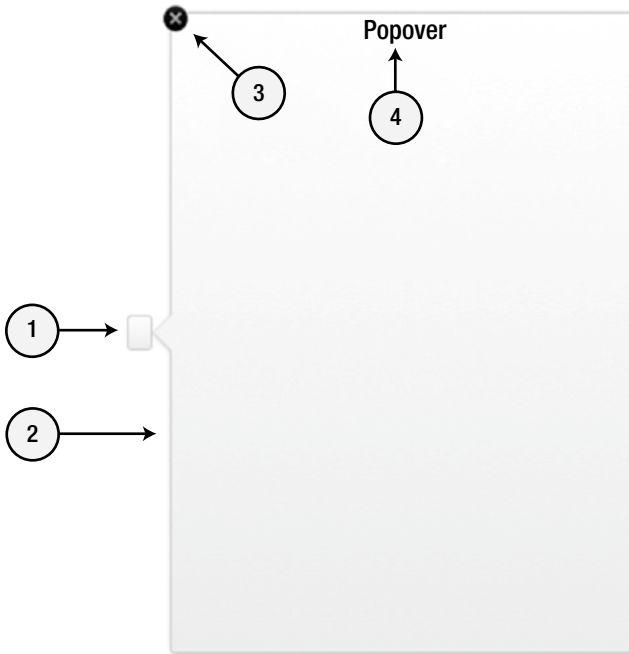


Figure 30-42. A new popover button ready for editing

The following elements make up the popover button interface:

1. *Button* — The button portion remains on the layout when the popover is closed and provides the user access to the expanded interface with a click. As with a standard button, this will be a blank icon until you specify a label and/or icon.
2. *Interface* — The attached box is the interface window that pops open when the user clicks the button. This is where fields and other objects are placed to build the options that will be available to the user when the popover is in its open state.
3. *Close Button* — This button, only visible when editing the popover in layout mode, will close the popover interface, collapsing it back to the button icon.
4. *Title* — This is the optional title of the popover interface window.

■ **Tip** Double-click the popover window to show a gradient control bar and resize handles.

Exploring the Popover Button Setup Options

Popover buttons are configured using the Popover Button Setup dialog, which automatically opens when a new button is created. This dialog, shown in Figure 30-43, can also be opened by any of the following methods:

- Double-click on a popover button or the expanded popover interface.
- With a button selected, click the Format > Popover Button Setup menu item.
- Open the button's contextual menu and select the Popover Button Setup menu item.

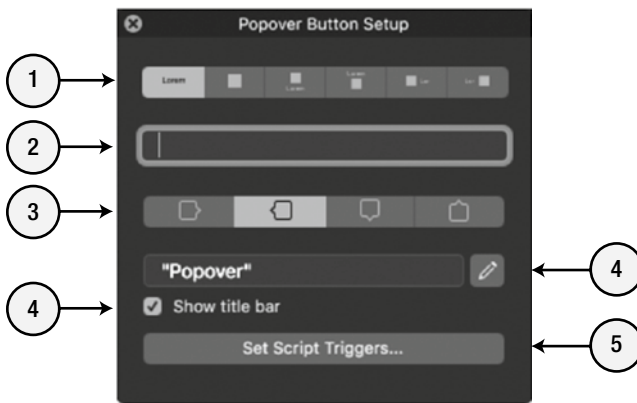


Figure 30-43. A Popover Button Setup dialog

The following elements make up the popover button interface:

1. *Button Labeling Options* — Like a standard button, choose to have the button labeled with text, an icon, or both in a variety of orientations. The default selection is a text label with no icon.
2. *Button Labeling Specification* — Like a standard button, this area dynamically updates the available controls depending on your choice of labeling option above. The default, as shown, allows entry of the button label by typing into the visible text area. As with a button, this area will expand to allow selection of an icon and specification of its size.
3. *Preferred Directional Control* — These icon buttons allow control over the preferred direction the popover interface should open relative to the button icon with the following choices: Left, Right, Top, Bottom. FileMaker will override this choice to ensure that the popover interface does not get drawn off-screen. For example, if the button is placed against the right side of a layout and the Right directional option is selected, the popover may open to the left instead to avoid the interface extending beyond the window's boundary.
4. *Title* — This editable text area contains the title of the popover interface. Click the adjacent pencil icon to open a Specify Calculation dialog to enter a formula that dynamically calculations the title.

5. *Show title bar check box* — This check box controls if the title is displayed in the interface or not.
6. *Script Triggers* — This button offers a convenient way to open the Set Script Trigger dialog for the popover object.

Once configured, the interface of the popover can be customized like a regular layout.

Button Bar

A *button bar* is a layout object that defines a group of interconnected button segments, as shown in Figure 30-44. Button bars can run horizontally or vertically, and each segment can be configured as either a regular push-style button or as a popover button. This allows the placement of different controls together into a single, uniform layout element.

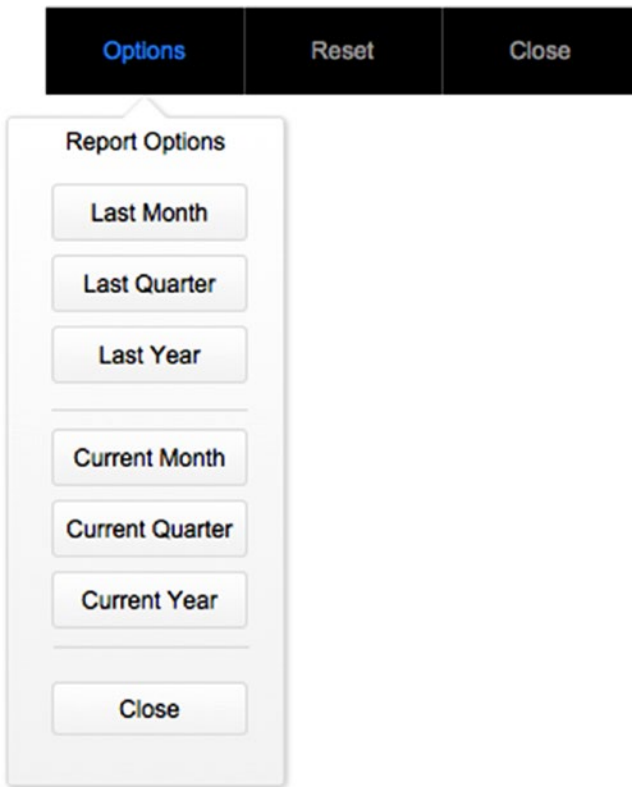


Figure 30-44. An example of a button bar with one popover button (open) followed by two regular push-style buttons

Exploring the Button Bar Setup Options

Button bars are configured using the **Button Bar Setup** dialog, which automatically opens when a new bar is created. This dialog, shown in [Figure 30-45](#), can also be opened by any of the following methods:

- Double-click on a button bar button or the expanded popover interface of a button bar segment.
- With a bar segment selected, click the **Format > Button Bar Setup** menu item.
- Open the button’s contextual menu and select the **Button Bar Setup** menu item.

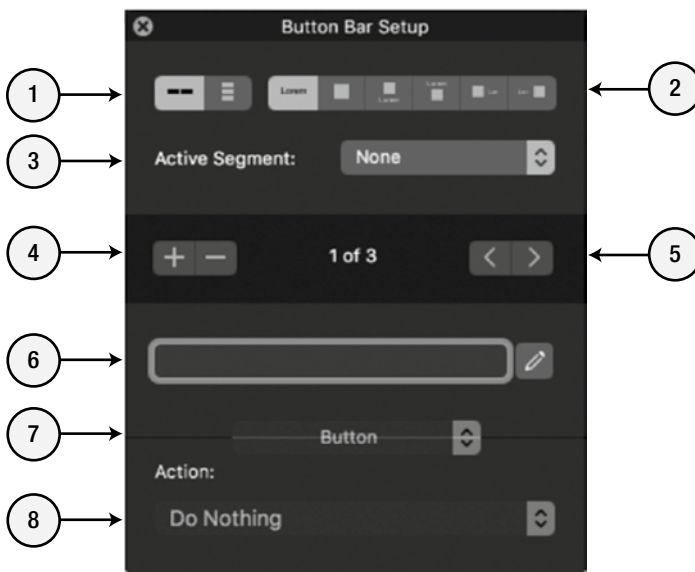


Figure 30-45. The **Button Bar Setup** dialog

This dialog combines the options of the **Button Setup** and **Popover Button Setup** dialogs with controls specific to the button bar.

The top portion of the dialog contains controls pertaining to the overall object. These include:

1. *Bar Orientation Control* — These two icon buttons control the direction of the bar segments allowing a choice of **Horizontal** or **Vertical**.
2. *Button Labeling Options* — Like a standard button, the button can be labeled with text, an icon, or both in a variety of orientations. The default selection is a text label with no icon.
3. *Active Segment* — This pop-up menu specifies which segment will be formatted as “active” by default in **Browse** and **Find** modes, with a default selection of **None**. The menu includes a list of all the defined segments in the bar, listed by segment number or by button label, if a label has been assigned. It includes a **Specify** option that opens a **Specify Calculation** dialog allowing entry of a formula to determine the active segment by returning the **Object Name** of the desired segment (from “**Position Settings of the Inspector**,” [Chapter 29](#)).

4. *Segment Controls* — Button bars are created with a default of three segments. These buttons allow you to add or remove segments as follows:
 - The + button will add a new segment after the currently selected segment.
 - The - button will delete the currently selected segment.
5. *Current Segment Navigation* — These buttons allow you to move back and forth between the segments to select one for editing.

The bottom portion of the setup dialog contains controls that vary contextually based on the currently selected segment. These include:

6. *Button Labeling Specification* — Like a standard button, this area dynamically updates the available controls depending on your choice of labeling option above. By default, it allows for the entry of the button text in the visible field but will expand to allow selection of an icon and specification of its size. The values displayed apply only to the selected segment.
7. *Segment Type* — This pop-up menu allows specification of the type of button for the current segment, offering a choice of Button or Popover Button.
8. *Action Menu & Specification* — This area dynamically updates the available controls depending on the type of segment, displaying Button or Popover Button interface controls accordingly.

Converting an Object into a Button

In addition to the native button objects available for layouts, FileMaker allows any object on a layout to be converted into a push-style button. Select the object and click the **Format > Button Setup** menu or the contextual menu option of the same name to open a **Button Setup** dialog, as shown in Figure 30-46.

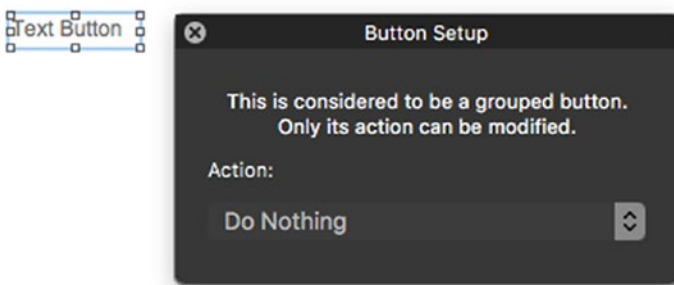


Figure 30-46. The *Button Setup* dialog for a non-button layout object, in this case as text object

The dialog only allows configuration of the button action with the same options as a regular button:

- Do Nothing
- Perform Script
- Single Step

Working with Panel Controls

A *panel control* is a layout object that contains multiple groups of layout objects, organized in separate panels, which can be alternatively displayed within a single framed region of a layout. Panels save space by allowing a single delimited area of a layout to be used for multiple purposes, one at a time as required by a user.

FileMaker has two object types that manage multiple panels:

- Tabs control
- Slide controls

Tab Control

A *tab control* is a layout object that makes multiple interface panels accessible one at a time by means of file folder tabs, like the example shown in Figure 30-47. The object can be configured with any number of separate panels, each represented by a labeled tab. When the user clicks on one of the available tabs, the corresponding panel activates and its layout elements are displayed in the rectangular area below while all other panels are simultaneously obscured from view. Each panel is like a mini-layout and can contain any number and type of layout objects arranged as they can on a regular layout. A tab panel can even contain another tab object, creating a hierarchical tab structure.



Figure 30-47. An example of a tab control with four tabs, actively displaying the second tab

Creating a New Tab Control

A new tab control can be added to a layout by:

- Selecting the Insert > Tab Control menu item will insert a new tab control at a default location and size.
- Selecting the Tab Control tool in the toolbar, shown in Figure 30-48, and then clicking and dragging the cursor to define the position and size of the new tab object.

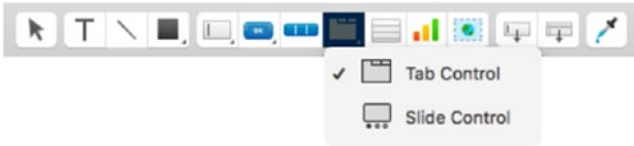


Figure 30-48. The Tab Control tool in the toolbar

Exploring the Tab Control Setup Dialog

Tab controls are configured using the Tab Control Setup dialog, which automatically opens when a new tab control is created. This dialog, shown in Figure 30-49, can also be opened by any of the following methods in Layout mode:

- Double-click anywhere on the background of a tab control.
- With a tab control selected, click the Format > Tab Control Setup menu item.
- Open the tab control's contextual menu and the Tab Control Setup menu item.

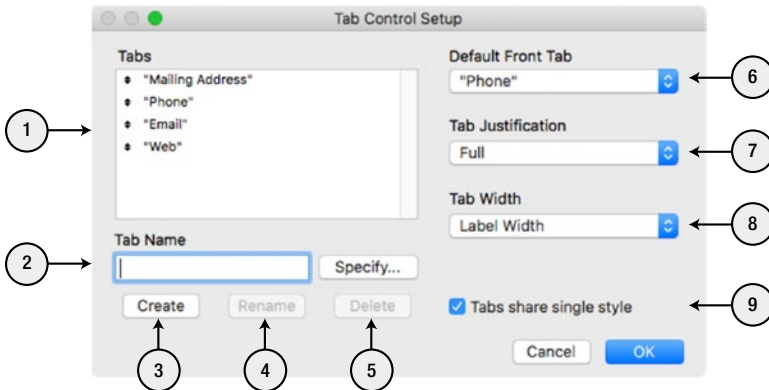


Figure 30-49. The Tab Control Setup dialog

The following controls are available on the `Tab Control Setup` dialog:

1. `Tabs` — This area lists the name of every tab panel currently defined for the control. Click and drag these up or down in the list to rearrange the order.
2. `Tab Name` — This text area is used to type a tab name when creating a new panel and to edit the name of the selected tab. Click the adjacent `Specify` button to open a `Specify Calculation` dialog into which a formula can dynamically calculate the tab name.
3. `Create` — This button will create a new tab with the name entered above. Tab names do not need to be unique.
4. `Rename` — This button will rename the selected tab with the name entered above.
5. `Delete` — This button will delete the selected tab(s) without warning, although the change doesn't take place on the layout until you click the `OK` button and clicking `Cancel` will reverse any changes.
6. `Default Front Tab` — This pop-up menu lists every defined tab name. The selected tab will be the default active tab when navigating to a record or creating a new record.
7. `Tab Justification` — This pop-up menu allows control over the justification of the tab labels on the control from the available choices:
 - `Left`
 - `Center`
 - `Right`
 - `Full`
8. `Tab Width` — This pop-up menu contains the following options for control over the width of the tabs:
 - `Label Width` — Set the width of each tab individually based on the width of the label name.
 - `Label Width + a Margin of` — Set the width of each tab individually based on the width of the label name with a specifiable margin.
 - `Width of Widest Label` — Set the width of each tab uniformly to the width of the longest label name.
 - `Minimum of` — Set the width of each tab individually based on the width of the label name while staying above a specifiable minimum measurement.
 - `Fixed Width of` — Set the width of each tab uniformly to a specifiable fixed width.
9. `Tabs share single style` — Check this box to automatically apply any appearance style settings made to one tab panel to every other tab panel, maintaining a uniform design.

Slide Control

A *slide control* is a layout object that makes multiple interface panels accessible one at a time by means of swiping left or right on an iOS device with FileMaker Go or by clicking a corresponding navigation dot, like the example shown in Figure 30-50.



Figure 30-50. An example of a slide control

Exploring the Slide Control Setup Dialog

Tab controls are configured using the Slide Control Setup dialog, which automatically opens when a new slide control is created. This dialog, shown in Figure 30-51, can also be opened by any of the following methods in Layout mode:

- Double-click anywhere on the background of a slide control.
- With a slide control selected, click the Format > Slide Control Setup menu item.
- Open the slide control’s contextual menu and select the Slide Control Setup menu item.



Figure 30-51. The Slide Control Setup dialog

The following controls are available on this dialog:

1. *Panel Controls* — Slide controls are created with a default of three panels. These buttons allow you to add or remove panels as follows:
 - The + button will add a new panel after the currently selected panel.
 - The - button will delete the currently selected panel.
2. *Current Panel Navigation*— These buttons allow you to move back and forth between the panels to select one for editing.
3. *Options* — These check boxes control whether the slide control will react to swipe gestures on an iOS device and whether the navigation dots are visible or not.
4. *Navigation Dot Size* — Modify the size of the navigation dots by typing in a new point size or clicking the up or down arrows to adjust the size one point at a time.

Working with Portals

A *portal* is a layout object that displays a list of related records from another table occurrence as a vertical stack of rows with an optional scroll bar. Each *portal row* represents one related record and can be designed to include related fields with any combination of other layout objects. For example, a *Company* layout might include a portal that lists employees from a *Contact* table, like the example shown in Figure 30-52. Users can then view, edit, or click a button to navigate to a specific contact record.

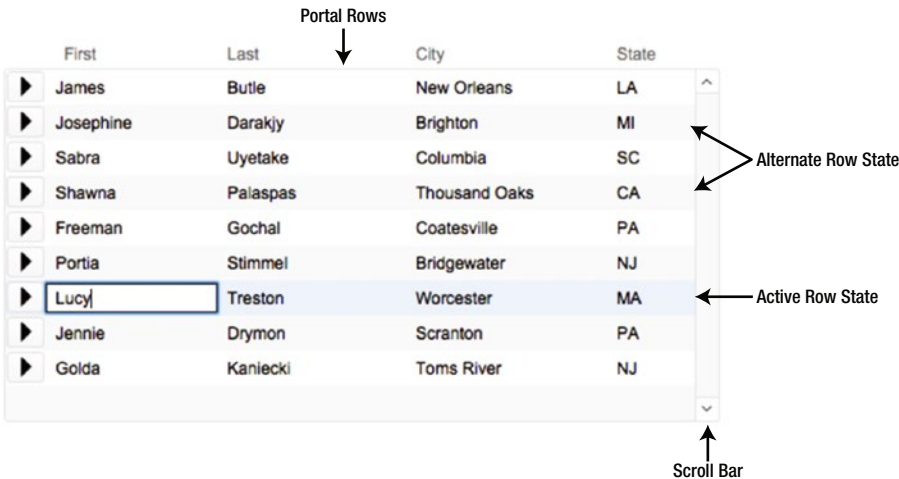


Figure 30-52. An example of a portal listing every *Contact* record related to the current *Company* record

Illustrating How Portals Work

A portal is essentially a layout manifestation of a relationship between two table occurrences. Relationships *integrate* two tables, allowing data exchange between fields on either side. Portals essentially provide the layout equivalent of this type of functionality, allowing a user to view records and perform data entry tasks on records through a relationship.

When designing a layout for one table, any other table occurrence that is related to that table can be used as the record source for a portal. In the example illustrated in Figure 30-53, a database contains two table definitions, Company and Contacts. A second table occurrence of the Contacts table called Company | Contacts is created and then related to the Company table so that, from a company record the relationship will provide a list of employees for the current company. This table occurrence is used as the data source for a Contacts portal placed on the Company Form layout. As a user navigates through the list of companies, the portal will update to show only those contacts related to the current company record.

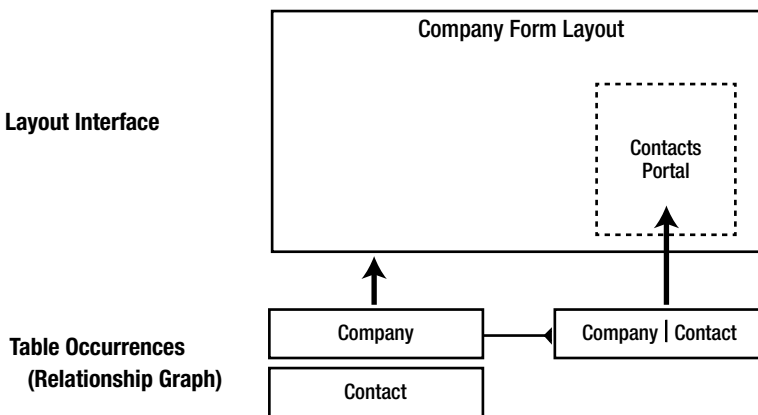


Figure 30-53. An illustration of a portal showing related contacts placed on a company layout

In this example, we can think of Company as the parent table and Contact as the child table because we are viewing a company record on a company layout and seeing one or more linked contacts through the relationship.

Exploring the Portal Setup Dialog

Portals are configured using the Portal Setup dialog, which automatically opens when a new portal is created. This dialog, shown in Figure 30-54, can also be opened by any of the following methods in Layout mode:

- Double-click anywhere on the background of a portal.
- With a portal selected, click the Format > Portal Setup menu item.
- Open the portal’s contextual menu and select the Portal Setup menu item.

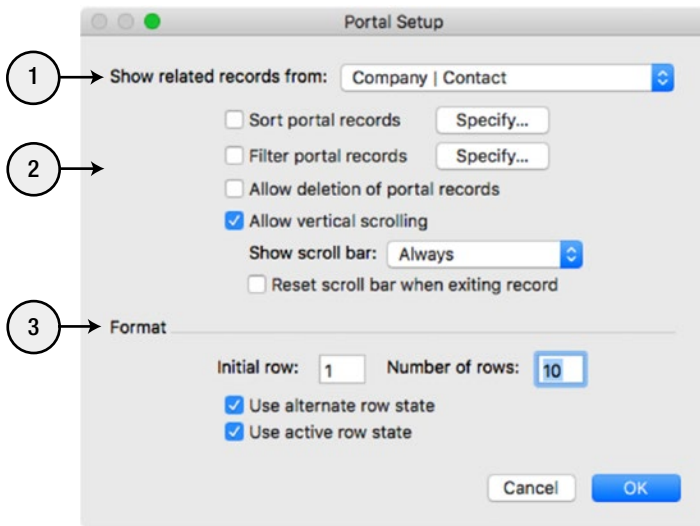


Figure 30-54. The Portal Setup dialog is used to configure the settings for a portal

The following options are available on the Portal Setup dialog:

1. Show related records from — This pop-up menu contains a list of every table occurrence defined in the database organized by their relationship status to the current layout's tables. Any related table can be selected and will become the data source for the portal.
2. The following check box options control the information displayed and properties of the portal:
 - Sort portal records — Opens a Sort Records dialog allowing the selection of fields that control how portal rows will be sorted and will override any sort order defined by the selected relationship.
 - Filter portal records — Opens a Specify Calculation dialog for a formula that can specify criteria used to determine which related records will be included (discussed later in this chapter).
 - Allow deletion of portal records — Check this option to allow a user to type the Delete key to delete a selected portal row and the related record it represents. If left unchecked, users will only be able to delete related records if a button and/or script is set up to explicitly do so.
 - Allow vertical scrolling — Check this option to allow the portal to become scrollable when the number of related records exceeds the number of rows visible.
 - Show scroll bar — This menu allows control over when the scroll bar will be visible, either always or only when a user is actively scrolling.
 - Reset scroll bar when exiting record — Check this option to automatically scroll back to the first row when the record is committed.

3. The following row formatting options are available:
 - **Initial row** — This number indicates where in the range of related records to begin displaying in the portal with any prior records excluded.
 - **Number of rows** — This number controls how many rows should be included in the portal at its minimum size. All related records after the **Initial row** will always be included in the portal, but some may be out of the scroll range, not initially visible and only accessible if scrolling is enabled. If the portal is configured for auto-sizing (Chapter 29) then the number of rows will increase as the window height is increased.
 - **Use alternate row state** — Check this option to format portal rows with an alternating fill color to help delineate them and improve a user's visual clarity when glancing along the objects included within.
 - **Use active row state** — Check this option to apply special formatting to help visually indicate which row is currently active.

Adding Objects to Portal Rows

In Layout mode, a portal's first row will appear different than the other rows. This indicates that the first row is the template area where you can add fields, buttons, and other objects; and the remaining rows will be rendered with the same objects, one for each related record.

The first row is like a mini-layout where you can add any object except other portals. Simply drag the item in, position, size, and place it as you would anywhere else on a layout. Just make sure that the items stay within the boundary of the row or they will either disappear completely or appear floating over the portal instead of repeated down the rows when rendered in Browse mode.

Creating Records in a Portal

When viewed in Browse mode, each record in the related table occurrence with the appropriate match field values will appear in the portal as a row. Adding new rows to the portal requires the creation of new records in that related table, which can be accomplished by one of two methods:

- Configuring the portal to allow direct record creation.
- Creating a related record linked to the parent record.

Once configured, each of these methods can be employed by both users and scripts.

Configuring the Portal to Allow Direct Record Creation

To allow the creation of records directly within a portal, enable the **Allow creation of records in this table via this relationship** check box in the **Edit Relationship** dialog (Chapter 9) on the side of the relationship representing the table occurrence that provides records for the portal. This will cause the portal to always display a blank row, as shown in Figure 30-55.

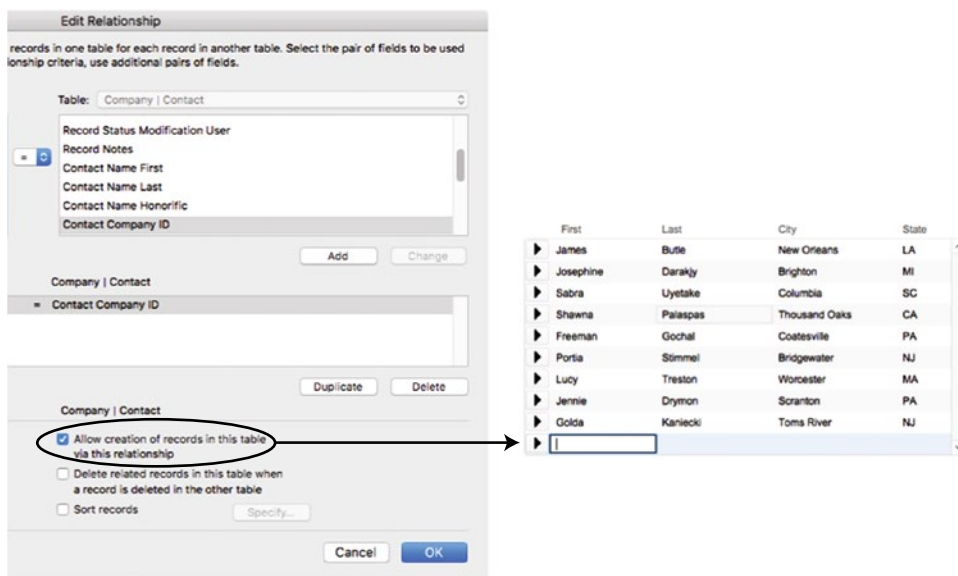


Figure 30-55. The check box in the *Edit Relationship* dialog enables a blank portal row that auto-creates new records

To create a new record, a user or script must enter a field within the blank row, begin typing, and then move to another field or commit the record. This will *instantly* cause the following events to occur:

- A new record will be created in the remote table with the field value(s) entered by the user.
- The new record will automatically populate each match field with the appropriate value to cause it to be related to the currently viewed parent record.
- The new record will appear in the portal.
- Another blank row will appear for the creation of additional new records.

The blank row is *only* an interface feature, implying the option for creating a new record. However, it does not actually exist in the related table until data is entered. This means that it will be *programmatically ignored* and will not affect functions that summarize related records such as Count, List, Max, Min, Sum, etc.

■ **Caution** Using this feature may be confusing for some users since the empty new record row will appear to them like a related record with empty fields. This may cause them to either attempt to delete the empty row thinking it is leftover data; *or* not attempt to delete an *actual* leftover record thinking it is the new record row. For this reason, many professional developers do not enable this feature in their solutions and prefer to write a script to create a related record.

Creating a Related Record Linked to the Parent Record

The other method for creating a new portal row can be achieved by:

1. Navigating to a layout for the table occurrence that provides records for the portal.
2. Creating a new record.
3. Entering the appropriate value(s) in the match field(s) to establish the link to the parent record.
4. Returning to the original layout displaying the portal.

Once these steps are completed the record will appear in the portal on the parent record.

Although these steps can be performed manually by a user, due to the number of steps and complexity involved, this process should be encoded into a script. In addition to the burden of time, performing the task manually increases the chance a user may forget to fill in an essential match field or enter the wrong value. In either case, this would result in the new record being linked to the wrong record and lost somewhere in the data.

A script linked to a button near the portal is a far better option. At the click of the button, the script would quickly load the match field value(s) from the parent into variables, navigate to the appropriate layout for the related table occurrence, create the new record, enter the match field value(s), return to the portal layout, select the new portal row and then return control to the user allowing them to begin typing into the new row that was created.

■ **Note** An example of each of the scripts described in this section appear in Chapter 35, “Scripting Portal Functions.”

Deleting Portal Rows

Deleting a portal row is accomplished by deleting the related record displayed in the row. Like adding rows to a portal, there are two options available for giving a user this ability.

- Configuring the portal to allow direct record deletion.
- Deleting a related record directly from the remote table.

Configuring the Portal to Allow Direct Record Deletion

The Portal Setup dialog contains a setting to Allow deletion of portal records. When this feature is enabled, a user can select a portal row (without clicking into a field) and type the Delete key to delete the related record and remove it from the portal display. FileMaker will present a confirmation dialog asking if the user wants to continue with the deletion, as shown in Figure 30-56. If they click Delete, the portal row and the related record it represents will be immediately deleted.

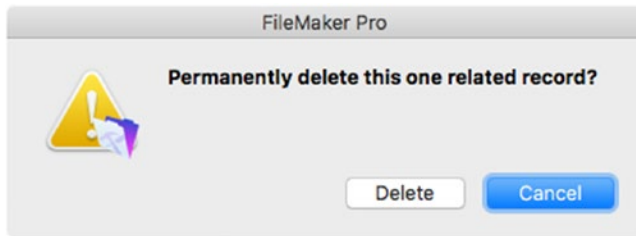


Figure 30-56. The dialog confirming the deletion of a portal row

■ **Caution** The default message in the dialog is vague and has the potential to be confusing to the users. Configuring the portal to Use active row state may help since the selected row will be highlighted. However, the non-specific message is still far from ideal. To create a more professional interaction, turn off the direct delete option and put a Delete button or icon into the portal row that will handle the process with a more detailed dialog message.

Deleting a Related Record Directly within the Remote Table

The other method for deleting a portal row can be achieved by:

1. Navigating to a layout for the table occurrence that provides records for the portal.
2. Finding the record to be deleted.
3. Deleting the record.
4. Returning to the original layout displaying the portal.

Once these steps are completed the record will be deleted and removed from the portal on the parent record.

This process *can* be performed manually, however, as mentioned when adding a record to a portal, it is *far* better to create a script and link it to a Delete button that can be placed within the portal row, as shown in Figure 30-57.

First	Last	City	State	
▶ James	Butle	New Orleans	LA	✕
▶ Josephine	Darakjy	Brighton		✕
▶ Sabra	Uyetake	Columbia	SC	✕
▶ Shawna	Palaspas	Thousand Oaks	C	✕
▶ Freeman	Gochal	Coatesville	PA	✕
▶ Portia	Stimmel	Bridgewater	NJ	✕
▶ Lucy	Treston	Worcester	MA	✕
▶ Jennie	Drymon	Scranton	PA	✕
▶ Golda	Kaniecki	Toms River	NJ	✕
▶				

Figure 30-57. A portal with a Delete button added to the right side to allow a user to click to run a script that will delete a specific record

The button can be linked to the Delete Portal Row script step, but that is essentially the same function as them typing Delete and will have the same vague dialog message. Instead, write a script that loads the Record ID field of the selected row into a variable; navigates to a layout for the table occurrence providing records for the portal; performs a search for the record; presents a warning dialog; and, if the user confirms their intent, deletes the record before returning to the original layout. Since you are creating this script yourself, you can control the dialog message, providing a much clearer warning to the user like the one shown in Figure 30-58.

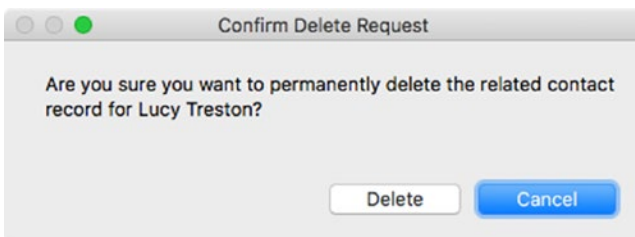


Figure 30-58. An example of a robust confirmation dialog for a script when deleting a portal row

Filtering Portal Records

Portal filtering allows a formula to determine which of the available related records are actually displayed in the portal. It can be driven by field values or other contextual conditions. The popular filtering setup described in this section involves a search field into which users can type some criteria that specifies a desired subset of related records.

A Search Filter field would be added to the layout near the portal, like the one shown in Figure 30-59. When the user types into the field and commits the record, the list of rows would be trimmed down to only those with matching values.

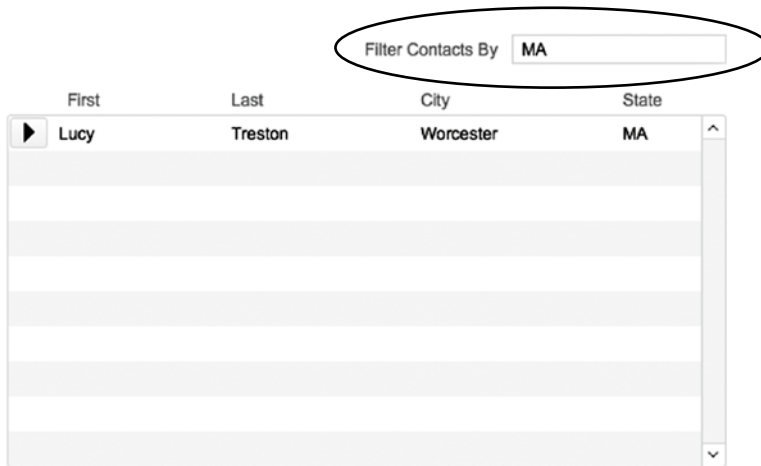


Figure 30-59. An example of a basic portal filtering field placed near a portal to filter records based on the user's entry

■ **Caution** Filtering applies *only* to related records *displayed* in a portal and does not affect the actual relationship. Therefore, any field or script calculations that access records through the relationship will continue to see all records even when the layout displays a filtered set.

Setting Up Portal Filtering

Setting up portal filtering is a relatively quick process. For example, creating a simple filter by the Contact Address State field can be accomplished by following these steps:

1. Create a text field in the parent table, represented by the layout upon which the portal will be displayed. In our ongoing example, this would be the Company table. For this example, the field name will be Company Contact Portal Filter.
2. Place the field on the layout just above the portal with a label that makes clear its function, as shown in the example in Figure 30-59.
3. In the Portal Setup dialog, click the Specify button adjacent to the Filter portal records option to open a Specify Formula dialog.
4. The formula must indicate when a related record in the Contacts table should appear in the portal based on what the user has typed into the filter field.
5. Then save the formula and close the Portal Setup dialog.

■ **Tip** To have the filter field apply to all records in the parent table and to avoid the record locking (showing busy for other users) use global storage on that field. Once the user enters filtering criteria, it will apply for any record during the session until they change the criteria.

Writing a Basic Portal Filter Formula

A portal filter formula is made up of one or more conditional statements that evaluate to a true or false result. Every related record is evaluated individually; if it yields a true result, it appears in the portal, if false, it is hidden.

The filter formula constrains the list of related records and already assumes the record matching criteria of the relationship itself. As a result, the filter formula:

- does not need to re-specify that criteria, which is a given
- cannot extend beyond the relationship's criteria to include additional records

If you don't want to require filtering criteria, the formula *must* include an option for situations when no criteria are entered in the filter field. In this case, the portal should continue to display *all* related records. This can be accomplished by starting the formula with the following code, indicating that related records should be included when:

```
Company::Company Contact Portal Filter = ""
```

After this, an or operator followed by a second condition can begin adding filtering criteria. Using the `PatternCount` function, we can specify a partial or full match between the `Contact Address State` field and the text the user has entered in the filter field, expanding the formula to the following:

```
Company::Company Contact Portal Filter = "" or
PatternCount (
    Company | Contact::Contact Address State ;
    Company::Company Contact Portal Filter
) > 0
```

Save the formula and the Portal Setup, then test the function. When a state abbreviation is typed into the filter field, the portal rows should only include those records matching the text you enter.

■ **Note** For the portal to update after typing into the filter field, the record must be committed. To dynamically update the results, use the `ObjectModify` script trigger (Chapter 37) to run a script that commits the record and re-enters the filter field after each keystroke.

Expanding the Filter Formula to Include Multiple Match Fields

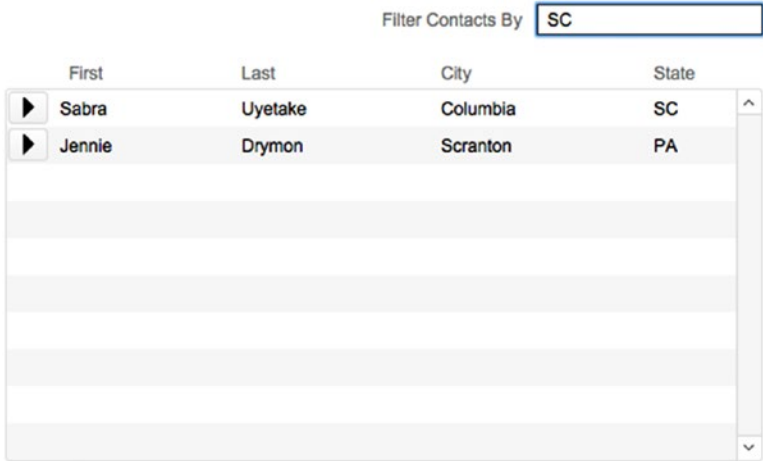
The filter formula can be expanded to include additional criteria with the appropriate operator. This includes filtering based on any combination of the following:

- Multiple fields in the parent table in addition to or instead of a search filter field.
- Multiple fields in the child table based on a single search filter field in the parent or multiple fields.
- Multiple criteria using combinations of fields using an and operator.

In the Contact Portal example, the formula can be expanded to find matches in multiple Contact fields based on the text in the single filter field like this:

```
Company::Company Contact Portal Filter = "" or
PatternCount (
    Company | Contact::Contact Name First ;
    Company::Company Contact Portal Filter
) > 0 or
PatternCount (
    Company | Contact::Contact Name Last ;
    Company::Company Contact Portal Filter
) > 0 or
PatternCount (
    Company | Contact::Contact Address City ;
    Company::Company Contact Portal Filter
) > 0 or
PatternCount (
    Company | Contact::Contact Address State ;
    Company::Company Contact Portal Filter
) > 0
```

Now, in the example shown in Figure 30-60, typing “SC” in the filter field will result in a match of two records: one based on the state and another based on a partial match in the city.



The screenshot shows a user interface for filtering contacts. At the top, there is a text input field labeled "Filter Contacts By" containing the text "SC". Below this is a table with four columns: "First", "Last", "City", and "State". The table contains two rows of data. The first row has "Sabra" in the First column, "Uyetake" in the Last column, "Columbia" in the City column, and "SC" in the State column. The second row has "Jennie" in the First column, "Drymon" in the Last column, "Scranton" in the City column, and "PA" in the State column. There are also several empty rows below the second row, and a vertical scrollbar on the right side of the table.

First	Last	City	State
Sabra	Uyetake	Columbia	SC
Jennie	Drymon	Scranton	PA

Figure 30-60. The portal filtered by multiple fields in the related table

Enhancing the Search Field

Once configured, consider improving the search field interface to more closely mirror what is seen throughout the operating system. The example shown in Figure 30-61 includes the following design improvements over the previous examples:

- The label has been replaced with Placeholder text of “Search by,” which is embedded within the field.
- The corners of the field are rounded.
- A button icon is added on the right that, when clicked, uses the Clear function to delete the search criteria.



Figure 30-61. An improved search field interface design

Working with Web Viewers

A *web viewer* is a layout object that use the operating system’s web browser technology to display a web page directly on a database layout. Viewers can be configured to display a static page or to allow user interactions. When a layout is loaded, the viewer can display a page generated by:

- A choice of a handful of standard web address formula such as Google web search, Google maps, FedEx, or Wikipedia.
- A web address pulled from a database field.
- A hard-coded web address pointing to any website.
- Custom HTML code created by a FileMaker calculation formula that may include hard-code elements, field data, and even images stored in fields.

Scripts can modify the address or data of the page displayed in a web viewer.

Web viewers will display fine when a window is displaying a Form view or List view; Table view will only display a web viewer placed in the header or footer. They can’t be placed in portals and shouldn’t be placed behind other objects.

■ **Note** Custom web addresses must begin with the correct scheme such as `http://`, `https://`, `ftp://`, or `file://`.

Exploring the Web Viewer Setup Dialog

To add a web page to a layout, select the **Insert > Web Viewer** menu item or the **Web Viewer** tool from the toolbar and click and drag on the layout to define the area for the new object. The **Web Viewer Setup** dialog, shown in Figure 30-62, will appear.

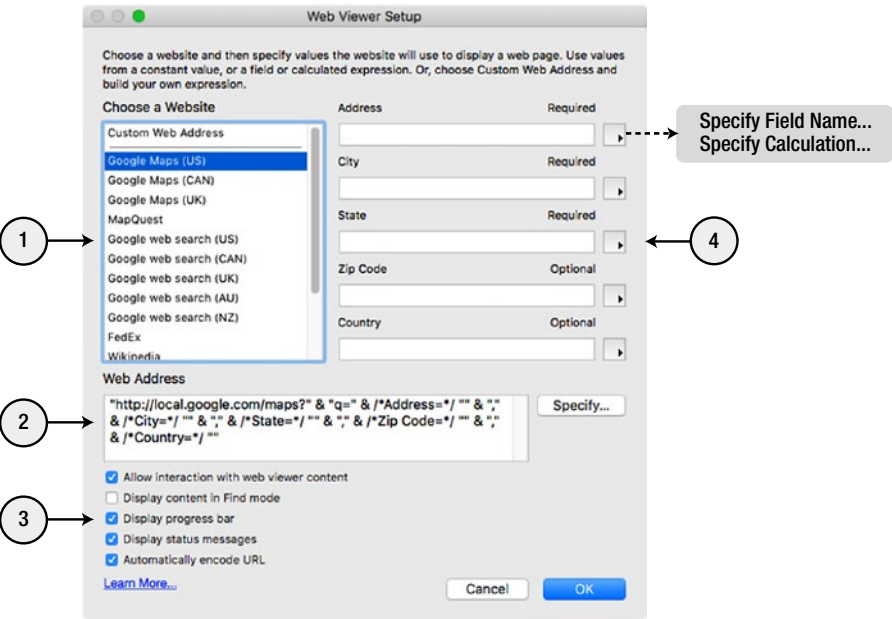


Figure 30-62. The Web Viewer Setup dialog

The following controls are available in the Web Viewer Setup dialog:

1. Choose a Website — This list displays all the available options for generating a web address. Choose from the list of templates for standard data-driven websites or select Custom Web Address to write your own URL or HTML calculation formula.
2. Web Address — This text area displays the address or formula for generating an address that will be displayed in the web viewer. The predefined options will insert a ready-made calculation with links to contextual website parameter fields on the top right of the dialog. Custom addresses or formulas can be entered directly into the text area or into a Specify Calculation dialog by clicking the adjacent Specify button.
3. Web Viewer Options — This set of check boxes control the following features of the viewer:
 - Allow interaction with web viewer content — Check this box to allow the user to interact with the displayed web page like they would in any standard web browser.
 - Display content in Find mode — Check this box to cause the web viewer to continue displaying content when the window is placed into Find mode.
 - Display progress bar — Check this box to include a progress bar in the footer showing the percentage of completion for loading the page.

- **Display status messages** — Check this box to have the viewer include a text footer that displays the loading status of the page.
 - **Automatically encode URL** — Check this box to allow FileMaker to automatically percent-encode any special characters contained in the address specified. For example, a space will be converted to %20 automatically if this box is checked.
4. **Website Parameters** — This set of input controls will display a set of input options for the predefined websites, which will be automatically inserted into the URL when the web page is rendered in the viewer. You can type a value directly into a field or click the adjacent button and choose from one of two options:
- **Specify Field Name** — Choosing this option will open a **Specify Field** dialog and allow you to select a field that will provide the required input value.
 - **Specify Calculation** — Choosing this option will open a **Specify Calculation** dialog and allow you to enter a custom formula to determine the required input value.

Interacting with Web Viewer Content

When the **Allow interaction with web viewer content** option is enabled, a user can interact with a web page as they would in any other browser. They can click links to navigate to other pages and should be able to view content such as PDF documents and movies. However, it is important to remember that the viewer isn't really intended to act as a full-featured web browser and there may be some limitations.

When enabled, several contextual menus will allow interaction with content as well as the page itself. While holding down the **Control** key, click on a:

- **Page** — to gain access a menu with two options: **Reload** the web page or go **Back** to the previously displayed page.
- **Text** — accesses a menu of standard operating system text functions. The macOS version includes the options to **Look up** the word or phrase in the dictionary, perform a **Google** search with the text, **Copy**, a submenu of **Share** options, a submenu of **Speech** options, and access to any available text **System Services**.
- **Image** — accesses a menu of image functions like what you would see in a web browser. The macOS version includes an option to open the image in a new window, **Download** the image, **Copy** the image, and include a submenu of **Share** options.
- **Link** — accesses a menu of options including **open the link**, **open the link in a new window**, **download linked file**, **copy link**, a submenu of **Share** options, and any available text **System Services**.

■ **Note** Choosing to open an image or link into a new window will leave FileMaker and open in your computer's default web browser application.

Building a Web Page Using Data from Fields

In addition to providing a web address, a web viewer formula can return HTML code to dynamically build a custom page. FileMaker uses the *data universal resource identifier* (URI) scheme, which is a standard method of including data within the code of a web page as if it were external resources. In FileMaker, the HTML code for the page itself is an instance of the data scheme as are any images embedded within it.

The data URI is expressed with the following formula (with square brackets indicating optional elements):

```
data:[<media type>][;base64],<data>
```

This formula contains the following elements:

- `data:` — This indicates the scheme being used.
- `<media type>` — The optional media type indicates the type of material contained in the data. A web page would use `text/html` while an image would use `image/<type>`, for example, `image/png`. If no media type is specified, the data will be assumed to be `text/plain` although it appears that FileMaker still interprets the data as `html`.
- `;base64`— The optional base64 extension, delimited from the media type with a semicolon, is used to indicate that the data content is binary data, which is encoded in ASCII format using the Base64 binary-to-text encoding scheme.
- `<data>` — The sequence of characters, separated from the preceding values by a comma, that make up the content being described, HTML code or Base64 image data. This text can contain ASCII characters for the lowercase and uppercase letters of the alphabet but they must be percent-encoded.

■ **Note** The `GetAsURLEncoded` function will percent-encode text (Chapter 13).

Creating a Hello World Web Page

To get started creating a formula to generate a custom web page, start with this simple Hello World example:

```
"data:text/html,
<html>
<head>
</head>
<body>
<h1>Hello World</h1>
</body>"
```

The page code can include a `<style>` tag to control text formatting with cascading style sheets (CSS). This simple example modifies the color of the `h1` style:

```
"data:text/html,
<html>
<head>
<style>
h1 {
```

```

color: green;
}
</style>
</head>
<body>
<h1>Hello World</h1>
</body>"

```

The page code can include a `<script>` tag to include JavaScript functions. This simple example will open an alert dialog when the page loads:

```

"data:text/html,
<html>
<head>
</head>
<body>
<script>
alert('Hello World!')
</script>
<h1>Hello World</h1>
</body>"

```

■ **Tip** Remember when writing HTML code for a web viewer that you are doing so within a FileMaker calculation formula. All text must be enclosed in quotation marks, and any quotes within that text must be escaped with a preceding backslash. To avoid this, the entire HTML code can be stored in and loaded from a field.

Including Text Fields in a Web Page

Since the code for the web page is written in a Specify Calculation dialog using a formula that returns text inserting fields is done the same as in any formula (Chapter 12). Place a field reference into the formula outside of the quoted text:

```

"data:text/html,
<html>
<head>
</head>
<body>
<h1>" & Company::Company Name & "</h1>
</body>"

```

Including a Container Field Image in a Web Page

Images that exist on the Web can be included by inserting the URL in an <image> tag. However, to include an image from a field, use the Base64Decode function (Chapter 18) to convert the image in a container field. This can be inserted into the code as follows:

```
"data:text/html,
<html>
<head>
</head>
<body>
<h1>" & Company::Company Name & "</h1>
<img src='data:image/imagemac;base64,'" & Base64Encode ( Company::Company Logo ) & "'>
</body>"
```

Working with Charts

A *chart* is a layout object that draws a graphical representation of data in one of several popular charting formats, shown in Figure 30-63.

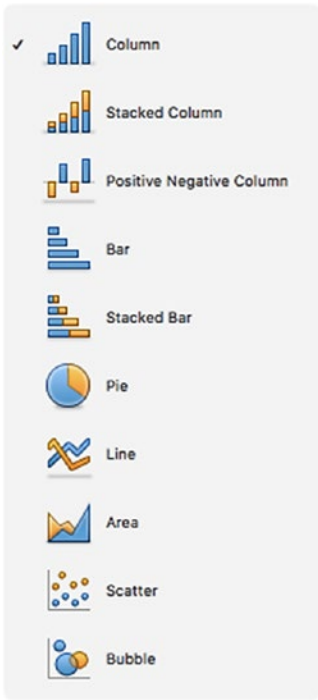


Figure 30-63. The available chart types in FileMaker

Charts are configured within a complex Chart Setup dialog, shown in Figure 30-64, which opens whenever a new chart object is inserted onto a layout. This dialog can be reopened for an existing chart object by any of the following methods:

- Double-click on the chart object.
- Open the chart object’s contextual menu and select the Chart Setup option.
- Select the chart object and choose the Format > Chart Setup menu item.

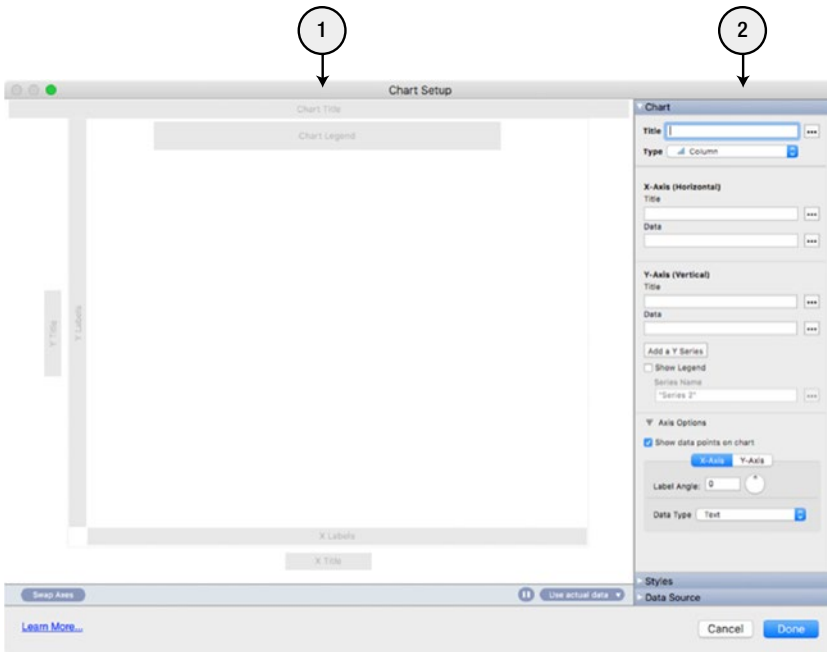


Figure 30-64. The Chart Setup dialog

The Chart Setup dialog is broken up into the following two sections:

1. *Preview* — This area will show a preview of the chart based on the configured settings and the current window’s context (table, found set, sort order, etc.).
2. *Controls* — This area contains all the controls used to define and style the chart. It contains three toggle sections that display tools and settings contextually based on the chart type selected and other options:
 - **Chart** — Set the chart title, type, axis data information (field reference or calculation results), legends and more.
 - **Styles** — Set the chart style, color scheme, background, gridlines, legend positions, borders, and more.
 - **Data Source** — Offers a choice for the source of the axis data from Current Found Set, Current Record (delimited data), or Related Records.

The charting features and interface controls are too numerous to cover in complete detail so we will explore simple examples using two chart types. With an understanding of general charting methodology and features from other popular applications such as Microsoft Excel, it should be relatively easy to explore the interface and figure out how to create additional charts.

Creating Charts

Charts can be created using data from the found set of records, a group of related records, or from calculated data.

Creating a Chart Using Calculated Data

Using calculated data to generate a chart will draw information from either hard-coded calculated values or fields from the current record. Let's work through an example of creating a pie chart using hard-coded values.

To get started, insert a chart onto a layout, and then configure the settings as shown in Figure 30-65.

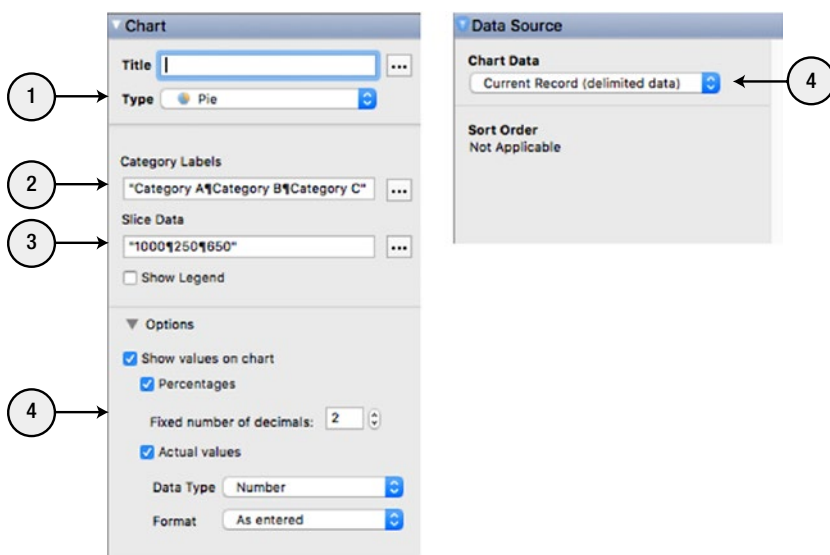


Figure 30-65. The configuration for a pie chart based on calculated data

The following settings are required for this example:

1. **Type** — Select **Pie** as the type. This will adjust the available settings to allow **Category Labels** and **Slice Data** values.
2. **Category Labels** — This area must result in a carriage return-delimited list of values that define the categories for the chart. Type directly into the text area or click the adjacent button to get a menu with two options:
 - **Specify Field Name** — opens a **Specify Field** dialog to allow a selection of the field containing the desired values.
 - **Specify Calculation** — opens a **Specify Calculation** dialog into which you can enter a formula that will return the desired values.

3. **Slice Data** — This area must result in a carriage return-delimited list of numeric values that define the slices of the pie chart that correspond to each of the **Category Labels** defined above. Type directly into the text area or click the adjacent button to get the same two options for **Category Labels**.
4. **Options** — These are optional settings to include labels on the chart showing the actual value and/or percentage of the whole.
5. **Chart Data** — From the **Data Source** section of the controls, select **Current Record (delimited data)** to instruct the chart engine to use data from the context of the current record only.

This example will generate a pie chart with three sections, as shown in Figure 30-66.

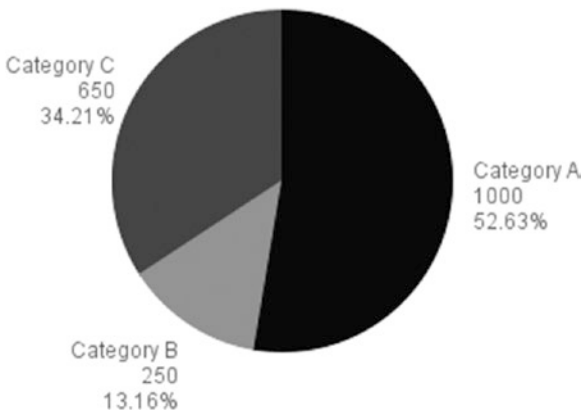


Figure 30-66. The chart generated by the example data

The **Category Labels** formula for this example is:

Category A¶Category B¶Category C

The **Slice Data** formula for this example is:

1000¶250¶650

Creating a Chart Using the Found Set

Charts can be created more dynamically using the records in the found set as the data source. Let's work through an example of creating a bar chart, displaying the number of **Contact** records by state.

To get started, insert a chart onto a layout, and then configure the settings as shown in Figure 30-67.

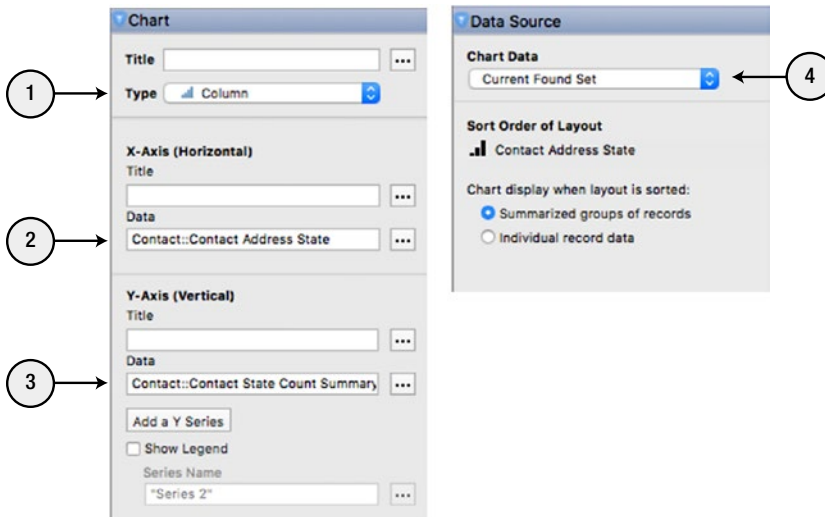


Figure 30-67. The configuration for a column chart summarized from the found set

■ **Tip** Creating charts from found sets can be a little confusing. It is like creating a report with a sub-summary part in that only records in the found set will be included and the sort order controls how a summary field counts records into subgroups.

The following settings are required for this example:

1. **Type** — Select **Column** as the type. This will adjust the available settings to allow X-Axis and Y-Axis values.
2. **X-Axis Data** — This area refers to a field containing text that will act as the labels for the bars. To avoid duplicate values, FileMaker will automatically summarize these values *if the found set is sorted by the field specified*. In our example, we point to the **Contact Address State** field, which is the field by which the found set is sorted.
3. **Y-Axis Data** — This area refers to a summary field containing the numeric values for the bar height. In our example, a new **Contact State Count Summary** field is used, which performs a **Count of the Record ID** field.
4. **Chart Data** — On the **Data Source** section of the controls, select **Current Found Set** to instruct the chart engine to summarize data from across all the records.

The resulting chart will appear as shown in Figure 30-68.

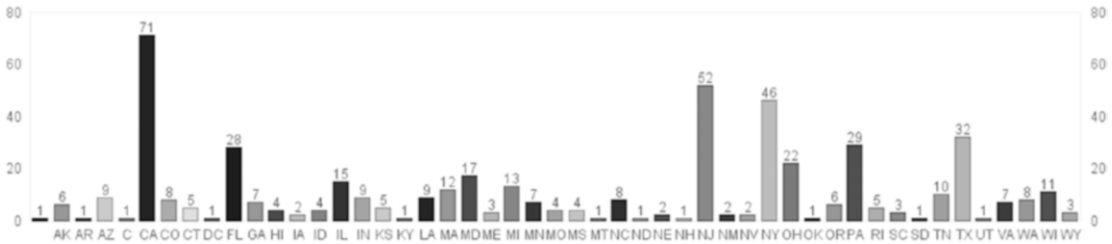


Figure 30-68. An example of a column chart displaying the number of contact records for each state

Remember, for this chart to summarize correctly, the records *must* be sorted by the x-axis (State) and the y-axis *must* be a summary field, counting the number of records by using a field such as the Record ID.

Creating a Quick Chart from a Table View

When viewing a found set of records in Table view, the field heading menu includes an option that allows any user to generate a quick chart based on the field whose heading menu was selected or by the first sort field. The Chart Setup dialog will be instantly configured with all the necessary default values and field references. The user can then modify settings and have a choice to Print Chart or, if their access settings allow, Save as Layout.

Summary

In this chapter, we discussed details about the objects that can be placed on layouts to allow a user to view and interact with the data structure and script functions of the database.

CHAPTER 31



Manipulating Objects

Once objects are placed on a layout, there are various ways to manipulate them. In this chapter, we will focus on manipulating objects, covering:

- Selecting
- Resizing
- Moving
- Arranging & aligning
- Hiding
- Conditional formatting
- Understanding tab order
- Naming objects

Selecting Objects

To *select* an object, click on it with the cursor or click on the layout near the object and drag the cursor toward it. The object will become highlighted with an outline that includes eight *sizing handles*, which are small dots located at the four corners and midway along each side, as shown in Figure 31-1.



Figure 31-1. An object at rest vs. a selected object which is highlighted and has eight sizing handles

To select more than one object at a time, either:

- Click on them one by one while holding down the Shift key
- Click on the background and drag the cursor so that the focal rectangle touches all the desired objects. Holding the Command (macOS) or Windows (Windows) key while dragging will require that an item be *completely* within the boundaries of the box before it will be selected.

When multiple objects are selected, they share one set of sizing handles among them, outlining the overall space of the group. See Figure 31-2.

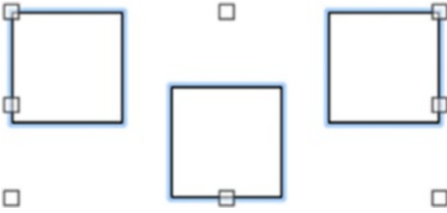


Figure 31-2. Objects selected together will have collective sizing handles

Objects can be de-selected by clicking on the background of the layout or by selecting another object. To de-select one of a group of selected objects, click on the object while holding the Shift key.

Resizing Objects

Objects can be resized using the following methods:

- Click on one of the eight sizing handles and drag the cursor away or toward the center of the object(s).
- Use the Position settings of the Position panel of the Inspector. Type a new measurement for the width or Height settings followed by the Tab or Enter key and the object will instantly adjust the object's size.
- Use the Resize To feature described later in this chapter.

Moving Objects

Once one or more objects are selected, they can be moved to a new location by clicking and dragging.

■ **Caution** Shape objects with no fill color are a little tricky to move by dragging. Clicking in the center of an object will click through the object onto objects behind it or the background of the layout. Instead, carefully click on the edge of the shape to grab and drag it.

In addition to dragging, selected objects can be moved using three other methods:

- Use the Up, Down, Left, or Right arrow keys to push the selection one pixel at a time in one direction.
- Use the Position settings of the Position panel of the Inspector (Chapter 29). Type a new measurement for the Left, Right, Top, or Bottom positions followed by the Tab or Enter key and the object will instantly shift to that new position.
- Use the Arrange and Align features described later in this chapter.

Layout Positioning Helpers

There are four features that can be helpful when positioning objects manually. These can be used individually or in unison to guide an object to a new location.

Ruler

The *ruler* is a horizontal and vertical strip running along the entire left and top of a window's content area that displays incremental markings based on a chosen unit of measurement: inch, centimeter, or point. The ruler, as shown in Figure 31-3, is only visible in Layout mode when the View > Rulers menu is active.

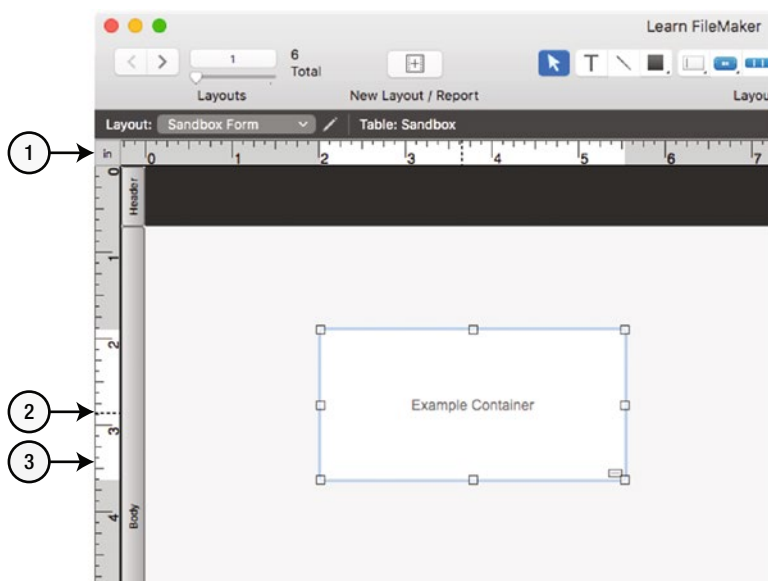


Figure 31-3. The Ruler helps you to drag an object into a precise position on the layout

The ruler has the following features:

1. At the corner where the horizontal and vertical rulers intersect is a button that cycles through the available units for the ruler.
2. The cursor's current position is marked in each ruler with a dotted line.
3. When dragging an object, the rulers both denote the boundaries of object with a white highlight.

■ **Tip** The unit of measurement can also be changed from the ruler's contextual menu.

When editing text directly on a layout, the top ruler transforms into a gray bar with a text ruler overlay that spans the width of text object, as shown in Figure 31-4. This ruler controls indentation, margins, and tabs within the text object in a similar manner to the text ruler accessible in Browse mode when editing a field.

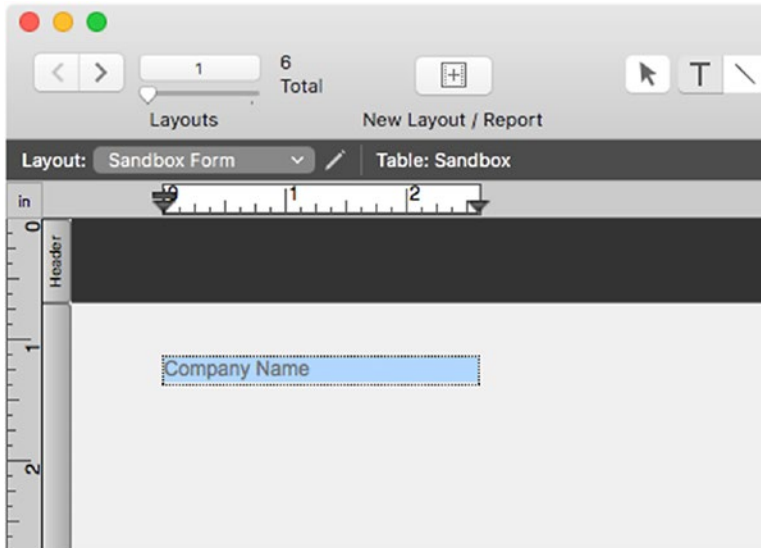


Figure 31-4. The top ruler transforms into a text ruler sized to the object being edited

Grid

The layout *grid* is a sequence of evenly spaced horizontal and vertical lines overlaid on a layout background that forms a regular grid of minor and major areas. These lines, resembling graph paper, are only visible in Layout mode when the View > Grid > Show Grid menu item is active or the corresponding check box in the Inspector is enabled (Chapter 29, “Grid”).

When visible, the lines can visually assist in uniform alignment of objects. Activating the View > Grid > Snap to Grid feature in the menu or the Inspector will cause an object being dragged to be subtly attached to the next grid line in the direction of travel.

Guides

A layout *guide* is a movable blue line used to align objects and define regions of a layout. Multiple guides can be placed horizontally or vertically on the layout, as shown in Figure 31-5. Guides don’t print and are only visible in Layout mode when the View > Guides > Show Guides menu is active. They are identical in form and function to the guides in desktop publishing and page layout applications.

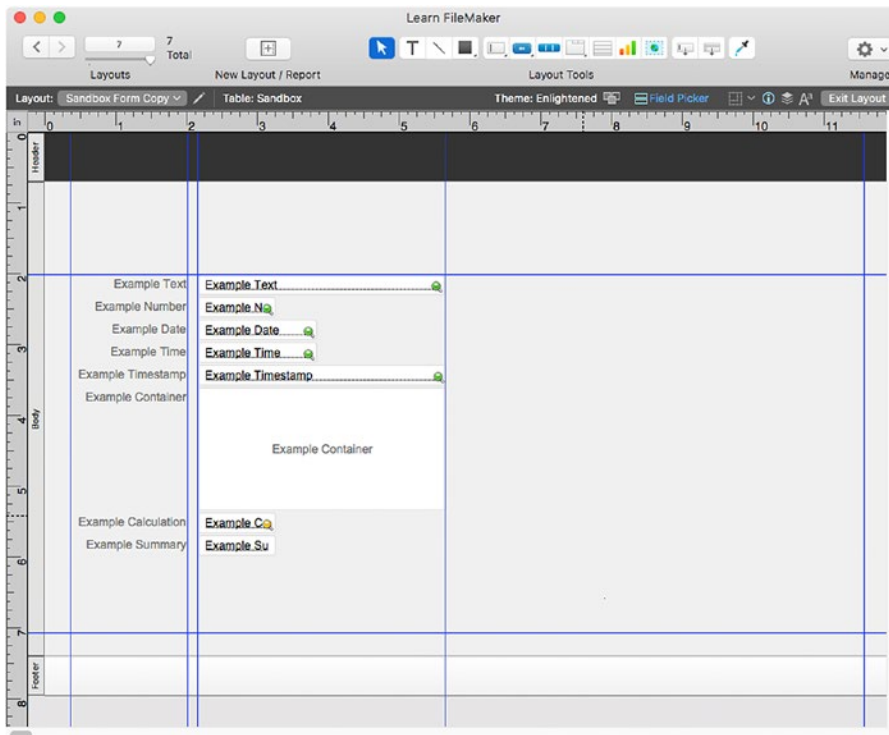


Figure 31-5. A layout with several guides outlining different regions

When visible, guides can visually assist in uniform alignment of objects. Activating the View > Guides > Snap to Guides feature in the menu will cause an object being dragged to be subtly attached to the next available guide line in the direction of travel.

■ **Note** Guides are layout specific so they must be configured for each layout.

With ruler must be visible, click on a ruler and then drag right or down respectively and release the new guide at the desired position on the content area. Guides can be repositioned by clicking, holding, and dragging to a new position or removed completely by dragging them back to the ruler.

Dynamic Guides

As an object is moved around a layout, blue lines called *dynamic guides* automatically appear and disappear on or around nearby objects showing alignment and spacing patterns that would result if the object were dropped in its current location. By showing multiple spatial relationships of groups of nearby objects relative to the position of the moving object, dynamic guides help to encourage consistency and order when manually moving an object. Like the grid and manually placed guide lines, objects will be subtly attracted to dynamic guides.

Activating the View > Dynamic Guides menu item will cause dynamic guides to appear on any layout when an object is being dragged with the cursor.

When moving an object, no guides will appear when the current position does not align to any other objects on the layout, as shown in Figure 31-6.

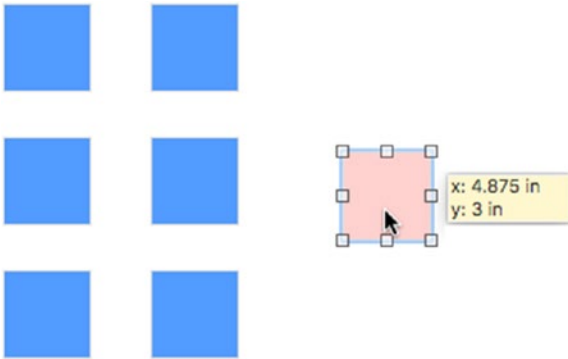


Figure 31-6. A dragged object that does not align with any other objects will show no dynamic guides

As the object is dragged further and begins to align with other objects, dynamic guides will appear, like the one shown in Figure 31-7. Alignments are shown for *left*, *right*, *horizontal center*, *top*, *bottom*, and *vertical center*, connecting to one or more other objects.

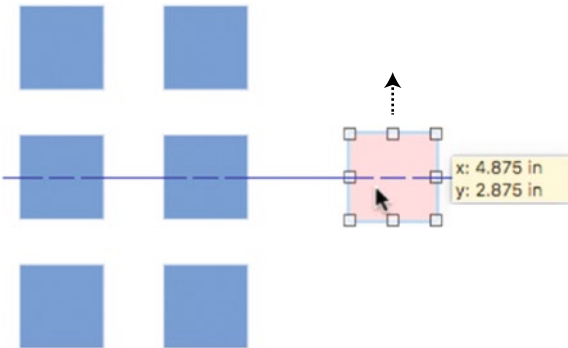


Figure 31-7. As the object moves into alignment with one or more objects, a guide will appear

When the object is dragged into a position that creates a consistent spacing pattern, those are highlighted as well, like that shown in Figure 31-8. These appear for horizontal and vertical arrangements and are displayed simultaneously with alignment guides.

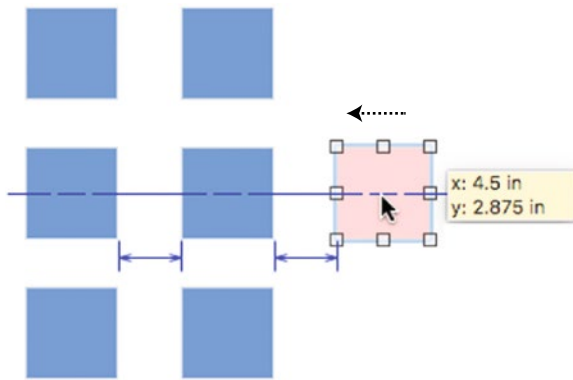


Figure 31-8. Multiple guides can appear showing both alignment and uniform spacing between objects

■ **Note** Many examples in this chapter are shown with square shapes for illustrative purposes only. Dynamic guides and other features described will work the same with *any* combination of different layout object types, non-uniform arrangements, and varying sizes.

Arranging & Aligning Objects

When working on complex layouts with dozens or hundreds of items, moving, sizing, and spacing groups of objects can be tediously repetitive to perform manually. Also, it is possible to rotate an object on a layout and to lock objects to avoid accidentally disrupting a meticulously designed layout.

FileMaker has functions to assist in the task of aligning, distributing, resizing, rotating, grouping, and locking objects, accessible from three places when in Layout mode:

- The Arrange menu in the menu bar.
- The Arrange sub-menu in the contextual menu.
- The arrange & align group of tools on the Position panel of the Inspector.

With the exception of Rotate, these functions all work with any combination of object type.

■ **Note** These functions work in FileMaker like those found in countless other page layout and desktop publishing applications.

Align

The **Align** group of functions include commands to automatically align groups of selected objects horizontally (**Left Edges**, **Center**, **Right Edges**) or vertically (**Top Edges**, **Middle**, **Bottom Edges**).

For example, with several misaligned objects selected, choose the **Arrange > Align > Left** menu item to instantly align them along their left border, as shown in Figure 31-9.

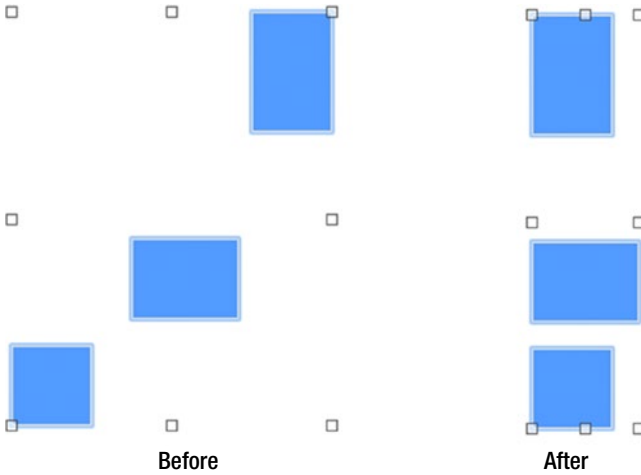


Figure 31-9. The transition from misalignment to **Left alignment** is instantaneous using the **Align > Left** function

Resize To

The **Resize To** group of functions include commands to automatically resize groups of selected objects to **Smallest Width**, **Smallest Height**, **Smallest Width & Height**, **Largest Width**, **Largest Height** or **Largest Width & Height**.

For example, with several non-uniformly sized objects selected, choose the **Arrange > Resize To > Smallest Width and Height** menu item to instantly resize every object to a uniform size and height based on the object with the smallest of each, as shown in Figure 31-10.

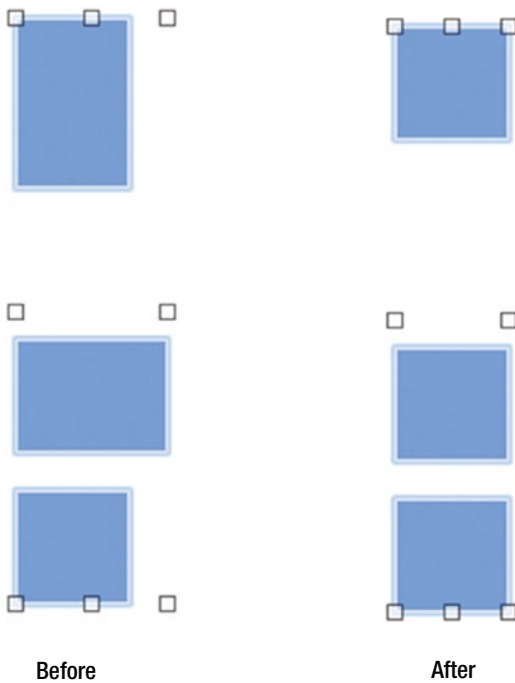


Figure 31-10. The transition from non-uniform size (Left) to uniform size (Right) is instantaneous using the *Resize To* functions

Distribute or Space

Objects can be distributed or uniformly spaced using the *Distribute* submenu in the *Arrange* menu or contextual menu as well as the *Space* buttons on the *Arrange & Align* section of the *Position* panel of the *Inspector*. The functions will reposition the objects so that they are uniformly spaced horizontally or vertically using the outside measurements of the group along the respective axis.

For example, with several non-uniformly spaced objects selected, choose the *Arrange > Distribute > Vertical* menu to instantly reposition every object with uniform space between them, as shown in [Figure 31-11](#).

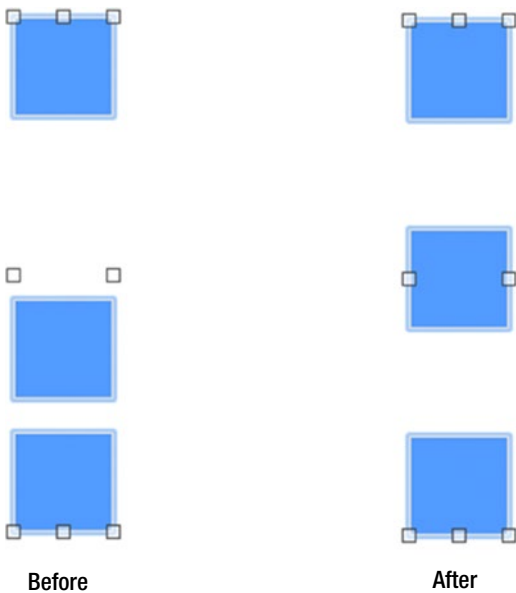


Figure 31-11. The transition from non-uniform spacing (Left) to evenly distributed vertical spacing (Right) using the *Distribute* function

Rotate

Many objects can be rotated in a clockwise direction 90 degrees at a time by selecting the **Arrange > Rotate** menu item, as shown in Figure 31-12.



Figure 31-12. A text object before and after two rotations

■ **Note** The following object can't be rotated: button bars, charts, popovers, portals, slide controls, tab controls, and web viewers.

Group

Grouping objects makes it possible to select and manipulate multiple objects as if they were a single object. Once grouped, they can be resized, moved, formatted as a button, and more.

The objects within the group are “frozen” in place in size and in position relative to one another and can only be edited as a group. However, most objects such as buttons, button bars, charts, fields, web viewers, and more can be edited *functionally* while included in a group by double-clicking directly on the object. In some cases, an object will need to be ungrouped for editing.

A set of selected objects can be grouped by selecting the **Arrange > Group** menu item. It is even possible to group groups into a grouping hierarchy. To ungroup formerly grouped objects, select the **Arrange > Ungroup** menu item.

Lock

Locking an object causes it to be uneditable and is a great way to avoid accidental modifications or deletions of carefully arranged layout items. A locked object can't be moved, resized, reassigned, reconfigured, or changed in any way unless and until it is unlocked. When selected, locked objects displays an "x" icon in place of each sizing handle, as shown in Figure 31-13.



Figure 31-13. A locked object is indicated by a restriction icon in place of sizing handles

An object can be locked by selecting the **Arrange > Lock** menu item. Once locked, the item will not respond to any edit commands. Occasionally, when attempting to change a setting in the Inspector, a dialog will appear stating that the change couldn't be made because of the object's locked status. However, in many cases, the command will just be ignored. For example, selecting a locked object and typing **Delete** will result in no action and no error dialog. If the locked object is selected with other unlocked objects, deleting will successfully delete everything except for the locked object.

An object can be unlocked by selecting the **Arrange > UnLock** menu item.

Object Stacking

Objects on a layout are stacked in an order relative to other objects based on when they were created. When two objects are moved together, the one created after the other will appear on top of the first, obscuring it from view. This is called the *stacking order*.

There are four commands that affect an object's position within the stack, as illustrated in Figure 31-14.

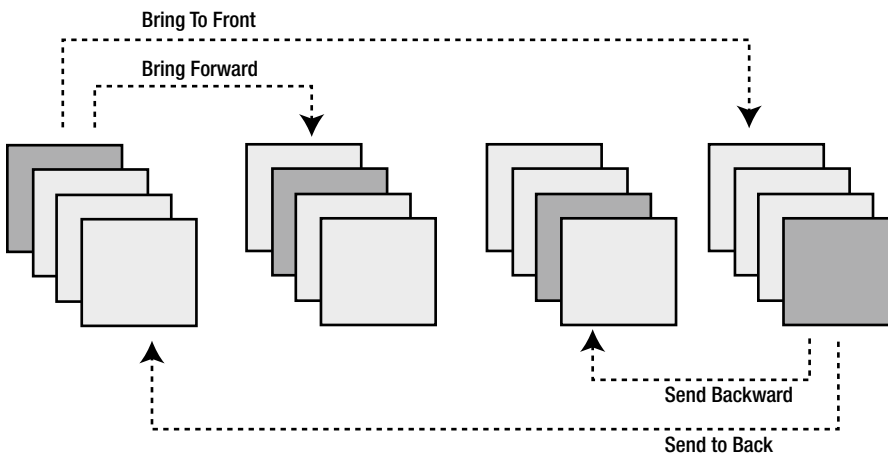


Figure 31-14. An illustration of all four commands that affect an object's stack position

The four stacking commands are:

- **Bring to Front** — Brings the selected object(s) *to the front* of the stack.
- **Bring Forward** — Brings the selected object(s) *one step forward* in the stack.
- **Send Backward** — Sends the selected object(s) *one step backward* in the stack.
- **Send to Back** — Sends the selected object(s) *to the back* of the stack.

Hiding Objects

The `Hide objects` when setting is a formula that controls whether an object will be visible on the layout. The feature makes possible conditional interface elements that appear or disappear based on the values in fields, conditions of the window context, or details about the user's environment, such as:

- Hide buttons until the appropriate value(s) are entered to make their function applicable or vary buttons depending on conditions, such as the open or committed status of a record.
- Hide fields based on the status or type of a record.
- Hide objects based on the window's mode.
- Hide a chart until enough information has been entered to make it useful.
- Have two overlapping portals or tab controls toggle depending on the type of record.
- Make layout objects only accessible for certain access privilege sets. This can be used to hide restricted fields and avoid users seeing fields stating "no access."

The hide formula is entered in the Behavior section of the Data panel of the Inspector. When it evaluates to a true result, the object will be hidden. When it returns a false value, the object will be visible to the user. Objects always remain visible in Layout mode.

■ **Tip** Using the `Hide objects` when feature with a formula that always returns a true result, short developer notes can be embedded directly within the content area, next to interface elements to which they refer. For example, when objects are layered on top of one another, a short piece of red text near them stating "Layered objects" is a helpful reminder when in Layout mode but could be hidden in Browse mode.

Conditional Formatting

Conditional formatting is a layout feature that defines format combinations that will be applied to an object when certain conditions are met. Each condition is defined as a custom formula or by selecting one of twenty predetermined content value conditions.

■ **Caution** Don't confuse conditional formatting with text formatting functions. The former affect the *appearance* of an object on a layout while the latter affects the actual text *content* of a field or formula-driven object text.

The rules for conditionally formatting an object are set in the Conditional Formatting dialog, shown in Figure 31-15. The dialog can be open when one or more objects is selected and then either:

- Choosing the Format > Conditional menu item.
- Open the object's contextual menu and select the Conditional Formatting option.

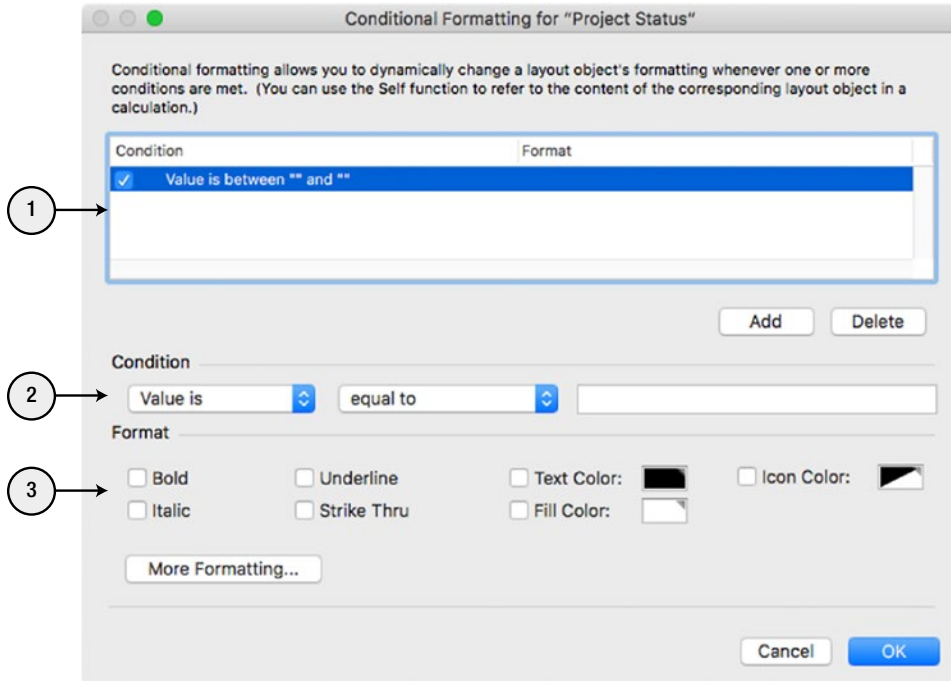


Figure 31-15. The Conditional Formatting dialog

The Conditional Formatting dialog options include:

- **Condition List** — a list of every condition defined for the selection object(s). An enabled check box indicates an active condition. Use the Add and Delete buttons to create or remove conditions from the list. Conditions are evaluated from top to bottom. If more than one evaluates as true and creates formatting conflicts, lower conditions in the list will override previous ones above. To ensure that conditions are applied as expected, you may need to drag them up or down in the list accordingly.
- **Condition** — defines the formula for the selected condition. When the result of the condition is true, the formatting will be applied to the object.
- **Format** — the formatting options that will be applied to the object(s) when the condition is met.

Condition Formula Options

A *condition formula* defines the contextual circumstances under which the format settings will be applied to the object(s). This can be one of two types: *Value is* and *Formula is*.

Value-Driven Conditions

The *Value is* condition compares the text content of the object to one or more static values depending on the choice of twenty different operators, as shown in Figure 31-16. Once an operator is selected, one or more fields appear into which static values can be inserted to complete the implied formula.

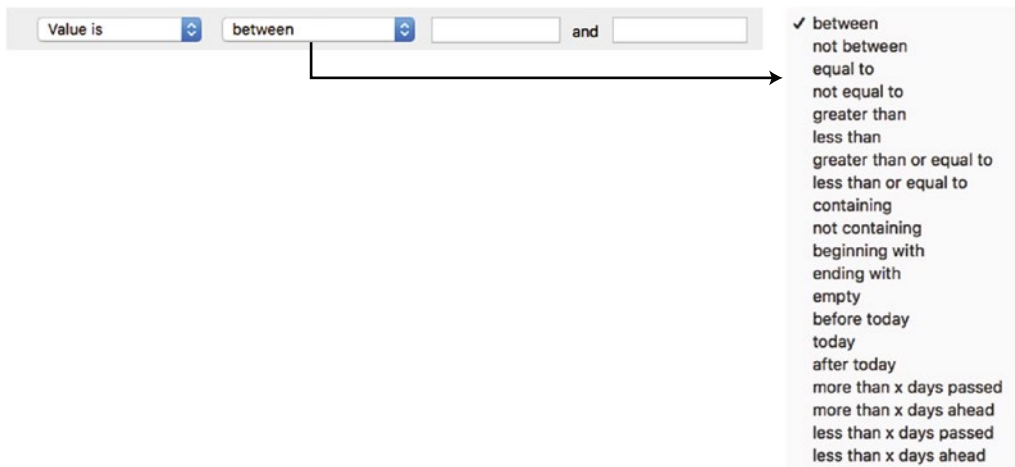


Figure 31-16. The options for a value-driven condition

This provides an ease-of-use option for simple conditions, comparing the contents of the object to the static values based on the selected operator. For example, if the value of a *Status* field is equal to “Urgent” or if a *Total Price* field is less than zero, the text can be colored red and made bold.

Formula-Driven Conditions

The *Formula is* condition accepts a totally custom formula that is used to evaluate the condition as Boolean and then apply the formatting or not.

Once selected, a text area and *Specify* button appears, as shown in Figure 31-17.



Figure 31-17. The options for a formula-driven condition

Being completely custom, the formula entered can use any contextual information in its evaluation. This includes the value of any accessible fields (local or related), global variables, built-in functions, custom functions, etc.

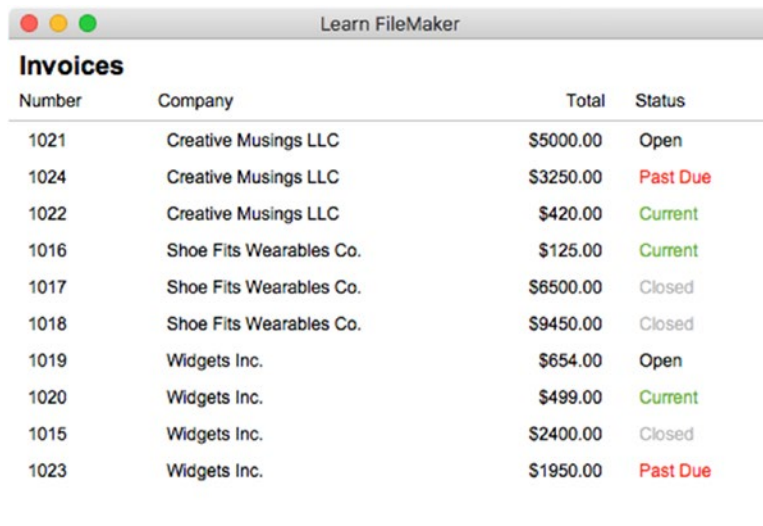
Conditional Format Settings

The format settings defined will be applied to the object when the formula evaluates to a true result. These settings can include one or more of the following:

- Text font, size, color, and style
- Fill color
- Icon color

Conditionally Formatting an Invoice Status Field

This example conditional formatting uses a Status field in an Invoice table, like the one shown in Figure 31-18.



Number	Company	Total	Status
1021	Creative Musings LLC	\$5000.00	Open
1024	Creative Musings LLC	\$3250.00	Past Due
1022	Creative Musings LLC	\$420.00	Current
1016	Shoe Fits Wearables Co.	\$125.00	Current
1017	Shoe Fits Wearables Co.	\$6500.00	Closed
1018	Shoe Fits Wearables Co.	\$9450.00	Closed
1019	Widgets Inc.	\$654.00	Open
1020	Widgets Inc.	\$499.00	Current
1015	Widgets Inc.	\$2400.00	Closed
1023	Widgets Inc.	\$1950.00	Past Due

Figure 31-18. An Invoice list view with conditionally formatted Status field

In this example, the field allows four values; Open, Current, Past Due, and Closed and has the following conditions:

- An Open invoice will display the status without any additional formatting applied. This option will not require a condition since the default field format will be used.
- A Current invoice will apply a fill color of green to the status, highlighting the fact that the money is due but not yet overdue.
- A Past Due invoice will apply a fill color of red to the status, highlighting the fact that the money is overdue.
- A Closed invoice will color the status text a slightly muted gray, making it less prominent and emphasizing the fact that the invoice has been paid and is now inactive.

To configure the last three conditions, select the field on a layout, open the Conditional Formatting dialog, and configure it as shown in Figure 31-19.

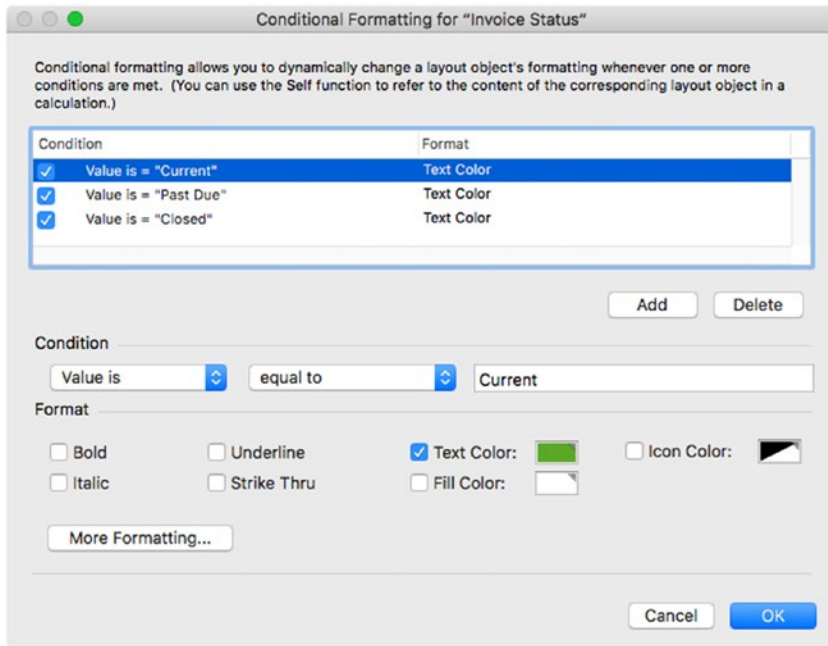


Figure 31-19. The Conditional Formatting dialog configured to apply three different text colors to the Invoice Status field

To set up the condition for a Current status:

1. Click the Add button to create a new condition.
2. Leave the Condition type to Value is and choose the equal to operator, then enter "Current" into the adjacent field.
3. Check the Fill Color box and choose a light green color.

To set up the condition for a Past Due status:

1. Click the Add button to create a new condition.
2. Leave the Condition type to Value is and choose the equal to operator, then enter "Past Due" into the adjacent field.
3. Check the Fill Color box and choose a light red color.

To set up the condition for a Closed status:

4. Click the Add button to create a new condition.
5. Leave the Condition type to Value is and choose the equal to operator, then enter "Closed" into the adjacent field.
6. Check the Text Color box and choose a medium gray color.

Understanding Tab Order

A layout’s *tab order* is the sequence in which objects will receive focus, as the user steps through them by typing the Tab key. The default tab order is from left to right and top to bottom through every object that can accept focus. However, if objects are moved around on a layout, the tab order may need to be reset. It can also be fully customized so that objects are excluded or flow in a custom order.

■ **Note** Depending on a field’s Go to next object using settings, typing Return or Enter may perform the same object stepping action as the Tab key (Chapter 29, “Behavior”).

Changing the Tab Order

To edit the tab order, select the Layout > Set Tab Order menu item while in Layout mode. This will place the layout of the current window into a tab-editing mode, shown in Figure 31-20.

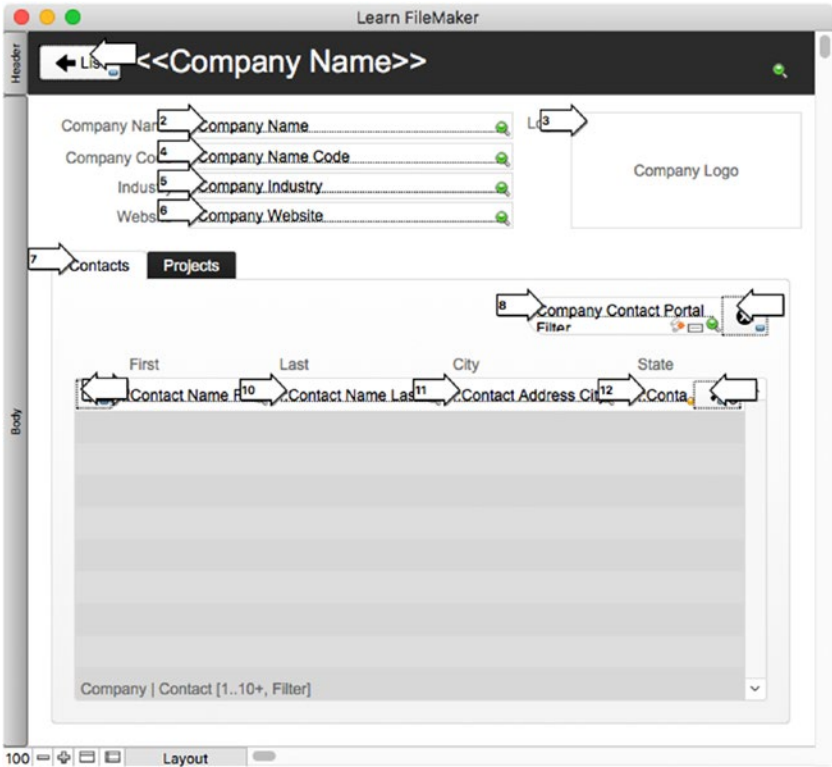


Figure 31-20. A window in Set Tab Order mode

While in tab-editing mode, each object capable of receiving focus will have a small arrow icon superimposed attached to one side. These contain the object's current number in the tab order and are editable fields. The following actions can be performed in these arrow fields:

- Delete the number to remove the object completely from the tab order. That object will be bypassed when a user is tabbing in Browse mode.
- Click into a blank arrow field to automatically assign the next available number in the tab order.
- Edit the number by typing over it. FileMaker will enforce a unique number sequence by automatically changing the number existing in one arrow if that same number is typed into another arrow.

Using Functions of the Set Tab Order Dialog

Tab order is modified using the Set Tab Order dialog, shown in Figure 31-21, which opens automatically when a layout is placed into tab-editing mode.

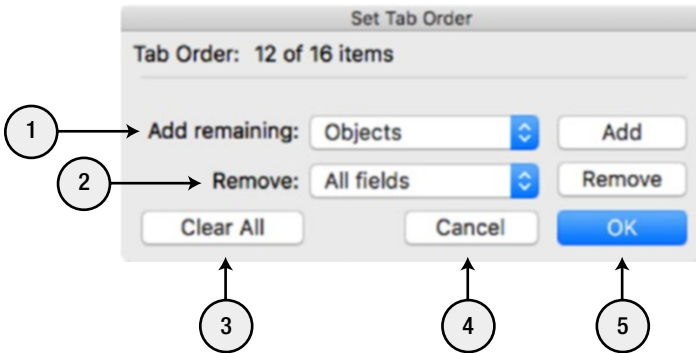


Figure 31-21. The Set Tab Order dialog from which tab order can be modified in several different ways for the entire layout

The following batch functions are available on the Set Tab Order dialog and can make specific changes across groups of objects on the current layout:

1. **Add Remaining** — This combination pop-up menu and button makes available several functions to add excluded objects into the tab order. Select from the following options in the menu and click the Add button:
 - **Objects** — This option will add to the tab order all qualifying objects not yet assigned.
 - **Fields only** — This option will add to the tab order any *fields* not yet assigned.
 - **Buttons only** — This option will add to the tab order any *buttons* not yet assigned.
2. **Remove** — This combination pop-up menu and button makes available two functions to remove objects from the tab order. Select from the following options in the menu and click the Remove button:

- All fields — This option will remove all *fields* from the tab order.
 - All buttons — This option will remove all *buttons* from the tab order.
3. Clear All — This button will automatically delete the tab order from every object on the layout, providing a clean slate from which you can begin adding items manually or with the Add Remaining function.
 4. Cancel — This button will close the Set Tab Order dialog and return the window to standard Layout mode without saving any unsaved changes made to the tab order.
 5. OK — This button will save changes to the tab order, close the Set Tab Order dialog, and return the window to standard Layout mode.

Naming Objects

The *name* property is a unique static text identifier manually assigned to a layout object or group of objects. Names are optional but become necessary to allow certain functions and script steps to refer to non-field objects such as portals, tab control, slide control, and web viewers. For example, the `GetLayoutObjectAttribute` and `Get (ActiveLayoutObjectName)` functions can *only* refer to an object by name. Similarly, the `Go To Object`, `Refresh Portal`, and `Set Web Viewer` script steps require an object name.

An *object name* should not be confused with a *field name* as they are two separate things. An object name is a property unique to a single object instance on one layout for any type of object. A field name is a universal definition of a specific data container common to the field on any layout it is placed. An instance of a field can be assigned an object name different from its defined name, and each instance of the same field on any layout can have a different object name.

To assign an object name, follow these steps:

- In Layout mode, select an object or a set of grouped objects. Multiple objects not grouped can't be assigned a name.
- In the Position group of the Position tab of the Layout Inspector, click into the Name text area.
- Type a name for the object.

■ **Note** Object names are not case sensitive and must be unique within the context of a layout. FileMaker displays a warning dialog if you attempt to enter a name already used by another object on the same layout.

Summary

In this chapter, we discussed various options for manipulating objects on a layout.

CHAPTER 32



Using Themes and Styles

Text formatting and paragraph settings can be applied to any object directly by selecting the object and then using options in the Format menu or the Appearance panel of the Inspector. This approach is fine, especially for beginners working on small database projects with only a handful of layouts. However, as a project grows, you will want to look for methods to streamline future design changes and improve interface performance. This can be accomplished by using themes and styles.

In this chapter, we will introduce and define theme and style basics, including the following:

- Introducing themes and styles
- Using themes
- Using styles

Introducing Themes and Styles

FileMaker automatically assigns a theme to every new layout and a Default style to every object added to a layout. As a developer, you can select from the following choices:

- Use the default theme or choose from available built-in themes, using the styles it contains *without modification*.
- Use the default theme or choose from available built-in themes, customizing the styles it contains.
- Create your own theme and styles, either completely custom or based on a default theme as a starting point.
- Ignore themes and styles and apply formatting changes as needed directly to objects without updating the theme.

Defining Styles

A *style* is a named collection of various appearance settings for one layout object type.

The functionality of styles can be compared to styles found in word processing, desktop publishing applications, and cascading style sheets (CSS) with some differences in their accessibility. Styles are *similar* in that they can quickly apply consistent formatting to new objects and update the formatting of existing objects by modifying the settings of a style they share.

However, styles in FileMaker are *different* in a couple of ways. Since they are a layout design mechanism used by *developers* to facilitate efficient layout design, they are *not available* for user activity when editing the text contents of a field in Browse mode like they are in word processing and desktop publishing. Also, unlike CSS and desktop publishing, styles in FileMaker are discrete and cannot inherit attributes from other styles.

Styles can be saved for the following object types:

- Layout background
- Layout part
- Shape
- Edit box
- Drop-down list
- Pop-up menu
- Checkbox set
- Radio button set
- Drop-down calendar
- Container
- Text
- Button bar
- Button
- Popover
- Slide control

Each of these objects can be assigned any one style defined in the layout's theme for that object type. For example, depending on the complexity of the layout and a designer's preferences, there may be a single style for *all* field labels or one or more separate styles for labels on a form view, list view, and in portals. When the formatting requirements are different, a new style should be created.

Styles contain format settings based on the type of object and the applicable properties, such as:

- Fill and border settings
- Corner radius
- Text font, size, and style
- Text alignment and paragraph borders
- Indents and line height
- Tab settings

Exploring the Hierarchy of a Style

It is important to understand that a style does *not* contain a single group of format settings. Instead, it comprises multiple format settings grouped into a rather complex hierarchy, as illustrated in Figure 32-1.

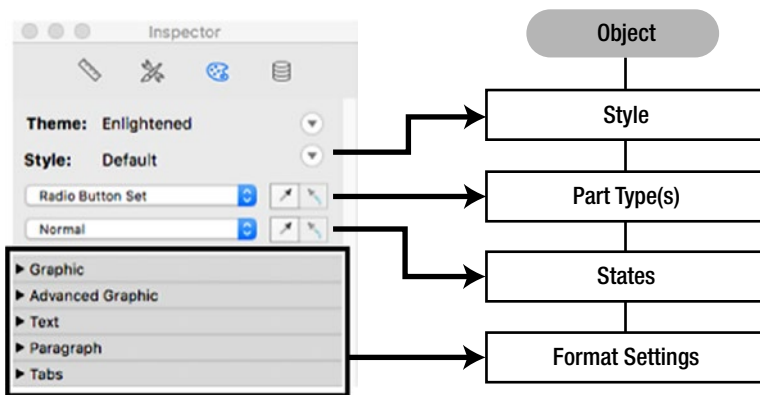


Figure 32-1. An illustration of the hierarchy defining the groups of multiple format settings that make up a style

Once an object is assigned a style, the interface options reflect choices for two levels of hierarchy above the format settings: *object states* and *type groups*.

Defining Object States

An *object state* is the status of an object based on the current type of interaction by a user. Each state stores one entire set of format settings allowing the appearance of the object to change based on the user's action.

For example, most objects have the following appearance states that are all stored under a single style:

- Normal — The style settings when the object is visible on a layout but not actively engaged.
- Hover — The style settings when the user moves the cursor over the top of it.
- Pressed — The style settings when the user clicks the cursor and holds it down.
- Active — The style settings when the object has active focus.

In addition to these, some objects have additional states available based on their properties. For example, *fields* have a Placeholder Text state and a *button bar's segment* has an Inactive state.

Defining Object Part Types

An *object part type* is one aspect of a complex layout object that is technically made up of multiple elements. The object itself is always represented as the primary type with additional types specified where applicable. Some examples include:

- A *button* is made up of two part types:
 - Button
 - Button: Icon

- A *button bar* is made up of four part types
 - Button Bar
 - Button Bar: Divider
 - Button Bar: Segment
 - Button Bar: Icon
- A *portal* is made up of two part types:
 - Portal
 - Portal: Row

The reason for these secondary elements is to allow any format setting applied to one part of an object to be configured differently to another part of an object. For example, a button bar might have one size, style, and color of border while the divider might have different border settings, both for the same object state. By subdividing the object into more granular components, a single state of a single style can contain two or more different details for the same format setting.

Each part type for an object contains its own complete set of object states. So, an entire button bar can change border color when the cursor hovers over it and the individual segment with focus can change to a different border color at the same time.

Illustrating the Exponential Nature of Format Settings for Complex Objects

With an understanding of the style hierarchy, we see that an object can have one or more part types, each comprised of multiple object states, each containing a full set of format settings, as illustrated in Figure 32-2 for a button.

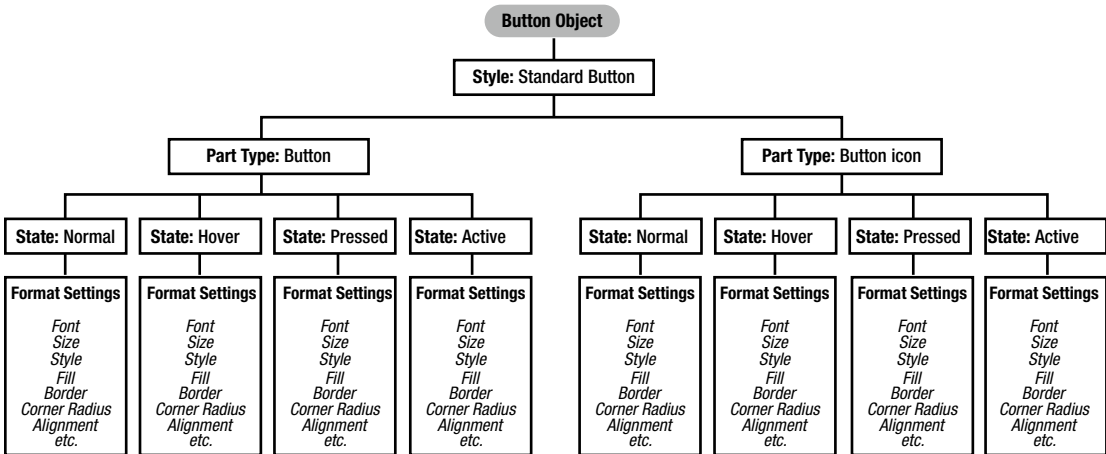


Figure 32-2. An example of the number of groups of format settings mandated for a button object

This illustration helps visualize the exponential number of format settings that can be defined for an object. A button will have eight different groups of format settings since it has two part types, each with four states.

Updating Styles

When an object's format settings are modified, they are *not* automatically saved back to the assigned style but are held as *unsaved object changes*. Any change(s) made to an object must be explicitly saved back to the style where they will be automatically applied to other objects on the same layout that are assigned the same style, as shown in Figure 32-3.

This update step only affects object styles on the *same layout*. To make changes globally, the style must be saved back to the theme.

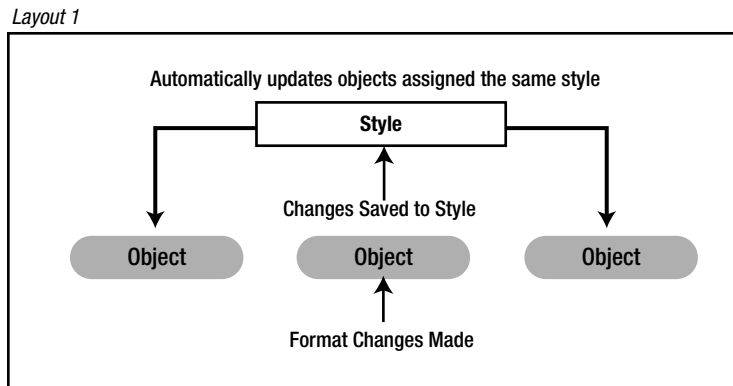


Figure 32-3. Format changes only apply to the object affected until those changes are saved to the style

Defining Themes

A *layout theme* is a single, named collection of styles for various object types used to coordinate styles uniformly within a layout and across multiple layouts in a single database file. Databases are created with a group of built-in themes that can be used without modification, with customized, or not at all by using a custom theme. Professional developers may want to create a robust custom theme with dozens of different styles for every object type to help them build visually pleasing and consistently formatted interfaces.

Every layout in a database *must* have a theme assigned. While they *can* vary from one layout to the next, to achieve the maximum benefit, a single theme shared by all layouts is recommended.

FileMaker ships with fifty-two built-in themes. Some are rather sophisticated and can provide practical examples of advanced layout features like the background image slice feature.

Updating Themes

When formatting changes are saved back to a style, they are *not* automatically saved back to the assigned theme but are held as *unsaved style changes*. Any change(s) made to a theme's styles must be explicitly saved back to the theme where they will be automatically applied to other objects on any other layout that is assigned the same theme, as shown in Figure 32-4.

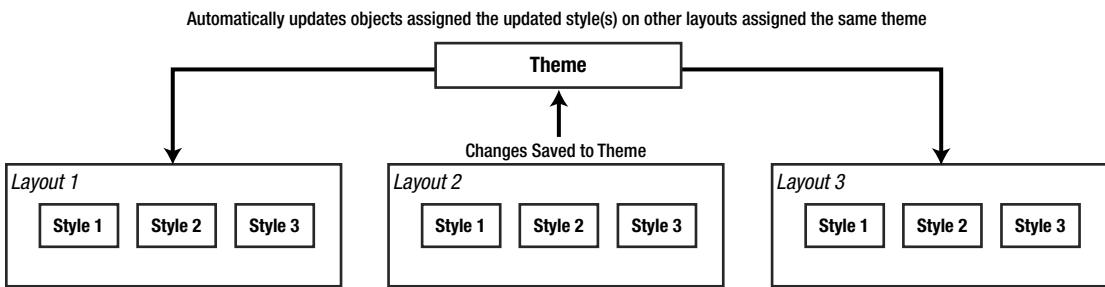


Figure 32-4. Style changes are only saved on the layout upon which they take place until they are saved to the theme

Remember, this step only updates layouts that are assigned the same theme. So, if every layout in a database is assigned a single theme, a change to one object type on one layout can be saved back to the style and then back to the theme to update every single similar object type assigned that style in the entire database!

■ **Caution** Since style and theme updates must be explicitly saved, get into the habit of doing so immediately after any change to an object to avoid a situation where many different unsaved changes are pending for multiple objects sharing a style.

Using Themes

As new layouts are created, FileMaker will automatically assign the theme it determines best suited for the new layout type. The somewhat awkward result roughly follow these rules:

- The default layout of a new database will always be assigned the Enlightened theme.
- New Computer layouts will always be assigned the same theme as the layout one is viewing when beginning the New Layout process.
- New Touch Device layouts will be assigned the Enlightened Touch theme unless the built-in theme of the layout one is viewing when beginning the New Layout process has a built-in touch variant. For example, if viewing a layout assigned the Luminous theme, the new touch layout will be assigned the Luminous Touch theme. However, if viewing a layout assigned the Cosmos theme, the new touch layout will be assigned the Enlightened Touch theme.
- New Printer layouts will be assigned the Enlightened Print theme.

Once the new layout is created, you can choose another theme for the layout.

■ **Caution** You can create custom themes and name them the same as a built-in or custom themes with a suffix of Touch and Print. However, these will not change the logic of automatic theme assignments listed above and must be manually assigned to new layouts. FileMaker does not look at custom theme names when determining the best default assignment for a new layout. All custom themes must be manually assigned to a layout.

Choosing a Layout's Theme

At any time after a layout is created, the theme assignment can be changed using the Change Theme dialog, shown in Figure 32-5. This dialog can be opened while in Layout mode using one of two methods:

- Click the icon next to the Theme name listed in the second row of the toolbar.
- Open the contextual menu on the layout's background and select Change Theme.
- Choose the Layout > Change Theme menu item.

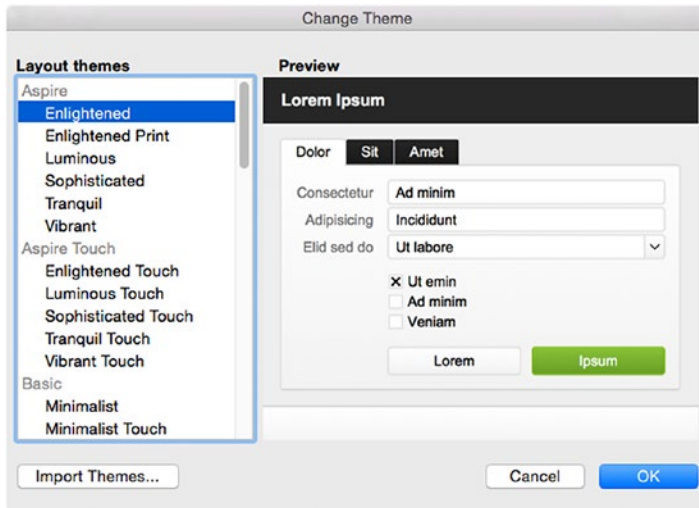


Figure 32-5. The Change Theme dialog window in which a theme can be selected and assigned to the current layout

The list of available themes is organized in a categorized list with the theme assigned to the current layout selected. If the database has custom themes defined, they will be listed in a Custom Themes category at the top of the list.

To change the layout's theme, select any theme in the list and click OK. The selected theme will appear in the second row of the toolbar and at the top of the Appearance tab of the Inspector.

When the theme assignment has changed, every object on the layout will be affected in some way depending on the styles available in new theme compared to those in the old. Some things to consider:

- Objects will retain their assigned style if the new theme has a style with the same name, changing the format settings as needed. Style names are *not* case sensitive.
- Any object assigned a style name that is not found in the new theme will be assigned the Default style for the object's type from the new theme.
- Most unsaved formatting changes, those applied directly to objects and not saved back to the style and theme, will be lost during the theme transition.
- However, some unsaved text formatting changes, such as text size, will be retained when the theme is modified. Other settings like font color and font family will always change to reflect the corresponding style in the new theme, regardless of their saved status on the old theme.

To assist in retaining and reapplying unsaved formatting changes, FileMaker has a rather clever two-stage undo process for theme changes. Immediately after making a theme change, using the Undo command the *first time* will retain the new theme assignment but restore any object attributes that were lost due to being unsaved in the old theme. Using the Undo command a *second time* will fully restore the layout to the previous theme with the previously unsaved modifications still in place. This allows you to assign a new theme on a trial basis, go back a half step, and then save the previously unsaved changes to the new theme if needed.

■ **Tip** For the best experience, don't change themes unless necessary. Choose a theme, duplicate it with a new name, and use it consistently throughout your database, constantly updating any changes made to an object. This will help you avoid manually cleaning up a formatting mess after a theme change.

The list of themes in the Choose Theme dialog is not editable and built-in themes can't be deleted from FileMaker. However, you *can* duplicate built-in themes and create fully custom themes from the Manage Themes dialog.

Managing Themes

Themes are managed from the Manage Themes dialog, shown in Figure 32-6, which can be opened by selecting the File > Manage > Themes menu item. As you assign themes to layouts, they are added to this dialog.

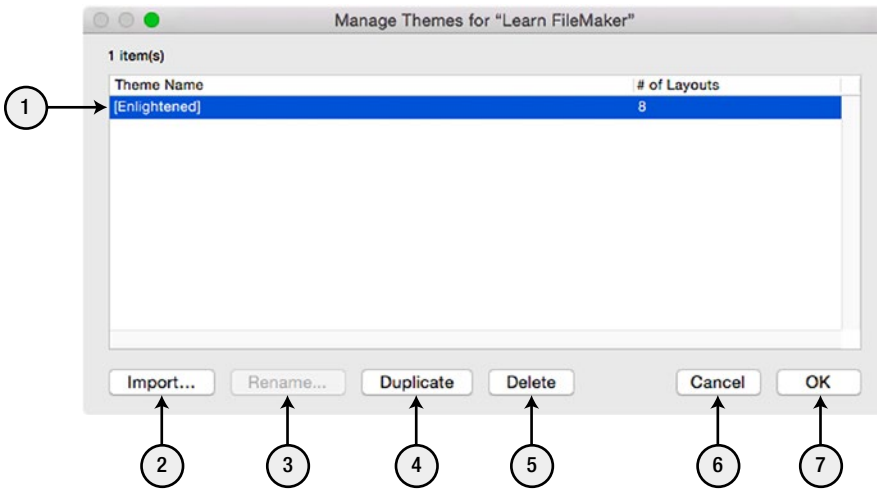


Figure 32-6. The Manage Themes dialog

The following controls are available on the Manage Themes dialog:

1. Themes — This area contains an alphabetical list of every theme that has been used on a layout within the database file, even if it is no longer in use. They remain here until you explicitly delete them. The first column contains the name of the theme, and the second column displays the number of layouts to which the theme is currently assigned. Built-in themes will always be contained within square brackets.

2. **Import** — This button begins the process of importing custom themes from another database file.
3. **Rename** — This button opens a dialog allowing you to rename a custom theme. Built-in themes in the list can't be renamed; however, a built-in theme can be duplicated and renamed to create a custom theme.
4. **Duplicate** — This button allows you to duplicate the selected theme. The new theme will have the same name as the selection with the word “Copy” added as a suffix. It can then be renamed if desired.
5. **Delete** — This button allows you to delete themes listed above that are no longer assigned to any layouts. If you attempt to delete a theme that is currently in use, an error dialog will be presented informing you of this fact.
6. **Cancel** — Click this button to close the dialog without saving any changes to the theme list.
7. **OK** — Click this button to save changes and close the dialog.

Using Styles

As an object is added to a layout, Filemaker automatically assigns it the Default style based on the type of object from the layout's theme. This style assignment can be changed to a different existing style or saved as a new style. As styles are created, deleted, and modified, these changes must be saved back to the Theme to update other layouts using the same theme.

Changing an Object's Style Assignment

To change the style assigned to an object, perform one of these actions, shown in Figure 32-7:

- Select the object and then choose a style from the list on the Styles panel of the Inspector.
- Open the object's contextual menu and select a style from the Object Style submenu.

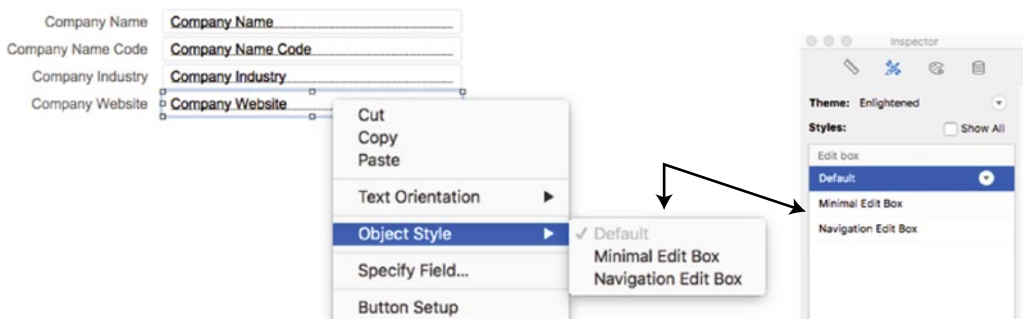


Figure 32-7. The two methods by which to assign a new style to a selected object

Once the change is made, all the appearance and format settings for every state of every part of the object will be updated to reflect the settings defined by the style.

Editing a Style's Settings

To modify an object's style, select the object and then open the Appearance panel of the Inspector and follow the actions, as shown in Figure 32-8.

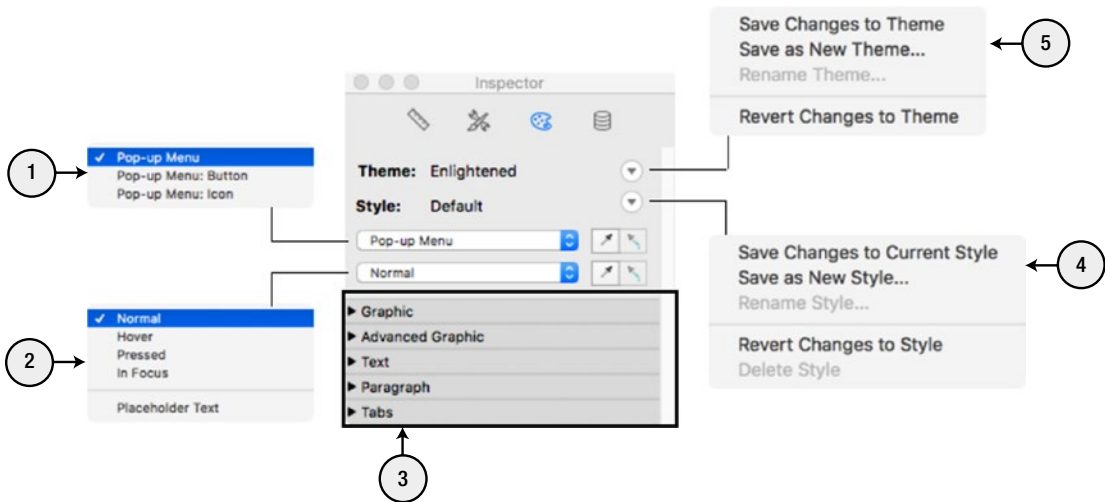


Figure 32-8. The process for modifying an object's style

To change the style of the selected object, follow these steps:

1. Choose the *object part type* to indicate if you are editing settings for the object itself or a sub-part. The default selection will always be the object itself.
2. Choose the *object state* to indicate for which state you are editing settings. The default selection will always be Normal.
3. Edit settings for the object. After any changes are made, the menu icon adjacent to the Style in the Inspector will turn red indicating the presence of unsaved changes.
4. Choose one of the following *style actions* from the menu:
 - Revert Changes to Style — this option will eliminate all the unsaved formatting changes made to any state for any part of the object, restoring it to the previously saved settings.
 - Save as New Style — this option will leave the current style unchanged and save all the object's current format settings as a new style, which you name as part of the process.
 - Save Changes to Current Style — this option will update the current style with all the format settings for the selected object. This, in turn, will update the format settings of all objects on the layout that share the same style and the menu icon next to the Theme in the Inspector will turn red indicating the presence of unsaved changes.

5. Choose one of the following *theme actions* from the menu:
 - **Revert Changes to Theme** —this option will eliminate all the unsaved changes made to any style on the current layout, restoring every object’s format settings to those last saved to the theme.
 - **Save as New Theme** — this option will leave the current theme unchanged and save a copy of all its styles into a new theme, which you name as part of the process.
 - **Save Changes to Theme** — this option will update the current theme with all the saved styles. This, in turn will update the format settings of all objects on every layout that shares the same theme.

Summary

In this chapter, we discussed the basics of using themes and styles to improve the efficiency of the interface design process and the performance of layout rendering.

CHAPTER 33



Customizing Menus

One of the most powerful capabilities of the FileMaker Pro Advanced application is the ability for developers to customize the menu bar and gain total control over the application experience.

In this chapter, we explore this powerful capability, covering the following topics:

- Exploring the capabilities of custom menus
- Getting started with custom menu basics
- Exploring the manage custom menus dialog
- Creating a custom menu set
- Customizing menu items
- Installing a menu set
- Exploring the link between commands and menus
- Creating status-based custom menu

Exploring the Capabilities of Customize Menus

Customizing the menu bar allows a host of new capabilities over the standard application experience. These include:

- *Adding New Menus* — Add new menus with items that execute custom scripts.
- *Modifying Key Commands* — Add, remove, or change key commands for existing menu items.
- *Customizing Standard Menu Names* — Rename existing menus and menu items to better reflect the user's current context within the database or to reflect familiar terminology. Custom names can be hard-coded or generated dynamically with a formula. For example, the New Record menu item can include the current layout's table name for clarity (New Contact Record) or user-specific jargon (Initiate Client Partnership).
- *Customizing Standard Menu Functionality* — Reroute a standard menu function to a custom script. For example, the Delete Record menu item can be routed to a custom script that includes a more robust warning dialog that informs the user of exactly *which record* will be deleted from *what table*.
- *Conditionally Hiding Menus* — Control when custom menus are visible based on the current table, window mode, operating system, or any other criteria. For example, removing the Delete All Records menu item is a good safety measure for users who don't require the function.

The menu customization feature allows you to define *multiple sets* of menus so that, in an extreme example, every layout *could* have a completely unique set of menus for each user group if necessary.

Getting Started with Custom Menus Basics

Before delving into menu customization, take a moment to review the objects and terminology applicable when working with custom menus, as shown in Figure 33-1.

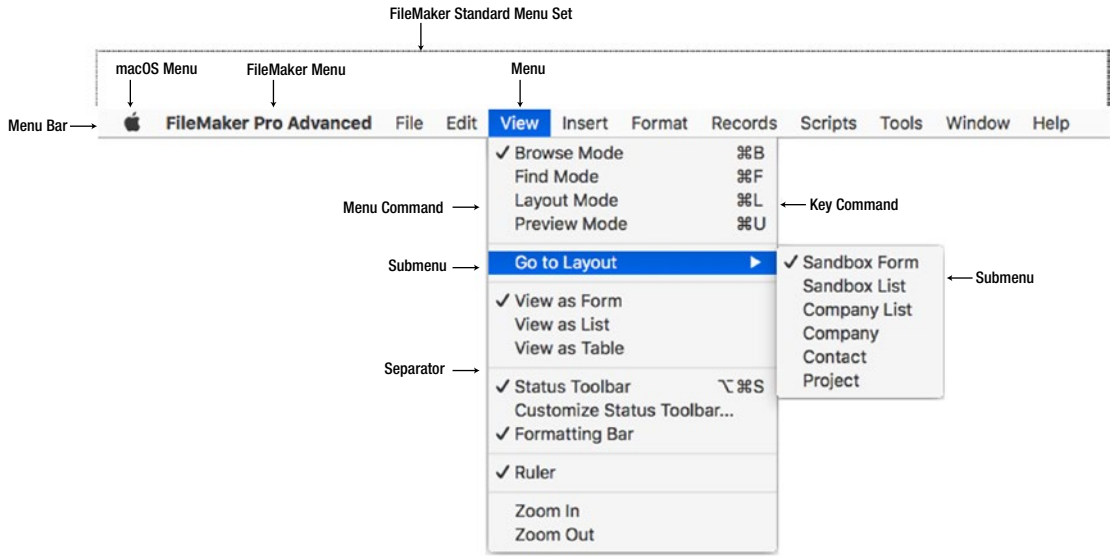


Figure 33-1. The terminology for various menu components

The *menu bar* is the horizontal strip running along the top of the screen (macOS) or the top of a database window (Windows) that displays the currently active menu set.

A *menu set* is a named collection of menus that can be installed into the menu bar, becoming the *active menu set*. Every database starts with one non-deletable and non-modifiable default menu set called [Standard FileMaker Menus], but developers can create custom menu sets to be installed in place of the standard set all the time or depending on the layout, user or other conditions. The various resources that make up a menu set can be illustrated in a hierarchical structure, as show in Figure 33-2.

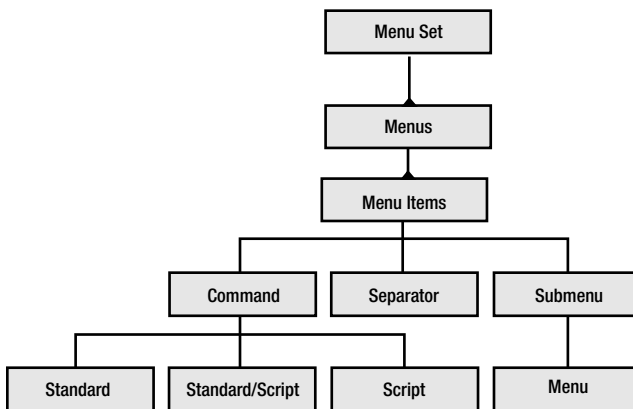


Figure 33-2. The hierarchical structure of menu components

A *menu* is a list of *menu items* grouped together under a single title, which appears in the menu bar.

A *menu item* is an item contained in a menu with properties defined by the assignment of one of three available types:

- A *command* is a menu item that triggers specific functionality: either a standard command or a custom script. These may be assigned a *key command* to provide a shortcut for triggering the associated function directly from the keyboard. Key commands can be configured using any single alpha-numeric key and a combination of the Shift, Option/Alt, and/or Command/Windows key(s) to form a unique keystroke for triggering a menu item, but only if that combination isn't reserved by the operating system or some other menu item.
- A *submenu* is a menu item with an arrow icon that, when selected, reveals a secondary menu of options subsumed under it. Submenus are useful in helping to delimit menu commands into categorized groups and avoid overwhelming the user with too many options in a single list. They can be nested so that a submenu can include item(s) that open another submenu that includes item(s) that open another, etc.
- A *separator* is a horizontal line used to separate groups of menu items to create a more visually productive organization of items.

Menu items that perform a command can be *enabled* or *disabled* based on a user's access settings. They can also be installed or not based on a formula.

Standard menu items are either *selection based*, where a click temporarily highlights the item and performs a function, or they can be *status based*, where a click toggles a check mark next to the item name to indicate it is "on" or "off." Custom menus can *only* be selection based. However, a menu can use a calculated name that can toggle between two (or more) status modes, which can simulate a status-based menu item (see example at the end of this chapter).

Exploring Custom Menu Set Options

A *custom menu set* can be constructed using the following types of menus:

- A totally custom menu, created from scratch.
- Any of three *non-customizable* standard FileMaker menus with all their standard menu items: Format, Scripts, and Window.

- Any of eight standard FileMaker menus that can be copied and customized as needed (with a few exceptions): FileMaker Pro, File, Edit, View, Insert, Records, Request, and Help.
- A choice of about two dozen *submenus* found in standard menus, which can be attached to the menu set as a stand-alone *menu* with its standard non-editable menu. These include submenus such as Open Recent, Import Records, Go to Layout, Manage, Sharing, Show Window, and more.

There are many different approaches available to design a custom menu set depending on a database's requirements. A custom set may begin as a duplicate of the *entire* standard FileMaker menu set, which is then customized, a totally *empty* set that is built from scratch, or some combination of these two.

Understanding Customization Exceptions

There are four standard menus that can't be completely removed from the application. Even a completely *empty* custom menu set, when installed, will continue to display these menus, as shown in Figure 33-3.



Figure 33-3. The absolute minimum menu bar when a custom menu set is installed with no menus defined

The following menus will always be visible, even for a completely empty custom menu set:

- Apple (macOS only) — This menu contains standard operating system functions and is completely unaffected by the custom menu set and can't be modified in any way.
- FileMaker Pro — The application menu will always be present in the menu bar regardless of it being present in a custom menu set. However, the exact configuration of this menu will vary as follows:
 - The top portion of the menu — the About FileMaker Pro item and first divider line — are *only* present if a copy of the menu is added to the custom menu set and left unmodified.
 - The remaining portion of the menu is *always* present in the menu bar regardless of a copy of the menu being present in the set or not. Also, these items are *not* shown when editing a copy of the menu in a custom menu set and, therefore, they can't be removed or customized in any way.
- Tools — This menu is exclusive to FileMaker Pro Advanced, is always included in the menu bar, and can't be added to a custom menu set or modified in any way.
- Help — The application menu will always be included in the menu bar regardless of its being present in a custom menu set. However, the exact configuration of this menu will vary as follows:
 - The top portion of the menu — the Search field — is *always* included in the menu bar regardless of a copy of the menu being present in the custom menu set or not.
 - The remaining items in the menu are *only* present if a copy of the menu is added to the custom menu set and they are left unmodified.

Exploring the Manage Custom Menus Dialog

Custom menus are created and configured in the Manage Custom Menus dialog, as shown in Figure 33-4. This dialog can be opened by any user with full access privileges using the following methods:

- Select the File > Manage > Custom Menus menu item.
- Select the Tools > Custom Menus > Manage Custom Menus menu item.
- Choose Manage Custom Menus from the Menu Set pop-up menu in the General tab of the Layout Setup dialog.

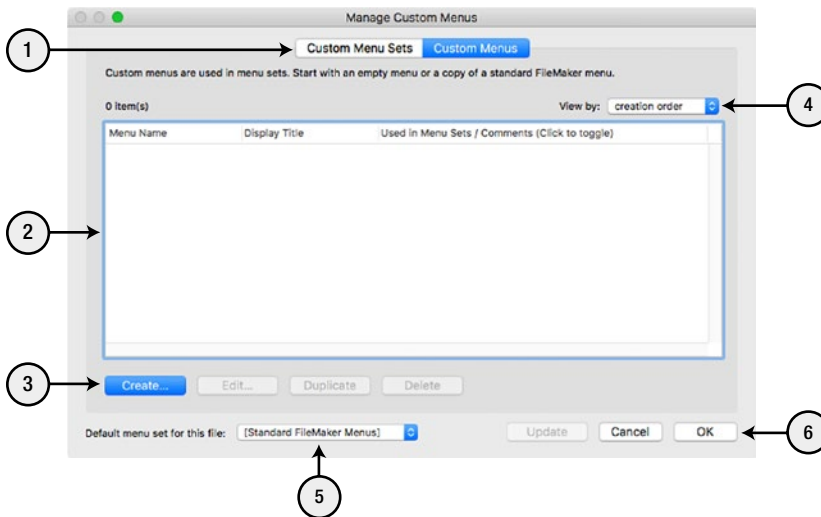


Figure 33-4. The common elements of both tabs of the Manage Custom Menus dialog

Both tabs of the Manage Custom Menus dialog contain the following controls:

1. *Tabs* — The dialog is divided into the following two tabs, each containing a list of one type of menu-related entity that can be managed with the other common elements:
 - Custom Menu Sets
 - Custom Menu
2. *Item List* — The central area on each tab contains a list of all the items defined. These can be sorted by clicking on column headings or using the View by menu.
3. *List Buttons* — Four buttons provide control over the list of items:
 - Create — Click to begin the process of adding a new item to the list.
 - Edit — Click to open the selected item in an Edit dialog.
 - Duplicate — Click to duplicate the item(s) selected in the list.
 - Delete — Click to delete the item(s) selected in the list.

4. *Sort menu* — The *View by* pop-up menu contains the available options for sorting the list of items.
5. *Default menu set* — The *Default* menu set for this file pop-up menu contains a list of all menu sets defined. The selection will become the default for the file at startup and for any layout configured to use the [File Default] menu set.
6. *Dialog Buttons* — Three buttons provide control over the dialog:
 - *Update* — Click to add any missing menus to the database when the database is upgraded from a very old version of FileMaker, which may be missing some newer items.
 - *Cancel* — Click to close the dialog without saving changes to the menu definitions.
 - *OK* — Click to save changes to the menu definitions and close the dialog.

■ **Caution** All the functions of the *Custom Menus* tab can also be accessed through subsequent dialogs opened from the *Custom Menu Sets* tab. This redundancy provides more direct access to individual custom menus but be confusing at first.

Creating a Custom Menu Set

Since the most common use of custom menus is to customize the standard FileMaker menu set, our examples in the remainder of this chapter will follow that approach. However, as mentioned earlier, it is possible to completely abandon the default menu set and start completely from scratch.

To begin, create a new custom menu set following these steps:

1. Open the *Manage Custom Menus* dialog.
2. Click on the *Custom Menu Sets* tab.
3. Click the *Create* button to open a new empty custom menu set in an *Edit Custom Menu Set* dialog, shown in Figure 33-5.

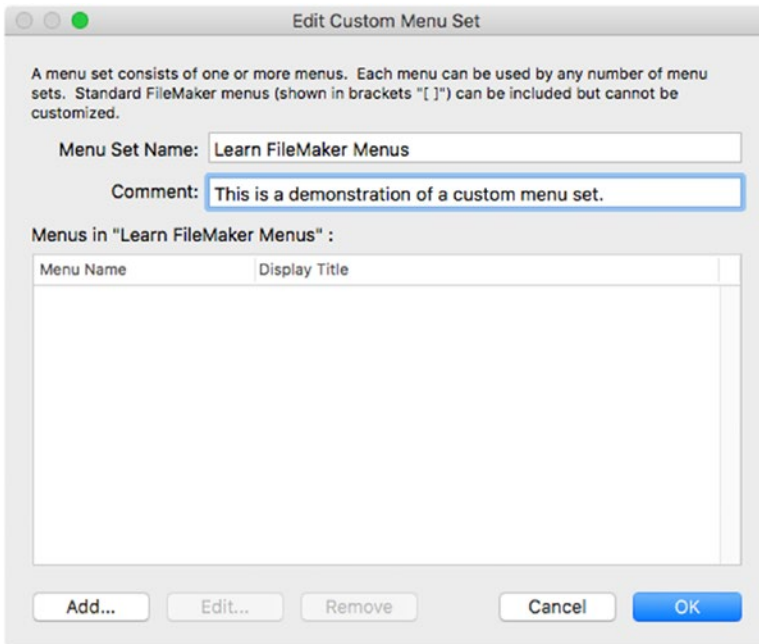


Figure 33-5. The *Edit Custom Menu Set* dialog

4. Enter a name into the Menu Set Name text area, for example, Learn FileMaker Menus, as shown.
5. The Comment area is used to enter developer-only notes about the menu set being defined. For example, this can contain information about the purpose of the menu set or the conditions for when it will be used.

Adding Copies of Standard FileMaker Menus

Next add a copy of each standard FileMaker menu into the Learn FileMaker Menu set. To get started, click the Add button to open the Select Menu dialog, shown in Figure 33-6.

■ **Caution** The process of adding standard menu items to a custom menu set is done two different ways and can be a little confusing. Three standard menus can be added directly from the default selection of menus on the Select Menu dialog. The remaining eight must be *first* copied into that list of available menus and *then* added to the set. This additional step is explained in the process to follow.

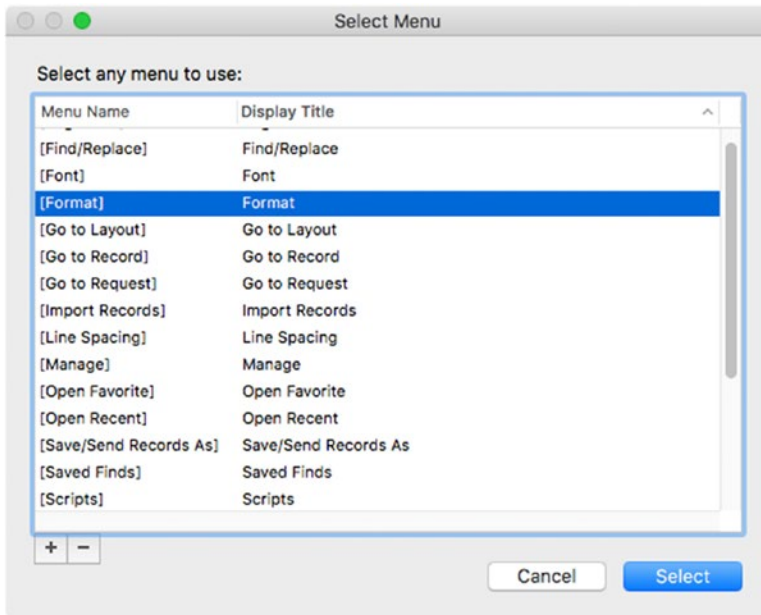


Figure 33-6. The *Select Menu* dialog lists all available standard and custom menus

The *Select Menu* dialog contains a list of every standard menu, standard submenu, and custom menu defined within the database file. For a new database without any custom menus yet defined, the list will contain only the three standard menus that can be added without customization as well as the other submenus that can be added as a menu.

To continue, hold down the Command key to allow for multiple selections, then click on the three standard menus in the list: [Format], [Scripts], and [Window]. Once selected, click the *Select* button to add the three selected menus to the custom set.

Next, to add the remaining standard FileMaker menus to the custom set, we need to create a copy of the menu by once again clicking the *Add* button on the *Edit Custom Menu Set* dialog and then clicking the + button on the *Select Menu* dialog. This will open the *Create Custom Menu* dialog, shown in Figure 33-7.

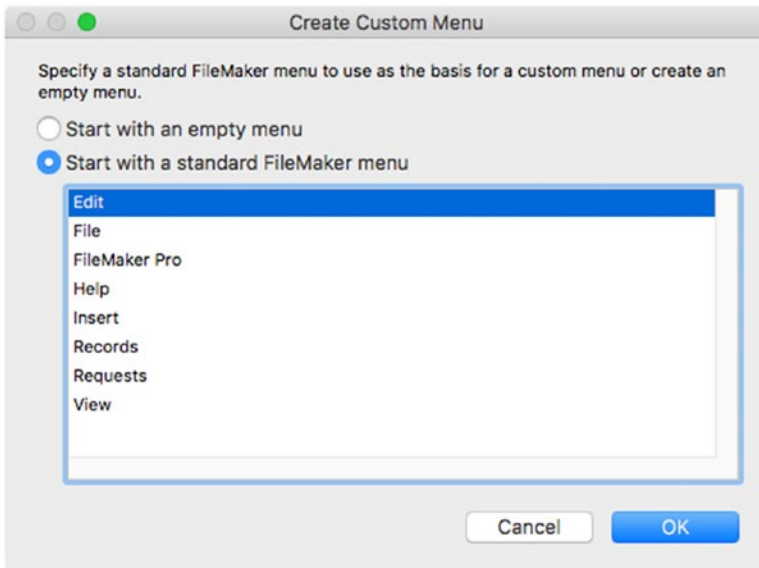


Figure 33-7. The *Create Custom Menu* dialog is used to create a new menu

The *Create Custom Menu* dialog allows the choice of two options for the new menu we are adding to the list of available menus:

- **Start with an empty menu** — Creates a new unnamed menu with no menu items.
- **Start with a standard FileMaker menu** — Creates a copy of a selected standard FileMaker menu, which can be further customized as needed.

Add each of the eight standard menus listed, one by one, by selecting them and clicking OK. For each, the new menu will be created and opened in the *Edit Custom Menu* dialog. For now, just click OK in that dialog to save the new menu copy. Once finished, the *Select Menu* dialog should contain a list of all eight copies of the standard menus, shown in [Figure 33-8](#).

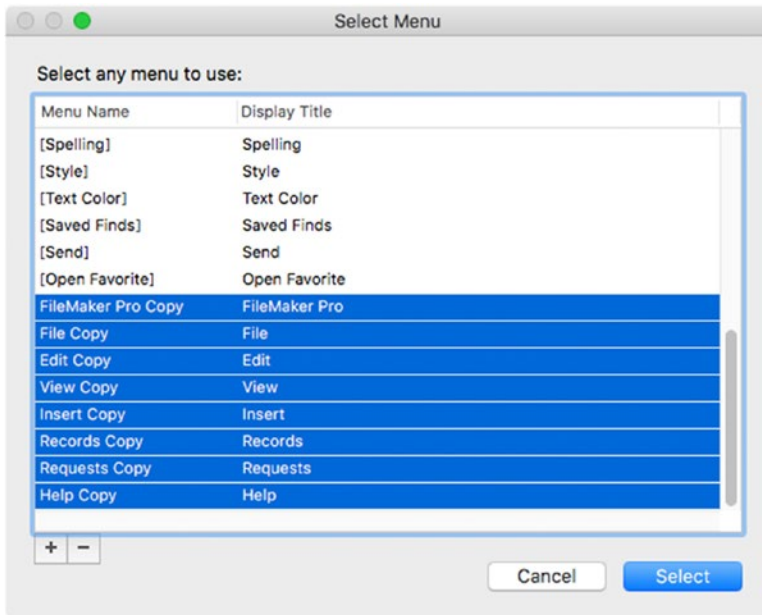


Figure 33-8. The Select Menu dialog after creating a copy of every standard FileMaker menu

■ **Tip** FileMaker will automatically name each copy of a customizable standard menu with a suffix of “Copy” as shown in Figure 33-8. If you don’t rename them as you create them, be sure to do so later. It’s always a good idea to add a common prefix to the beginning of each custom menu just to help differentiate them apart from the other non-customizable submenus in the list and to integrate them together into a group.

Holding the Command (macOS) or Windows (Windows) key, select all eight of the standard menu copies and click the Select button. This will return to the Edit Custom Menu Set dialog, showing a list of all eleven standard menu items, shown in Figure 33-9.

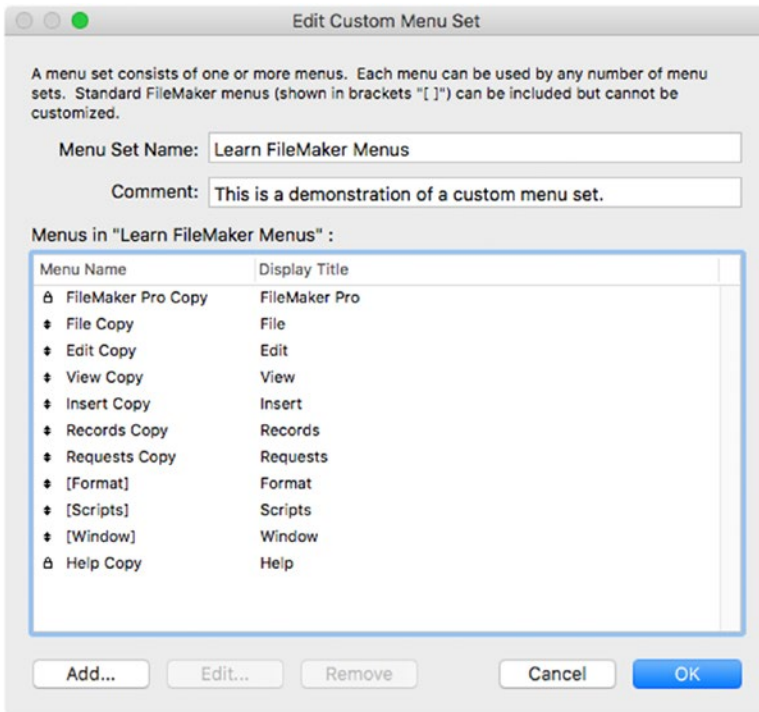


Figure 33-9. All the standard FileMaker menu items added to the custom menu set

Depending on the order in which the menus were selected and added to the custom set, they may not be in the correct order, as shown in Figure 33-9. Menus can be dragged and dropped within the list to reorder them with the following two exceptions:

- The FileMaker Pro menu will remain locked as the *first* in the list.
- The Help menu will remain locked as the *last* menu in the list.

Once the menus are in the desired order, click OK to save the changes and close the Edit Custom Menu Set dialog. Then click OK to close the Manage Custom Menus dialog. Now you can manually install the custom menu set by selecting it from the Tools > Custom Menu menu, shown in Figure 33-10.



Figure 33-10. Select the custom menu set to install it as the current menu

When the custom menu set has a check mark next to its name in this menu, it is installed. If everything worked correctly, you should notice no differences between the current menus compared to the standard FileMaker menus. Now you are ready to begin customizing the new menu set.

Customizing Menu Items

Whether editing standard menu items or configuring new empty menu items, the process of customizing a menu is essentially the same.

Exploring the Edit Custom Menu Dialog

Configuring menus and the items they contain is done from the Edit Custom Menu dialog, shown in Figure 33-11. This dialog can be opened from the Manage Custom Menus dialog one of two ways:

- From the Custom Menus tab, select a menu, and click the Edit button or double-click directly on the menu.
- From the Custom Menu Sets tab, select a *menu set* and click the Edit button or double-click directly on the menu set. Then select a *menu* and click Edit button or double-click directly on the menu.

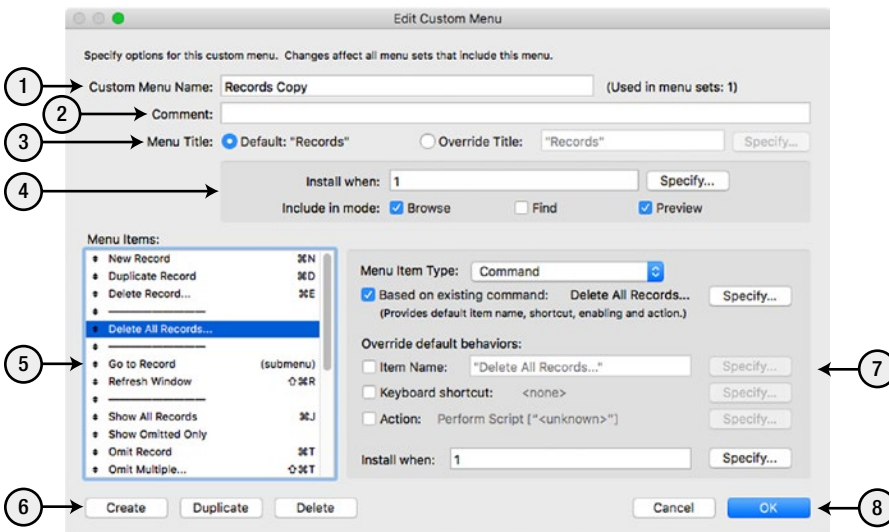


Figure 33-11. The Edit Custom Menu dialog

The Edit Custom Menu dialog contains the following controls:

1. Custom Menu Name — This text area contains the editable name of the custom menu, up to 100 characters. This name is only visible within the programming interface and is *not* the title that will appear in the menu bar. If the menu is based on a standard menu, the default name will be the menu plus a suffix of “Copy.”
2. Comment — This text area is used to store a comment describing the menu, up to 30,000 characters. Comments are only visible from dialogs within the programming interface and will *not* appear in the menu bar.

3. **Menu Title** — This set of controls allows specification of the menu title that *will* be visible in the menu bar. If the menu is a copy of a standard FileMaker menu, a **Default** option will be available to use the standard name. To override this or to enter a name for a custom menu, the **Override Title** area contains a text area to type in a name and an **Specify** button to open a dialog in which you can enter a formula to dynamically calculate a name. Custom menu names *can* be up to 30,000 characters in length; however, due to the practical limitation of the width of a screen (macOS) or a database window (Windows), as well as the nature of human perception, it is best to limit each menu name to a single word.
4. **Install Settings** — This area controls when a menu is installed in the menu bar using one of two sets of controls that work in concert:
 - **Install when** — this text area can contain a static Boolean value of 1 or 0 to control if the menu will be installed. The adjacent **Specify** button will open a **Specify Calculation** dialog into which a formula can determine the Boolean.
 - **Include in mode** — these check boxes allow a quick selection of which mode(s) the menu will appear.
5. **Menu Items** — This area contains a list of every item that makes up the menu. Reorder items by dragging them up and down within the list. Select an item to edit it in the adjacent **Menu Item Settings** area.
6. **Menu Item Buttons** — These buttons are used to manage menu items in the list:
 - **Create** — Click to insert a new menu item below the current selection.
 - **Duplicate** — Click to duplicate the selected menu item(s) in the list.
 - **Delete** — Click to immediately delete the selected menu item(s) in the list.
7. **Menu Item Settings** — This area contains all the settings relevant to the selected menu item based on the **Menu Item Type** selected in the pop-up menu at the top of the area. These will be explored in the next section.
8. **Menu Buttons** — These buttons control whether changes to the menu are saved when closing the dialog. Click **Cancel** to close the dialog without saving changes. Click **OK** to save changes and close the dialog.

Exploring the Menu Item Settings

The **Menu Item Settings** area of the **Edit Custom Menu** dialog will vary depending on the type of menu item selected.

All three menu types — commands, submenus, and separators — have one common setting, an **Install when** text area and adjacent **Specify** button. This works exactly like the similarly named feature of the menu at the top of the dialog, allowing a Boolean to control if the menu item will appear within the menu.

■ **Note** The **Install when** option is the *only* option available for Separators.

Defining Settings for Commands

The settings available for a Command menu item are shown in Figure 33-12.

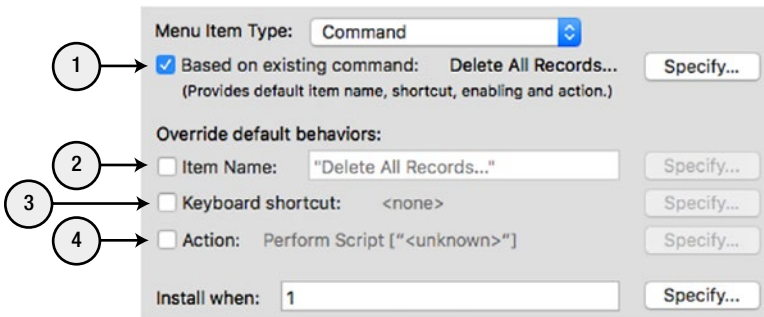


Figure 33-12. *The Menu Item Settings for a Command menu item*

The following settings are unique to Command menu items:

1. **Based on existing command** — Enable this box if the menu item will be based on an existing FileMaker command. A *Specify FileMaker Command* dialog listing all available commands will open the first time the check box is enabled. This dialog can be opened later using the *Specify* button. A menu item can be based on an existing command, run a custom action script or both. For information about the benefits of basing an item on a command even when the menu item will perform a custom script, see “Exploring the Link Between Commands and Menus” section later in this chapter.
2. **Item Name** — Enable this box to override the default command name or to enter a name for a custom menu item not based on an existing command. Then, either type a name into the adjacent text area or click *Specify* to enter a formula that determines the menu item name. Menu item names should be short but clear in describing the function to be performed and the object affected or generated. For example, “Print Sales Report” and “Send Proposal Request” are examples of good names while “Send to Accounting” doesn’t make clear what it being sent and “Get Approval” doesn’t make clear the action being taken.
3. **Keyboard shortcut** — Enable this box to override a default keyboard shortcut for a standard menu item or to enter one for a custom item. A *Specify Shortcut* dialog, as shown in Figure 33-13 will appear the first time this box is checked and can be reopened again using the adjacent *Specify* button. While this dialog is open, typing any key while holding down any key combination will assign that combination as the shortcut for the menu item. Click *OK* to save the shortcut, *Clear* to clear the currently assigned shortcut, or *Cancel* to close the dialog without saving changes.



Figure 33-13. The Specify Shortcut dialog in which a typed shortcut is captured for storage on the menu item profile

4. Action — Enable this box to assign a script or script step to the menu item. If the menu item is based on an existing command, the selected script or step will override that command's functionality. To reassign a different action, click the Specify button.

Defining Settings for Submenus

The settings available for menu items that display a Submenu are shown in Figure 33-14.

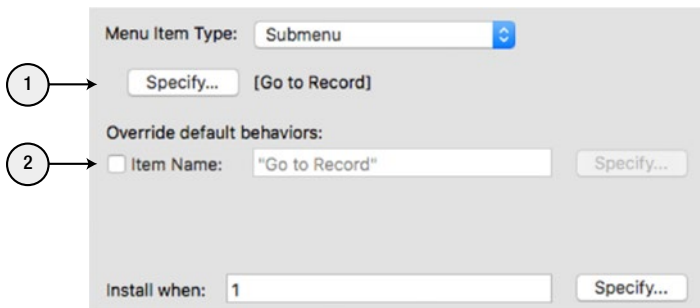


Figure 33-14. The Menu Item Settings for a Submenu item

The following settings are unique to Submenu items:

1. Specify Submenu — Click the Specify button to open a Select Menu dialog.
2. Item Name — The default name for a submenu will be the name of the menu you select. Enable this box to override that with a custom name.

Modifying Standard Menu Items

Let's work through a few examples of the most common type of menu customization that one might perform:

- Renaming a menu item
- Overriding a menu item function
- Removing a menu item conditionally

Renaming a Menu Item

One popular use of custom menus is the ability to rename menu items, making them more specific to the user's context. FileMaker's default menus are named generically and are not contextually sensitive. So, if a user is viewing several different layouts for different tables in separate windows at once, and they choose the **Records > New Record** menu item without paying attention, they might accidentally create a record in a different table than the one they intended. The easy solution to this is to modify the **New Record** menu's name to include the name of the table for the layout in the current window. This will assist a user in identifying more precisely what action they are about to undertake by reminding them of the window that is active.

To rename a menu item:

1. Open the **Manage Custom Menus** dialog and click on the **Custom Menus** tab.
2. Double-click on the **Records Copy** menu to open the **Edit Custom Menu** dialog.
3. Select the **New Record** menu item in the list.
4. Enable the **Item Name** check box to override the default name.
5. Click the adjacent **Specify** button to open the **Specify Calculation** dialog if it didn't automatically open.
6. Enter a formula to enhance the naming of the menu item. The exact formula you use may vary depending on how you name your tables. If they are named clearly like the examples we have used previously — **Contact**, **Company**, **Project**, etc. — then the following formula should produce the conditional menu items shown in [Figure 33-15](#):

```
"New " & Get ( LayoutTableName ) & " Record"
```

New Company Record	⌘N	New Contact Record	⌘N	New Project Record	⌘N
Duplicate Record	⌘D	Duplicate Record	⌘D	Duplicate Record	⌘D
Delete Record...	⌘E	Delete Record...	⌘E	Delete Record...	⌘E
Delete All Records		Delete Found Records		Delete All Records	

Figure 33-15. Examples of a conditionally named **New Record** menu

Many other menu items, especially those in the **Records** menu can benefit from a conditional name that includes the name of the table, making clear which records will be affected by the command.

Overriding a Menu Item Function

Another common use of custom menus is the ability to override standard commands with a custom script, allowing additional steps to be inserted into a standard process or to enhance a dialog's verbiage. For example, the **Delete Record** and **Delete All Record** functions both have dialogs with a rather vague prompt, the former asking simply **Permanently delete this ENTIRE record?** As with the last example, imagine a situation where a user juggling multiple windows and not paying close attention might click to continue with the deletion process before realizing they were not actively viewing the window they thought they were.

To remedy this, a script using a Show Custom Dialog step (Chapter 35) can present any number of more specific confirmation prompts, including:

- Permanently delete this entire CONTACT record?
- Permanently delete the CONTACT record for John Smith?
- Permanently delete the CONTACT record for John Smith? This will also delete all email correspondence and call logs.

Any of these prompts would help to avoid the hypothetical scenario described above.

Once you have established the desired message formula, create a script that performs the steps to ask the user, evaluate their response, and take the appropriate action or not. For example:

```
Show Custom Dialog [ "Confirm Deletion Request" ; "<message formula>" ]
Set Variable [ $Button ; Get ( LastMessageChoice ) ]
If [ $Button = 1 ] // Cancel Button
    Exit Script
End If
Delete Record/Request [ with dialog:Off ]
```

This script presents a detailed dialog message asking the user to confirm the record deletion request. The index of the button they click is placed into the \$Button variable and the script is either exited if they clicked Cancel or the Delete Record/Request is performed if they clicked Delete.

Once written and tested, override the Delete Record menu item's Action to point to your script.

Removing a Menu Item Conditionally

Another use of custom menus is to allow you to conditionally hide a menu item from certain users or under certain conditions. The Delete All Records menu item is a great example of a command that is useful for developers but might be dangerous for certain other groups of users. Experienced developers will understand the real possibility of a user thinking they are deleting one contact record and inadvertently deleting every record in the table.

To enter a formula controlling when a menu is installed, click the Specify button next to the Install when setting for the menu item. Then enter a formula that evaluates to 1 (true) for any users, privilege sets, or extended privilege who will be allowed to continue to use the function and 0 (false) for everyone else. For example, the following formula will install the menu item only for users with full access to the file:

```
Case (
    Get ( AccountPrivilegeSet ) = "[Full Access]" ; 1 ;
    0
)
```

Adding a Custom Menu

Adding a totally new custom menu to a database may be necessary when:

- Numerous scripts need to be easily accessible to users.
- They aren't conceptually commensurate with existing menus or would create unnecessary clutter there.
- They are distracting when placed directly on a layout or need to be accessible across multiple layouts.

The standard Scripts menu *can* be used for this function; however, sometimes it is beneficial or desirable to create one or more custom menus to provide a more professional solution.

For example, a database with dozens of predefined search scripts that find groups of records in different tables by status or other criteria *could* be placed under the Records. Similarly, report scripts for invoices and other tables *could* be placed under the File menu near the Print item. However, in both cases, the custom menu items would be somewhat at a disadvantage. They would add clutter to already crowded menus and would require a user to know all the various locations in which to access these custom scripts.

Adding new menus solves these problems. The exact arrangement will vary depending on the number and variety of functions requiring a presence in the menu. Using the previous example, with a large quantity of search and report scripts, a developer may add two menus: Searches and Reports. However, if the number of items in each is sparse or these are two among dozens of potential new menus, one might consider adding a single menu called Actions and having each category be a submenu within.

Creating an Actions Menu

To create a new custom Actions menu, follow these steps:

1. Open the Manage Custom Menus dialog, click on the Custom Menus tab, and then click the Create button.
2. Select Start with an empty menu in the Create Custom Menu dialog, then click OK.
3. In the Edit Custom Menu dialog, perform the following:
 - Enter Custom Menu Name, for example, Actions.
 - Enter an Override Title, for example, "Actions."
 - Specify the Include in mode settings. For this example, the Find check box will be *disabled* so a user can only see the Actions menu in Browse and Preview modes.
4. Click OK to close the Edit Custom Menu dialog.
5. Click the Custom Menu Sets tab of the Manage Custom Menus dialog.
6. Select the Learn FileMaker Menus set and click Edit.
7. In the Edit Custom Menu Set dialog that opens, click on the Add button.
8. In the Select Menu dialog that opens, scroll to the bottom and select the Actions menu, then click the Select button.
9. In the Edit Custom Menu Set dialog, drag the Actions menu to the desired location within the list of menus for the Learn FileMaker Menu set.
10. Click OK to close the Edit Custom Menu Set dialog and then OK to close the Manage Custom Menus dialog.

Now, the menu bar should have a new Actions menu, as shown in Figure 33-16.



Figure 33-16. The new empty Actions menu added to the menu bar

Adding Items to the Actions Menu

With the new Actions menu added to the menu set, you can add menu items as needed to provide access to scripts.

Adding a Command Menu Item

To add a command menu item, return to the Custom Menus tab of the Manage Custom Menus dialog and open the custom Actions menu. Then click the Create button to add a new menu item.

As a simple Hello World style example, let's create an About This Menu item that displays a dialog explaining the purpose of the menu for users by following these steps:

1. Enable the Item Name check box and enter a name of About This Menu into the adjacent text area.
2. Enable the Action check box.
3. In the Specify Script Step dialog, select the Show Custom Dialog step and configure it to display a message describing the menu's purpose and function.
4. Click OK back out through all the dialogs to save changes.

Now the Actions menu should have a single menu item that, when selected, displays a dialog describing its function, as shown in Figure 33-17.

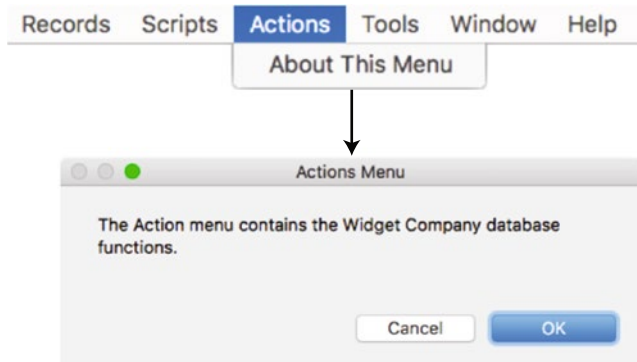


Figure 33-17. The About This Menu item of the Actions menu

Adding a Submenu Item

When adding a lot of items to a menu, if they can be separated into subcategories, consider organizing them into one or more submenus. For example, a submenu of Searches could list all the available search functions while a submenu of Reports could list all the reports.

■ **Tip** Since human perception begins to reach capacity for efficient processing of information at about 5-9 objects, use the “rule of about seven” to determine when to divide menu items into groups using separators or submenus. Try to integrate small groups of similar items together and differentiate them from other groups by either method of division.

To create a submenu, first create a custom menu containing the items that will be available in the submenu. This menu will *not* be added to the menu bar directly, but will be added to a menu as a submenu item.

For example, create new custom menu called Reports Submenu and add one menu item for each report script. Then, open the Actions menu and create a new item named “Reports...” and configure it as a submenu pointing to the Reports Submenu, as shown in Figure 33-18.

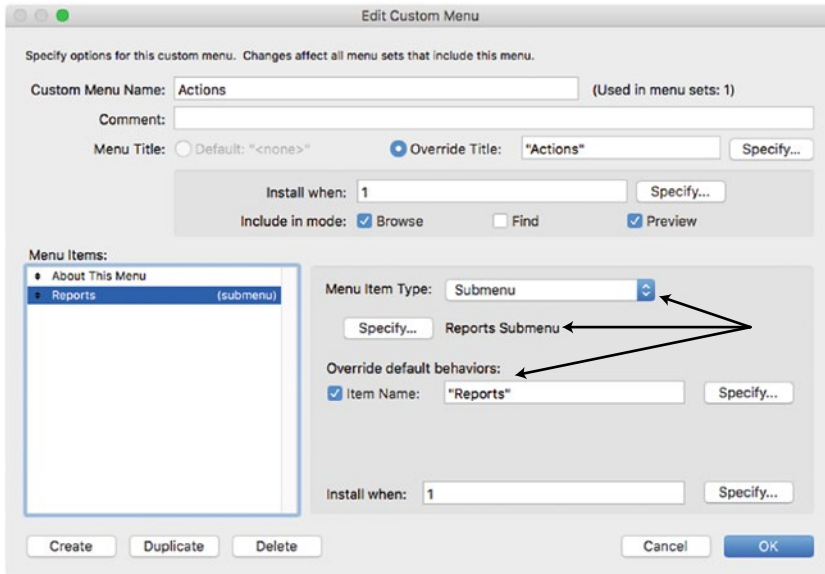


Figure 33-18. A Reports submenu added to the Actions menu

After stepping back out of the dialogs saving as you go, the Actions menu should now display a Reports submenu, like that shown in Figure 33-19.

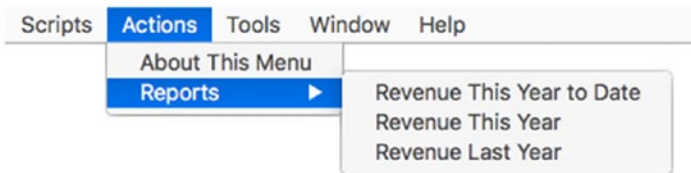


Figure 33-19. An example of a Reports submenu in a custom Actions menu

Installing a Menu Set

If you create a single custom menu, making it the default will cause it to appear for every layout configured to use [File Default] (Chapter 28, “Exploring the Layout Setup Dialog”).

However, when a solution become more complex, requiring multiple menu sets that need to change based on layout or other contextual conditions, the issue of installing menus dynamically will become important.

A custom menu set can be *installed* using the following methods:

- A custom menu set can be defined as the file's default set in the Manage Custom Menus dialog.
- Each layout can be assigned a Menu Set in the Layout Setup dialog, which will be installed whenever the layout becomes the current layout. The default choice is [File Default] that can be changed to any menu set defined in the file.
- Running a script that includes the Install Menu Set step can install any set defined in the database file and optionally make that set the default for the file.
- Users accessing the database with FileMaker Pro Advanced and with sufficient access privileges can manually select from any set defined in the file from the Tools > Custom Menus submenu.

Exploring the Link Between Commands and Menus

When using custom menus, it is important to understand the implications of the connection between standard FileMaker interface elements and the application command they make accessible. A command's representation in the menu bar influences its appearance in a contextual menu, the toolbar, and occasionally other buttons, and these are all linked to a standard command.

For example, the standard New Record command is accessible through the following elements, three of which are shown in Figure 33-20:

- The Record > New Record menu item.
- The New Record item of the layout background's contextual menu.
- The New Record toolbar button icon.
- The + button at the bottom of the list of records on a window viewed in Table mode.

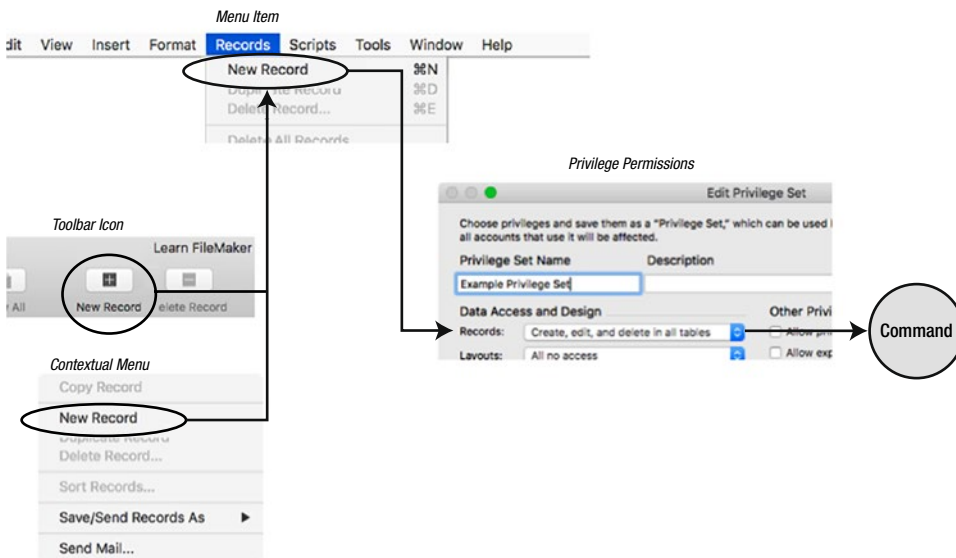


Figure 33-20. Three instances of the New Record command in the standard FileMaker interface

Ultimately, each of the interface elements points to the command that is performed when they are clicked. This connection provides the name and tooltip for the elements and runs through the security settings to determine the accessibility of the command for the current user. The visibility or enabled state of the interface elements is controlled by two factors. The interface element will be visible or enabled *only* if both conditions are true:

- The command is present in the current menu bar.
- Access to the command is permitted through the user’s security settings.

The interconnection of these interface elements must be taken into consideration when using custom menus and can be illustrated abstractly as shown in Figure 33-21.

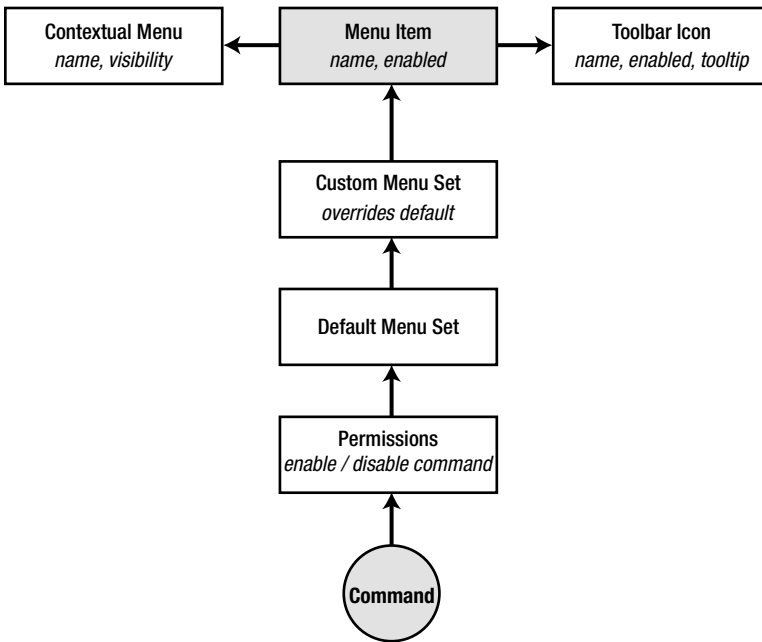


Figure 33-21. An illustration of the interconnection between a command, menu, and other standard interface elements

The elements across the top of this illustration are the visual interface manifestations of the command at the bottom. Notice how they each pull various properties from either the command, the intervening user permissions, and from either a default or a custom menu set. Each element will be affected as follows:

- *Menu Item* — will get its name from either the default or custom menu set; its enabled status from a combination of its menu settings and the privilege set defining the user’s permissions to the command it represents.
- *Contextual Menu Item* — will be visible in the interface only if the command is represented and enabled in the active menu bar; gets its name from the menu item, which comes from from either the default or custom menu set.
- *Toolbar Icon* — will be enabled only if the command is represented and enabled in the active menu; gets its name and tooltip from the menu item, which comes from either the default or custom menu.

When customizing the standard New Record menu item, assigning it a new name and configuring it to perform a custom script, that menu item will *remain* linked to the command. Therefore, the menu item will continue to enable or disable based on the current user account permissions. It will also update the custom name in the menu, the contextual menu, and toolbar, and each of these will now perform the custom script. For example, if the New Record menu is renamed New Sheet and a custom script is assigned, both the toolbar button and contextual menu will change to display this name, as shown in Figure 33-22, and perform the custom script rather than the default New Record function.

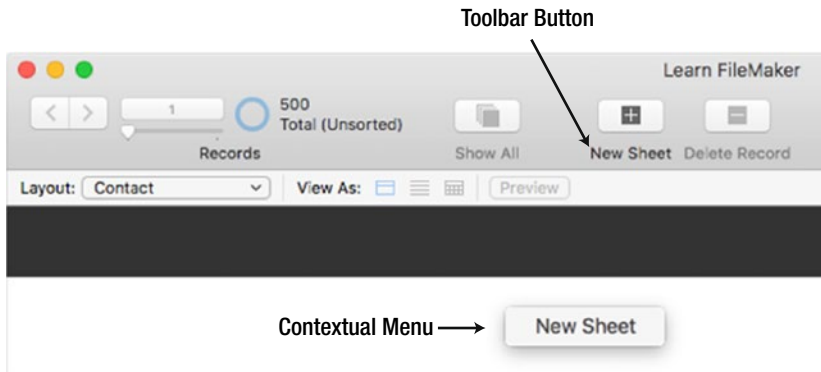


Figure 33-22. The result of renaming the New Record command in a custom menu

However, if the New Record menu item is deleted from the custom menu set and new blank menu item is added to handle new record creation, it will *not* automatically be linked to the command. Therefore, the new menu item will not automatically enable and disable based on permissions and its presence in the menu will not pass properties to the toolbar button or contextual menu, resulting in these becoming disabled or invisible. This is true *even* if the new menu item is assigned the standard name of New Record.

For a new menu item to act as the hub of interface elements described above, it must be explicitly assigned to the command using the Based on existing command settings in the Edit Custom Menu dialog, as shown in Figure 33-23.

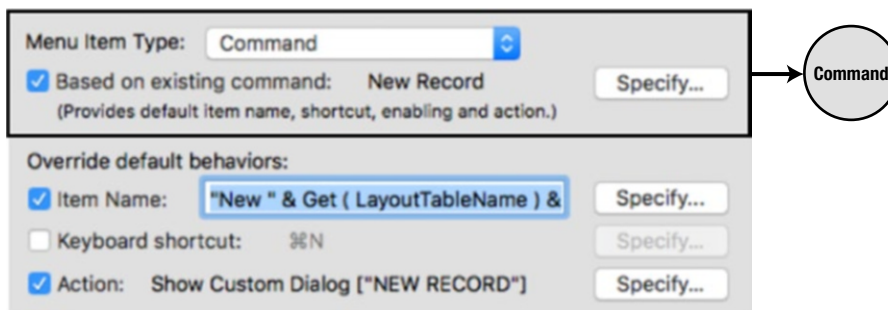


Figure 33-23. A custom menu must be explicitly linked to a command to react to permissions and enable and name other interface elements

A command menu item's relationship to a standard command can take one of three ways:

- *Standard Command Only* — When linked to a standard command *without* a name or script override, the menu will appear and perform the normal functionality associated with the command and influence other interface elements normally.
- *Script Only* — When not linked to a standard command and assigned a custom name and a script, the menu will perform the script as its function and any other command-related interface elements will be disabled or invisible.
- *Both Standard and Script* — When linked to a standard command *with* a name or script override, the menu will appear and perform a script based on the custom settings and will be linked back the standard command for permission and influence other interface elements similarly.

It is vitally important to understand this method of integration to provide users with access to the full range of built-in tools and convenience when customizing naming or behavior.

Creating a Status-Based Custom Menu

As mentioned earlier, some standard menus are *status based*, meaning they indicate the on-off status of some mode of operation. The View menu has several examples, as shown in Figure 33-24.

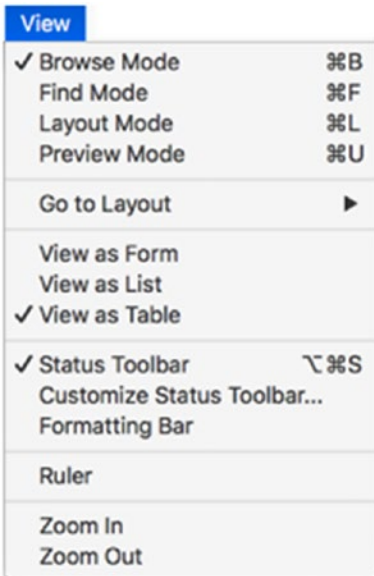


Figure 33-24. The View menu has several status-based menus, denoted with a check mark when active

Some status-based menu items can allow multiple choice status menus, where there are more than two options. For example, one of the four available modes at the top of the View menu will always have a check mark next to it indicating it as the active window mode. Similarly, one of the three View as options will be marked indicating the window view type selected for the active window.

Other status-based menus are a single menu indicating a setting as being on or off. For example, the `Status Toolbar` menu item displays a check mark when the toolbar is visible and no check mark when it is not visible. The single menu item can be selected to toggle this choice back and forth.

When using custom menus there is no a built-in option to make one or a set of menu items status based, and there is no way to include a check mark in the proper position to simulate such a state. However, there is a technique available to approximate a status-based menu where the *name* of the menu changes to indicate the status. Since a menu item's name can be the result of a calculation, it is possible to have the name change to indicate the current state and create a toggle effect each time a user selects the item.

A menu item might alternate between two names: `Enable <Mode>` and `Disable <Mode>`. This can be accomplished using a global variable to store the current state. The menu item's name calculation looks to the global variable to alternate the name while other resources or scripts looks at the global to determine whether the mode is active or inactive. The script assigned as the menus action toggles the global variable, which changes the state and the menu item's name.

Working through an example, let's create a menu item that allows users to control whether tooltips on layout objects will be displayed or suppressed. This is a practical example since new users will tend to need tooltips to guide them but more experienced users might find them annoying. So, imagine a single menu item that will toggle between `Enable Tooltips` or `Disable Tooltips` implemented by following these steps:

- A global variable named `$$Mode_Tooltips` will be initialized to a value of 0 by an `OnFirstWindowOpen` script trigger. This will indicate that the tooltips feature is "off" by default when the database file opens.
- Every layout object that has a tooltip will check this variable to determine if it should or should not display a tooltip, using this formula:

```
Case ( $$Mode_Tooltips = 1 ; "<tooltip text>" ; "" )
```

- A custom menu item, perhaps under the `File` menu or elsewhere, will use a `Name` formula that determines a name based on the opposite of the current state. If the global variable has a value of 0 indicating tooltips are *off*, its name will be `Enable Tooltips`; if the global variable has a value of 1 indicating tooltips are *on*, its name will be `Disable Tooltips`, using this formula:

```
Case ( $$Mode_Tooltips = 1 ; "Disable Tooltips" ; "Enable Tooltips" )
```

- The custom menu item will run a script that toggles the value of the `$$Mode_Tooltips` variable, using this formula:

```
Case ( $$Mode_Tooltips = 1 ; 0 ; 1 )
```

Now, when the user selects `Enable Tooltips`, the global variable is assigned a value of 1, and tooltips begin appearing whenever they hover the cursor over objects and the custom menu item's name changes to `Disable Tooltips`.

Summary

In this chapter, we introduced the menu customization options available with FileMaker Pro Advanced to take full control of the application experience.

PART V



Automating Tasks with Scripts

Scripts can be used to automate data manipulation and interface navigation to improve efficiency and provide a convenience for users. These chapters explore scripts, script triggers, and plugins.

34. *Introduction to Scripting* — Introduces the basics of how to create and manage scripts.
35. *Common Scripting Tasks* — Provides many examples of popular scripts.
36. *Debugging Scripts* — Explores how to use the debugger to test and troubleshoot scripts.
37. *Using Script Triggers* — Introduces script triggers to create responsive interfaces.
38. *Extending Features with Plugins* — Introduces how to install and use plugins to expand script capabilities.

CHAPTER 34



Introduction to Scripting

FileMaker has extensive scripting capabilities, making possible a richly interactive interface with automation-assisted functionality. Tables *store* information; layouts *display* information; and scripts *navigate, analyze, manipulate, repurpose, and share* information.

A *script*, sometimes called a macro, is a developer named and assembled sequence of customizable action steps that are stored for later execution. Scripts can perform simple tasks like navigating, searching, sorting, and inserting. They can perform complex tasks that manipulate or compile information from numerous fields across found sets of records from multiple tables. They can even be integrated across multiple database files. Any repetitive, non-creative, logically definable task that a person can perform manually within a database, a script can perform automatically.

In this chapter, we will introduce scripting, covering the following topics:

- Identifying the benefits of scripting
- Exploring script basics
- Exploring the script editor interface
- Writing scripts

■ **Note** The number of scripts in a file is limited by disk space. The number of scripts that can be displayed in the Scripts menu is 512 on Windows and 32,767 on macOS.

Identifying the Benefits of Scripting

There are many benefits of using scripts to automate database functions. These include:

- *Saving Time* — Any functionality that can be encoded into a script can be run quickly, saving a user time and reducing effort.
- *Improve Consistency* — Once a script is configured and tested, it will perform exactly the same each time it is run without deviation. This ensures that repetitive and complex processes are reproduced exactly each time they are performed.
- *Reduce Errors and Omissions* — Unlike humans, scripts never become distracted or forgetful. They will perform the tasks put to them without error or omission, over and over, as often as required.

- *Refocus on Quality* — By freeing users from onerous procedural details that can be encoded into a script and reproduced automatically, users are free to focus more effort on improving the quality of their work.
- *Improved Performance* — Scripts run without any hesitation so they can perform complex tasks much faster than a person. For added benefit, when a database is hosted on a FileMaker Server, scripts that don't require user input and don't care about the current context can be run on the server, thereby offloading the task away from the user's computer, freeing their workspace for manual work.
- *Scheduled Automation* — When used with a FileMaker Server, scripts can be scheduled to run automatically on a schedule. This means that maintenance functions can be automatically triggered at night or on weekends, without any intervention from users.

Exploring Scripting Basics

Scripts can be run by any of the following methods:

- A user clicking a button on a layout.
- A user clicking a custom menu.
- An interface event occurs that is assigned a Script Trigger.
- Another FileMaker script running it as a sub-script.
- An external operating system scripting language: Apple Events (macOS) or ActiveX (Windows).
- A FileMaker URL, internally or externally.
- A schedule on a FileMaker Server.

When any of these objects or events triggers a script, it executes the sequence of steps that make it up and optionally returns a result, as shown in Figure 34-1.

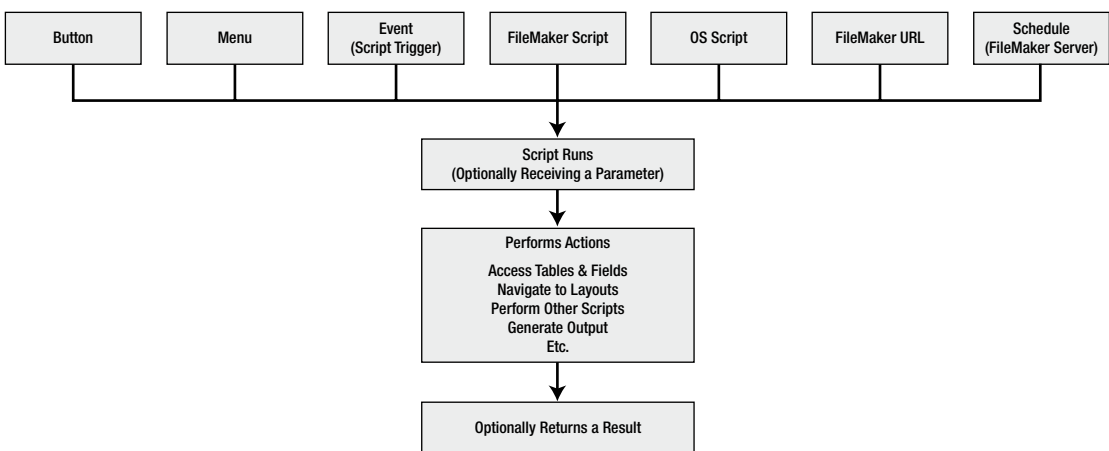


Figure 34-1. The script flow: an object or event runs the script, which, performs actions and optionally returns a result

Calling Other Scripts

Using the Perform Script step, a script can run other scripts. After adding the step to a script, it will appear with an unknown script specification. Clicking on the parameter opens a Specify Script dialog where you select a script and enter a parameter, as shown in Figure 34-2.

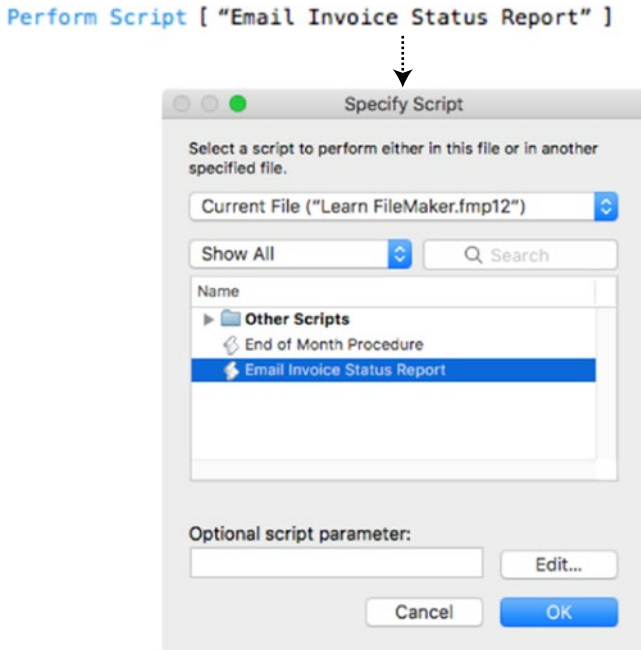


Figure 34-2. The Perform Script step and Specify Script dialog are used to instruct a script to run another script

■ **Note** The Specify Script dialog was explored in detail in Chapter 30, “Configuring a Button to Perform a Script.”

Script calls can be compounded and nested in several ways:

- One script can call one or more scripts directly.
- One script can call another that calls another, forming a “stack.”
- Some combination of the two, as shown in Figure 34-3.

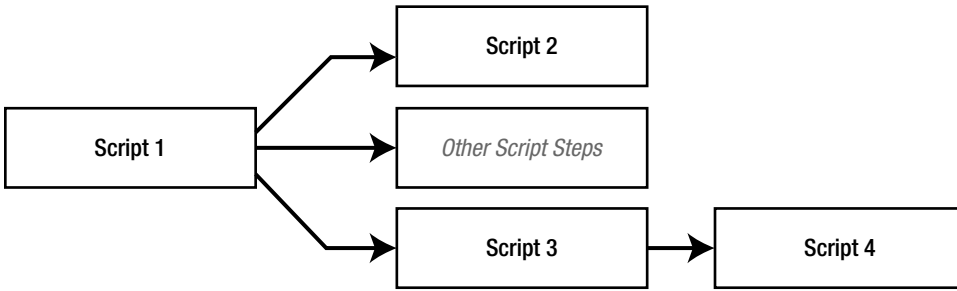


Figure 34-3. An illustration of a script calling other scripts, both directly and in a stack

The ability of a script to run other scripts and exchange data back and forth, allows scripts to be more delimited and less monolithic. Instead of building one massive script that performs dozens or hundreds of steps, the process can be broken into separate scripts, each focused on a certain task. This allows scripts to be designed like subroutines or functions in object-oriented programming languages. It makes it possible to setup a hierarchy where some scripts perform open-ended, fundamental tasks that can be shared by many other scripts regardless of context, some managing the top-level process for a specific layout, and others somewhere in between.

Exchanging Data between Scripts

Each interaction between two scripts includes the option to *send* a parameter to the sub-script and/or to *receive* a result, as shown in Figure 34-4. This allows for robust interaction between a set of delimited script functions.

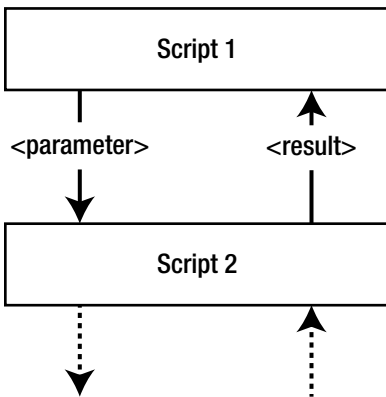


Figure 34-4. An illustration of a script pushing a parameter to a sub-script and then receiving a result

Defining Script Parameters

A *script parameter* is a text string provided to the script by the triggering object or event that is used to communicate variable information or instructions. The script may use the information for a variety of different reasons. For example, parameters can vary one or more functions of a script or provide raw material for processing. Instead of hard-coding field references or other values within the script, requiring it

to reach out and *pull in* information, a script parameter allows the triggering object to *push in* information so that it can vary from one run request to the next. Using parameters allows scripts to be written in a flexible, open-ended way with some or all the functionality being controlled or fed information from the triggering object or event.

A simple example to illustrate the use of parameters is a script that displays a dialog. The information can be hard-coded into the script's Show Custom Dialog step or it can be pushed in as a parameter, allowing each calling object to provide a unique message, as illustrated in Figure 34-5. In this example, clicking either button would display a different message even though they are running the same script.

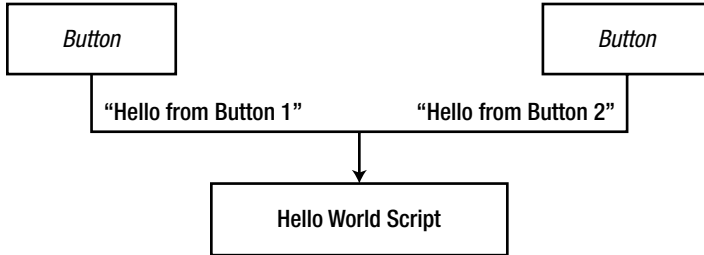


Figure 34-5. An illustration of a simple script parameter changing the message the script will display in a dialog

Admittedly, this example is far too simplistic to adequately demonstrate the full power and benefit of using parameters. The data transmitted as a script parameter can vary in complexity as needed, including any of the following:

- A single word.
- A sentence.
- A field reference.
- A list of paragraphs carriage return-delimited values containing a mixture of raw information or instructional commands.
- A JSON object containing an array of key/value pairs containing structured information.

The script receiving parameters must be explicitly programmed to access the input using the `Get (ScriptParameter)` function. If multiple values are received, it must parse the data into the appropriate variables.

Parsing a Script Parameter

A parameter is sent to a script as a text string that can be accessed and placed into a variable:

```
Set Variable [ $Parameter ; Get ( ScriptParameter ) ]
```

Although there is only one script parameter, it can take the form of a carriage return-delimited list to accommodate multiple values. When using this technique, it will be necessary for the receiving script to parse the values into separate variables. They can be parsed any number of ways depending on the circumstances.

If there are a fixed number of values, a `Let` statement can use a sequence of `GetValue` functions to parse out each value and place it into a variable. If the number of values varies, a `Loop` statement might be necessary to extract them one by one. For more complex situations, consider a structured format such as the JSON object, which can be easily parsed and manipulated (Chapter 14).

■ **Caution** Other methods of exchanging data between scripts includes the use of global variables or global fields. For a single exchange of information between two scripts, script parameters are the best recommendation. Global variables *can* be used and are often justified, but these should be limited to situations where the data is needed on a global scale. Using global fields for this purpose is dated, having become obsolete with the introduction of global variables and parameter passing.

Exploring Script Results

A script run by another script can send back a result value using the `Exit Script` step. The step has a single parameter: a calculation formula that results in text that will be sent back to the parent script.

The result can be as simple or complex as an incoming script parameter, ranging from a single word indicating success or failure of the process to a complex multi-value JSON object providing a detailed report or data ready to be passed on to another script for further processing. This example shows a simple result of the word “Success” indicating to the parent script that the sub-script reached the end of its steps without failure:

```
Exit Script [ Text Result: "Success" ]
```

An `Exit Script` step can be placed in multiple positions within the script workflow, each returning a different value. For example, after performing a find, a conditional `If` statement can exit the script if no records are found. In that case, the resulting string can vary to indicate what functions were or were not performed, as shown in this example:

```
Perform Find [ ]
If [ Get ( FoundCount ) = 0 ]
    Exit Script [ Text Result: "No records found" ]
End If
Loop
    <steps to process records>
End Loop
Exit Script [ Text Result: "Success" ]
```

Perform Script on Server

When a database is hosted on a FileMaker Server, there is an option to run a script locally, on the current user’s computer using the `Perform Script` step, *or* to offload it onto the server by choosing `Perform Script on Server` instead. Running a script on the server tends to be much faster since the client and server don’t need to exchange as much information across the network. In instances where the local script doesn’t require a response, the task can be fully handed off to the server, instantly freeing the user’s computer to perform other script functions or manual work.

Before using this feature, it is important to consider a couple of important points.

The context of the local computer's database window will *not* be known or accessible to the server. Therefore, any information about the current table, layout, record, found set, sort order, or window required by the script being called must be sent to the script as a script parameter so the server can replicate the context or process using the information provided about the context. New scripts can be designed specifically to address this issue. When converting an existing script to run on the server, it may be necessary to split it into two: one script that loads the context information into a script parameter and another that will receive that information and can safely run on the server.

If the script is critical to the database functioning normally at launch, it is a good idea to include an If statement to allow the script to be run on the server during normal operations but locally when the database is not hosted. For example, if you take the databases offline and open them in a copy of the client software for troubleshooting, maintenance, or migration to a newer version, you don't want to be struggling with a file that won't open properly because of scripting errors. This example demonstrates an If statement choosing between running the script locally or on the server with the help of a built-in function:

```
If [ LeftWords ( Get ( HostApplicationVersion ) ; 1 ) = "Server" ]
    Perform Script On Server [ <script name> ]
Else
    Perform Script [ <script name> ]
End If
```

■ **Caution** The Perform Script on Server step should not be confused with running a scheduled script on the server. The former is used by a script running locally on a client to request a script be run on the server; the latter is triggered by a scheduling mechanism on the server software and does not involve a client.

Understanding the Importance of Context

Like when writing calculations, understanding FileMaker's reliance on context is critically important when writing scripts. A *context* indicates the table occurrence of the active layout in the current window when a script step is executed. It can also mean the active field and the current record in that window. As a user or script navigates to different records, layouts, or windows, the context changes.

The necessity of considering context varies from one script step to the next. Some steps assume and rely on context, some will change the context, and others aren't context sensitive at all. For example, steps will:

- Use the context of the window's current *layout* when creating records.
- Use the context of the window's current *record* when deleting or duplicating records.
- Use the context of the window's current layout *and* the current record when interacting with fields.
- Change the *layout* context when navigating to another layout or creating a new window.
- Change the *record* context when navigating to other records or performing searches.
- Ignore the window context completely when performing application functions or other control functions.

When writing scripts, especially complex situations where layouts are changing, multiple windows are juggled, finds are being performed, and sub-scripts are run, you must maintain an understanding of how and what context applies with each step being performed.

Exploring the Script Workspace

Scripts are viewed, written, and managed within the Script Workspace dialog, shown in Figure 34-6. This dialog can be opened using any of the following methods:

- Select the File > Manage > Scripts menu item.
- Select the Scripts > Script Workspace menu.
- Running a script containing the Open Script Workspace step.

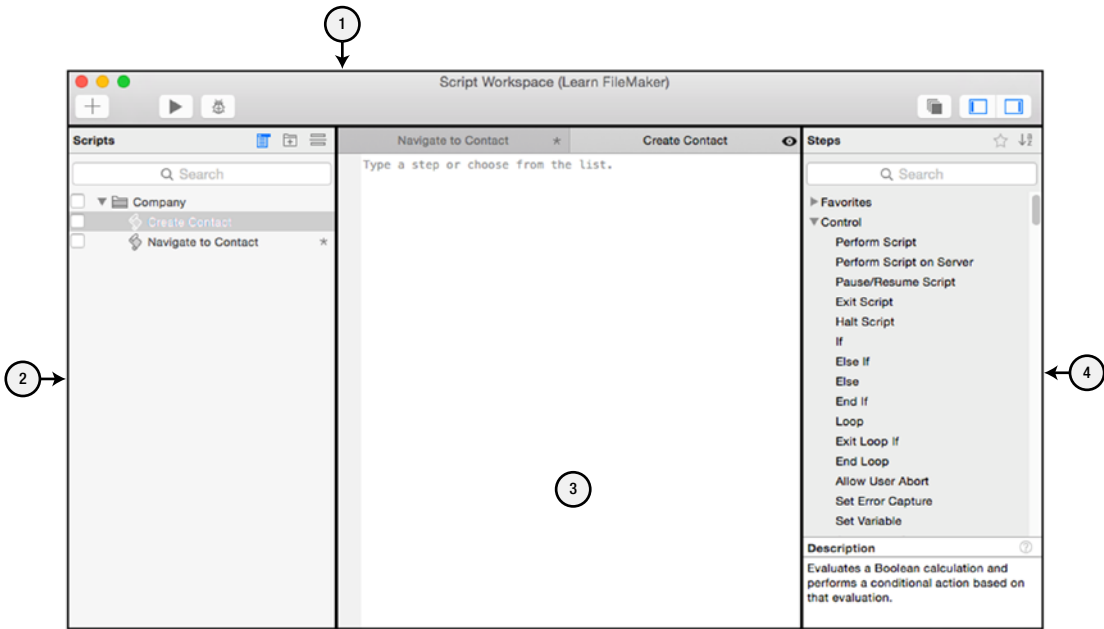


Figure 34-6. The Script Workspace dialog in which scripts are created and managed

Once open, the workspace window has standard operating system controls for minimizing and closing. The dialog is divided into the following four sections:

1. Workspace toolbar
2. Scripts pane
3. Script editor
4. Script steps pane

Workspace Toolbar

The *script workspace toolbar* of the Script Workspace dialog, as shown in Figure 34-7, is a static toolbar containing a handful of buttons used to create, edit, debug, and run scripts.

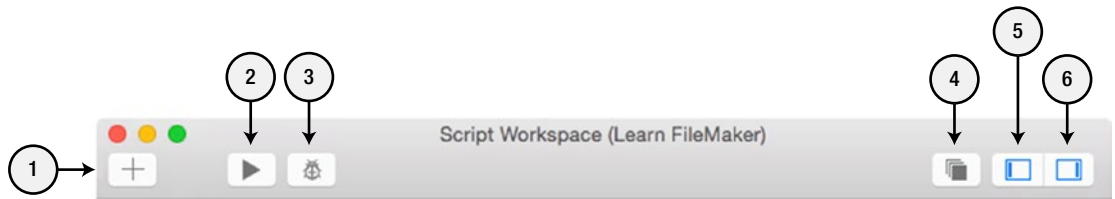


Figure 34-7. The Script Workspace dialog toolbar contains several static function buttons

The following function buttons are available in the toolbar:

1. **Create New Script** — creates a new empty script in the scripts pane below.
2. **Run Script** — runs the selected script directly from the Script Workspace using the active window's context.
3. **Script Debugger** — opens the Script Debugger window and runs the currently selected script with debugging by using the active window's context.
4. **Compatible Steps by Platform** — reveals a hidden menu of device and software platforms used to determine which script steps are compatible with a selected platform. Any script steps incompatible with the selection will be formatted in gray text.
5. **Toggle Scripts Pane** — toggles the visibility of the Script pane on the left side of the Script Workspace dialog.
6. **Toggle Script Steps Pane** — toggles the visibility of the Step pane on the right side of the Script Workspace dialog.

Script Pane

The *scripts pane* of the Script Workspace dialog, shown in Figure 34-8, displays a list of every script defined in the database file. Here scripts can be renamed, rearranged, and organized into a folder hierarchy.

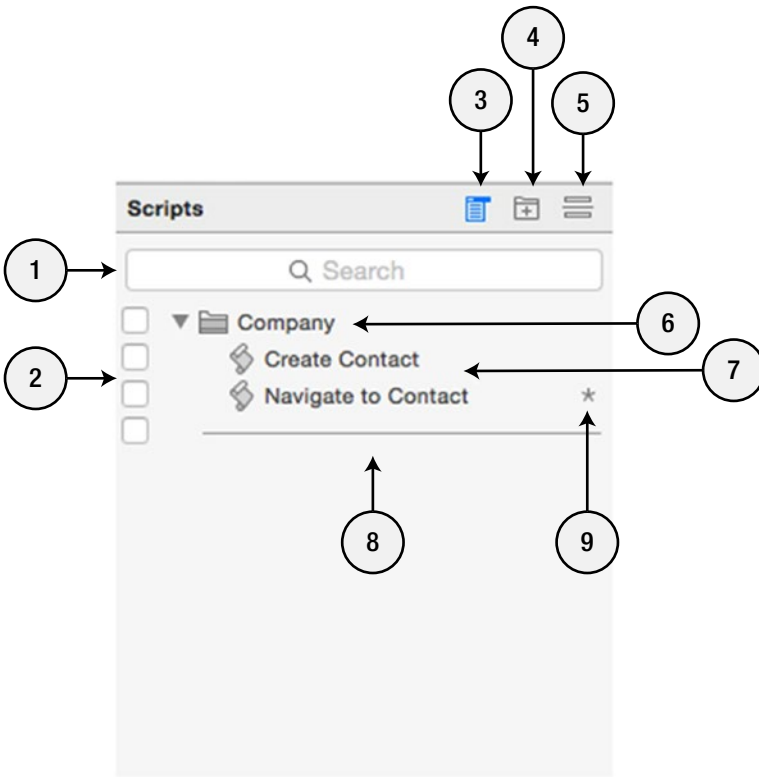


Figure 34-8. The Scripts Pane of the Script Workspace dialog

The following function buttons are available in the Scripts Pane:

1. Search Filter — filter the list of visible scripts to include only those whose name contains the value entered in this text area.
2. Scripts Menu Management — select which scripts, folders, and separator lines will be visible to the user under the Scripts menu.
3. Toggle Menu Management — toggle the visibility of the Scripts Menu Management check boxes to the left of the list of scripts. The icon is color coded to indicate its current state. When the check boxes are visible, the icon is blue; when hidden, the icon is gray.
4. Create Folder — insert a new folder into the list just below the current selection. Scripts and folders can be rearranged and grouped by dragging them into folders.
5. Create Separator — insert a new separator line into the script list just below the current selection. Separators can be placed within the list to provide divisions between groups of scripts and, if the Script Menu Management check box is selected, in the Scripts menu.
6. Script Folder — a script folder is represented in the list with a small folder icon and name. They can contain any number of scripts, folders, and separators. A single-click will allow editing of the assigned name. The folder can be opened or closed with a single-click on the disclosure triangle or double-clicking anywhere on its name.

7. **Scripts** — a script is represented in the list by a small script icon and name. There are several ways to interact with a script:
 - A click will select the script and temporarily display its contents within the editing pane.
 - A second single-click will allow editing of the script name.
 - A double-click will lock the script in editing mode within the editing pane even though no changes have been made.
 - The script's contextual menu contains a subset of functions from the Scripts menu.
8. **Separator Line** — a separator line is used to divide groups of scripts. These can be dragged up or down the list to help visually divide groups of scripts or folders.
9. **Unsaved Change Indicator** — scripts open with unsaved changes display this asterisk icon.

Script-Editing Pane

The central *script-editing pane* of the Script Workspace dialog, shown in Figure 34-9, displays the selected script and is used to view or edit the steps that make it up. Scripts are automatically opened into tabs making it easy to jump back and forth between multiple scripts; however, they can be moved into separate windows.

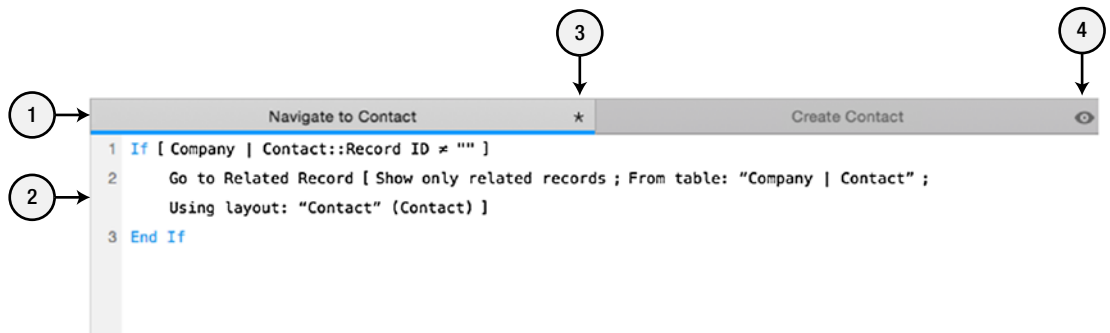


Figure 34-9. The Scripts Editing Pane of the Script Workspace dialog

The following function buttons are available in the Scripts Pane:

1. **Script Tab Bar** — each open script is represented in the tab bar. Click on a tab to bring that script forward. A small close tab icon will appear on the left when the cursor is placed over a tab, which can be clicked to close the script and remove it from the tab bar. The tab's contextual menu contains the following functions:
 - **Move Tab to New Window** — opens the tab's script in a new dedicated window to allow side-by-side viewing and editing of scripts.
 - **Select in Scripts Pane** — selects the tab's script in the Scripts pane, which can be helpful in locating a script in the list when viewing or editing multiple scripts.

- **Rename Script** — makes the script name editable within the tab. You can also double-click on the tab to edit the name.
 - **Check for Problems** — highlight in red any script steps within the script that have a detectable problem. The feature can't be turned off manually but will reset when a tab is closed.
 - **Close Tab** — close the tab after a warning to save any changes.
2. **Script Steps** — each line of a script represents one script step and is displayed in a scrollable area with automatic numbering.
 3. **Unsaved Change Indicator** — asterisk icon appears next to a script with unsaved changes.
 4. **Preview Mode Indicator** — icon will appear when a script is open in a “quick view” state and no changes have been made. In this case, the open script is displayed *temporarily* and selecting another script will replace it. Three actions will cause a script to be retained until explicitly closed:
 - Open the script and click the Preview Mode Indicator
 - Double-click on the script in the list
 - Editing the script

■ **Tip** Starting in FileMaker Pro 16, script tabs can be reordered by dragging them around in the tab bar. Also, when using the Perform Script step, clicking on the selected script name in the statement while holding the Command (macOS) or Windows (Windows) key will automatically open the script into a new tab!

Script Step Pane

The *script step pane* of the Script Workspace dialog, shown in Figure 34-10, provides an organized list of every script step available for insertion into a script, including built-in steps and those installed by plug-ins. The pane's visibility can be toggled by clicking the toolbar button. The list can be displayed hierarchically by category or alphabetically and provides search filtering and integrated help guide and hints.

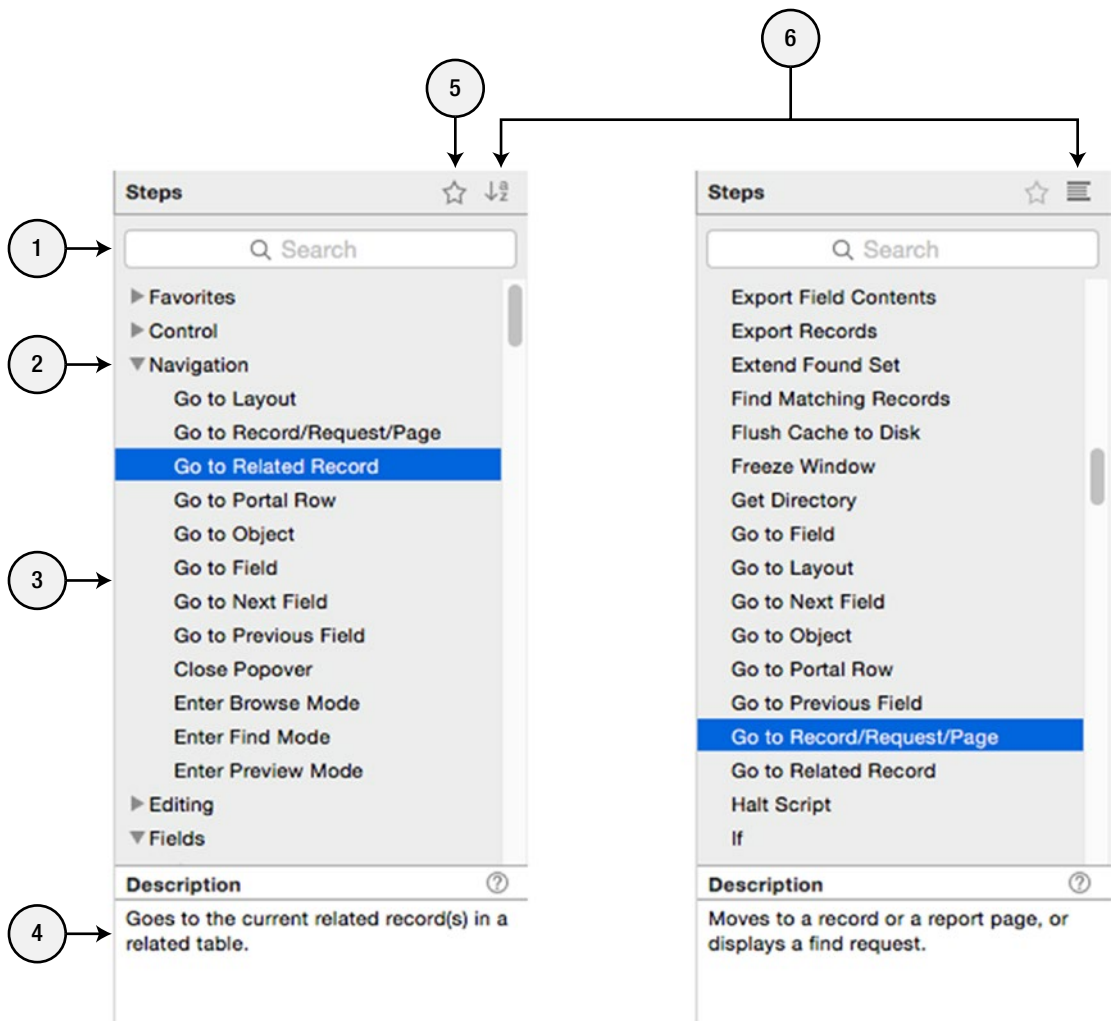


Figure 34-10. The Scripts Steps Pane of the Script Workspace dialog provides access to every available script step

The following function buttons are available in the Scripts Step Pane:

1. Search Filter — enter text in this area to filter the list of script steps to include only those whose name contains the value entered.
2. Category — when viewed in hierarchical order, every script step is grouped under a category, which can be opened and closed by clicking the disclosure triangle on the left.

3. **Script Steps** — lists all the available steps, displayed alphabetically or hierarchically, optionally filtered to only those that match the search criteria entered. The step's contextual menu contains the following functions:
 - **Insert Into Script** — insert the script step into the script. This can also be accomplished by double-clicking on the step.
 - **Add/Remove Favorites** — toggle between adding and removing the step to favorites.
 - **Lookup In Documentation** — open the FileMaker help guide for the script step.
4. **Help Guide Hint** — displays a brief description of the selected script's function. Open the full help guide by clicking the help icon on the right.
5. **Favorites Toggle** — adds or removes the selected script step from the favorites category, which is displayed at the top when viewing steps hierarchically. The icon will be gray when a selected step is not listed in favorites and blue when it has been assigned.
6. **View Toggle** — toggles the list of script steps between the hierarchical and the alphabetical view.

Script Workspace Menu Changes

When working within the **Script Workspace**, many inapplicable menus are removed and the following menus are completely transformed for dedicated to script work:

- **Edit**
- **View**
- **Scripts**

Edit Menu

The **Edit** menu, when working in the **Script Workspace**, contains a modified unique set of functions, as shown in [Figure 34-11](#).

Undo Reorder	⌘Z
Can't Redo	⇧⌘Z
Cut	⌘X
Copy	⌘C
Paste	⌘V
Clear	
Duplicate Step	⌘D
Select All	⌘A
Disable	⌘/

Figure 34-11. The Edit menu when working in the Script Workspace

In addition to standard Edit menu functions such as Cut, Copy, and Paste, the following new functions are available when working with scripts:

- Undo <last task> — reverses the last change to a script step within the current script window. The menu updates to indicate the type of change that will be reversed. For example, it might say Undo Reorder, Undo Create, or Undo Delete.
- Duplicate Step — create a duplicate of the selected script step(s).
- Disable — toggle the enabled/disabled status of the selected script step(s).

View Menu

The View menu, when working in the Script Workspace, contains a completely unique set of functions, as shown in Figure 34-12.

Hide Scripts Pane	
Hide Steps Pane	
Hide Step Description Pane	
Show Scripts Menu Management	
Close Tab	⌘W
Close All Tabs	⇧⌘W
Close Workspace	⇧⌘W
Select Next Tab	⌘}
Select Previous Tab	⌘{
Move Tab to New Window	
Compatibility	▶
Script Workspace Preferences...	

Figure 34-12. The View menu when working in the Script Workspace

The functions available in the View menu are also accessible through buttons and contextual menus within the dialog:

- Show/Hide Scripts Pane — toggle the visibility of the Scripts Pane on the left side of the dialog.
- Show/Hide Steps Pane — toggle the visibility of the Steps Pane on the right side of the dialog.
- Show/Hide Step Description Pane — toggle the visibility of the Step Description Pane at the bottom right of the dialog.
- Show/Hide Scripts Menu Management — toggle the visibility of the row of check boxes to the left of the scripts name in the Scripts Pane.
- Close Tab — close the current script tab after a save warning, if required.
- Close All Tabs — close every open script tab after a unified save warning, if required.
- Close Workspace — close the Script Workspace dialog after a unified save warning, if required for any open scripts. Any scripts that are left open when closing the workspace will remain in place the next time the workspace is opened.
- Select Next Tab — select the tab to the right of the current selection.
- Select Previous Tab — select the tab to the left of the current selection.
- Move Tab to New Window — open the current tab's script in a new dedicated window to allow side-by-simultaneous-side editing of scripts.
- Compatibility — opens a submenu with a list of device and software platforms. To see which script steps are compatible with a specific platform, select it from the menu. Any incompatible script step will be formatted in gray text.
- Script Workspace Preferences — opens a small dialog that contains preferences specific to the Script Workspace dialog. From here you can control whether new scripts are automatically visible in the Scripts menu and control the syntax coloring of steps within the script workflow.

Scripts Menu

The Scripts menu when working in the Script Workspace contains a completely unique set of functions, as shown in Figure 34-13.

New Script	⌘N
New Folder	
New Separator	
Import...	
Open Script in New Tab	⌘T
Open Script in New Window	⇧⌘T
Rename Script	
Duplicate Script	⌘D
Delete Script...	⌘X
Include in Scripts Menu	
Grant Full Access Privileges	
Run Script	⌘R
Debug Script	⇧⌘R
Save Script	⌘S
Save All Scripts	
Revert Script...	

Figure 34-13. *The Scripts menu when working in the Script Workspace*

The functions available in the Scripts menu are, with a few exceptions, also accessible through buttons and contextual menus within the dialog:

- **New Script** — create a new script.
- **New Folder** — create a new folder in the Scripts pane.
- **New Separator** — create a new separator line in the Scripts pane.
- **Import** — open a dialog to begin the process of importing scripts from another FileMaker database file.
- **Open Script in New Tab** — open the selected script in a new tab.
- **Open Script in New Window** — open the selected script in a new window.
- **Rename Script** — allow editing of the script name directly from the Scripts pane.
- **Duplicate Script** — create a duplicate of the selected script.
- **Delete Script** — delete the selected script after a warning dialog.
- **Include/Exclude in Scripts Menu** — toggle the check box that controls if the selected script appears in the Scripts menu.
- **Grant/Revoke Full Access Privileges** — toggle the setting for the selected script that allows it to run with full access privileges even when the current user does not have full access. This allows you to maintain stricter limitations on what a user can do manually within the interface while still allowing scripts to function without restrictions. When a script has been granted full access, a small icon of a person appears next to its name in the Scripts pane.
- **Run Script** — execute the current script.

- **Debug Script** — execute the current script to execute with the **Script Debugger** automatically turned on.
- **Save Script** — save changes to the current script.
- **Save All Scripts** — save changes to all scripts with unsaved changes.
- **Revert Script** — revert the current script to its last saved state.

Writing Scripts

The first time you open the **Script Workspace** in a new database file, the dialog will be completely blank except for a **Create Script** button, as shown in [Figure 34-14](#).

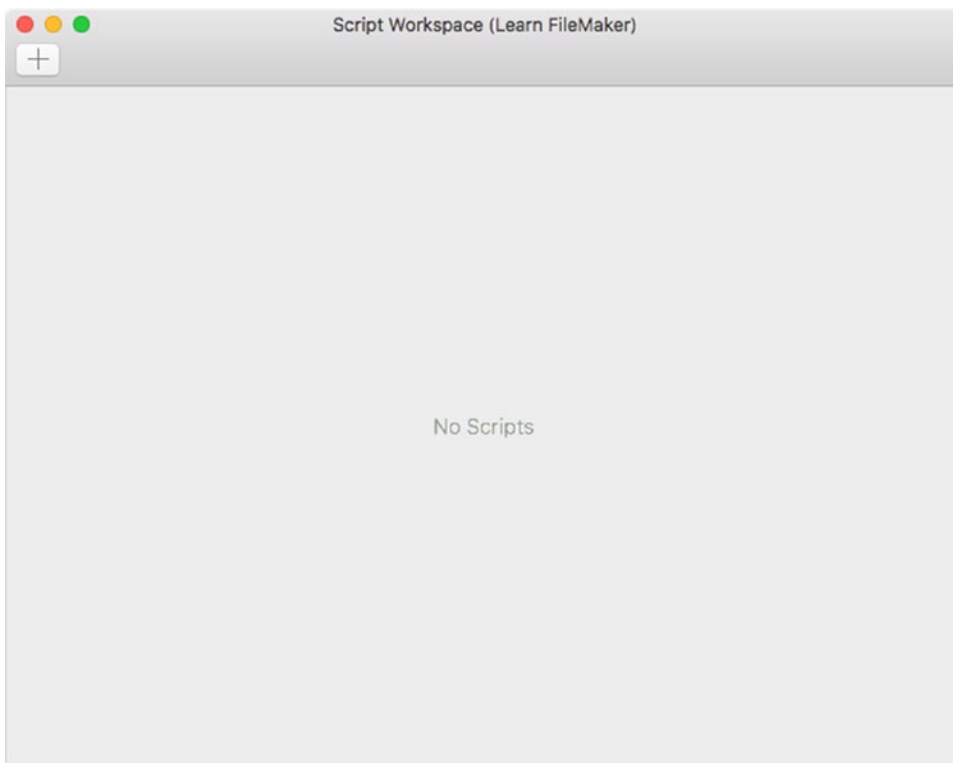


Figure 34-14. The empty **Script Workspace** dialog

To get started creating a script, follow these steps, as illustrated in [Figure 34-15](#):

1. Click the **Create Script** button to create a new script and make all the controls on the dialog visible.
2. The new script will appear in the **Scripts** pane and be selected as the only tab, with editing focus, awaiting entry of a custom name. Type the name “Hello World” for the script. Then type the **Enter** key or click away from the tab to commit this change.

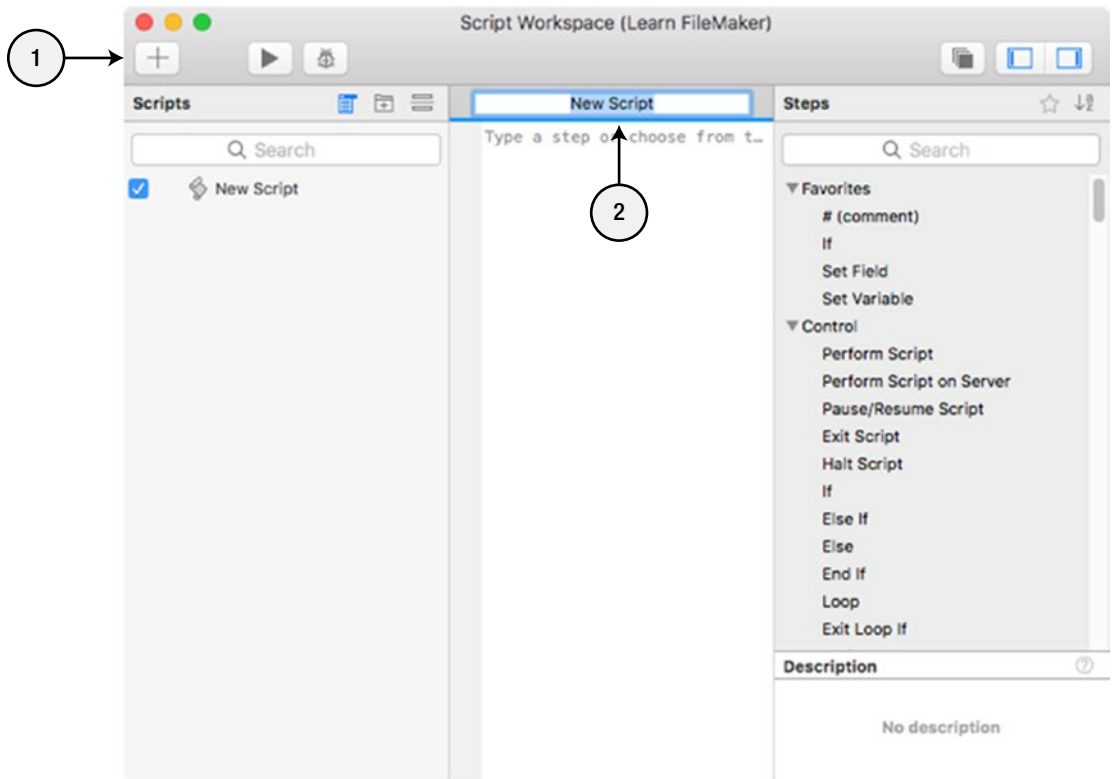


Figure 34-15. The empty Script Workspace dialog after a script has been created

Script Step Basics

A *script step* is a single function that can be inserted into a script workflow that instructs FileMaker to perform a specific action at a certain point within the overall sequence of events that make up the script. Steps are displayed in the script workflow as text-based code, appearing like a command-line programming language. However, while steps can be inserted into the flow by typing, once entered they become object-based entities that are no longer text and cannot be edited by typing. They can be selected, reordered by dragging, duplicated, copied, pasted, or deleted, but they can't be retyped to point to a different function. Those steps that have configurable options can be modified by clicking on one of a variety of different interface element types, which vary from one step to the next by the nature of the choice(s) available.

Inserting Script Steps

There are three methods available for adding steps into a script:

- Using auto-complete
- Choosing from the steps pane
- Copy/paste

It is also possible to select a script and use the Duplicate menu item.

Using Auto-Complete

Each new script will start with one blank row showing a text prompt indicating that you can either type a step or choose from the list in the Steps pane on the right.

If you begin typing a step name into this space, you will see an auto-complete suggestion interface appear that continuously filters based on what you have typed, as shown in Figure 34-16. You can ignore this list or use it to quickly enter the desired step.

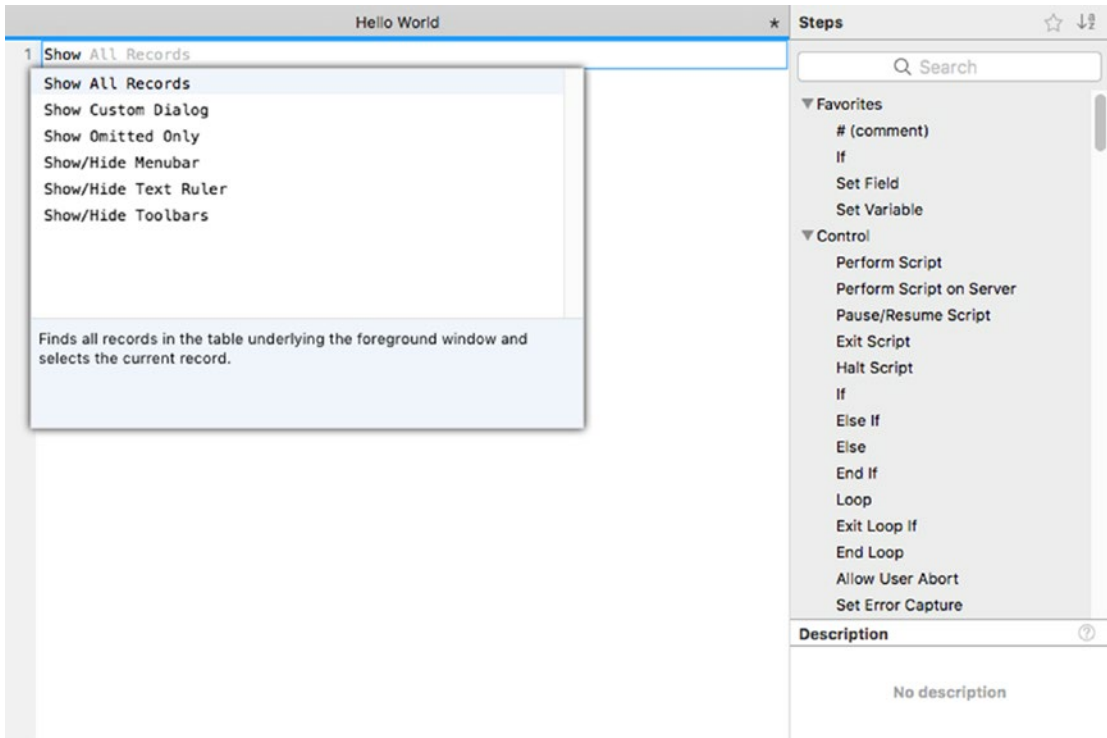


Figure 34-16. The list of suggested script steps appears as you type, offering to auto-complete the top selection.

To get focus onto the desired step, use any of these methods:

- Scroll and click directly on a desired step.
- Navigate through the list using the Up and Down arrow keys.
- Continue typing enough text until the desired item is at the top of the list.

As the selected step changes, the shaded area at the bottom of the list will update to display a brief description that will help you identify the selection. Once the desired step is selected, insert it into the script by either typing Enter or double-clicking on it in the list.

■ **Tip** For the fastest entry method using auto-complete, skip the cursor and arrow keys. Type enough of the step's name until it appears at the top of the list and then type Enter.

Once inserted, the script step is no longer text based and can't be edited as text. Instead, they become selectable objects, some with clickable interface elements used for configuration of optional behavior.

Choosing from the Steps Pane

The other way to insert a step into a script is using the Steps pane on the right side of the workspace window.

To begin, locate a step in the pane using any of the following methods:

- *Scrolling* — use the cursor to scroll up or down the list.
- *Arrow Keys* — use the Up or Down arrow keys to move the selection from one step to the next.
- *Key Scroll* — click into the pane area and begin typing the first few characters of the desired step. The selection will jump down to the next step whose name starts with the text you type. Although this works when the steps are organized hierarchically by category, it works better when viewing them alphabetically.
- *Script Filter* — type into the search text area at the top of the Steps pane to filter the list of visible steps, as shown in Figure 34-17.

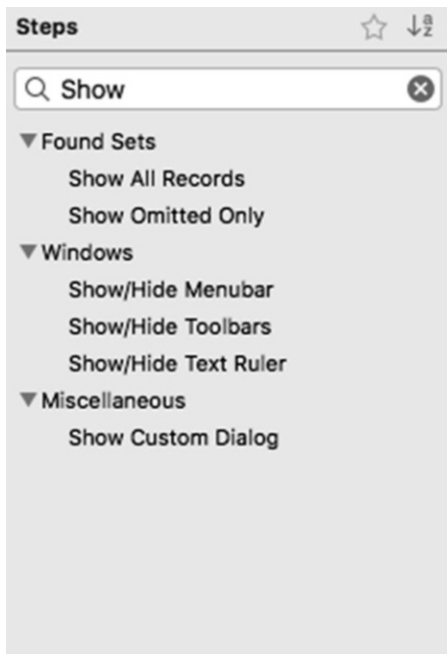


Figure 34-17. The Steps pane filtered with search criteria

Once a desired step is selected, insert it into the script workflow by any of the following methods:

- Type the Return or Enter key.
- Double-click on it with the cursor.
- Open the step's contextual menu and select *Insert into Script*.

Copy/Paste

One or more steps can be inserted into a script workflow by using the copy and paste functions. The steps can be copied from the within the same script or from another script in the same or a different database file.

To select multiple steps at once, use any of the following methods:

- To select a single step, click on it.
- To select a sequence of consecutive steps, click on the first step in the desired range and then click on the last while holding down the `Shift` key.
- To select a sequence of non-consecutive steps, click on the steps one at a time while holding the `Command` (macOS) or `Windows` (Windows) key.

Copy the steps with the `Edit > Copy` menu item. Then click anywhere into a script and paste using the `Edit > Paste` menu item.

Configuring Script Steps

There are two fundamental types of script steps: those without configurable options and those with one or more options.

Steps without Options

A script step without configurable options will perform its predetermined function without any variability. These will appear in the script workflow as a text object that simply states the function to be performed when the script is executed. Some examples include:

```
New Record/Request
Go to Next Field
Select All
Beep
Loop
End Loop
```

There is no configuration possible for these types of steps beyond including them at a particular position within the workflow.

Steps with Configurable Options

A script step with configurable options will appear with square brackets after the command name. When first inserted, they may or may not contain default settings or information between the brackets. Once configured, they will include all or some of the available settings. Some examples include:

```
Allow User Abort [ Off ]
Enter Browse Mode [ Pause: Off ]
Set Variable [ ]
Perform Script [ <unknown> ]
```

The way a step's settings are modified will vary greatly from one step to another. These can be roughly divided into two groups:

- Configuring settings directly inline
- Configuring settings with an options dialog

While each of these is described here discreetly, many script steps combine them depending on the option type.

Configuring Settings Directly Inline

A script step with *inline direct editing* of options has one or more interface components that are accessible and editable directly between the bracketed area of the step statement without opening a separate panel or window. There are three different types of inline editing control you will encounter: formula text, toggle buttons, and menus.

Inline Editing with Formula Text

Many script steps allow the entry of a formula, some of which have the option of inline formula editing. The most notable examples are `If`, `Exit Loop If`, and `Exit Script`.

A step allowing inline formula editing is identified by a red box between the square brackets. When the cursor hovers over this, the box deepens in color and gains a red outline. Clicking on this box will reveal a text area into which a formula can be typed directly. These three states are shown in Figure 34-18.



Figure 34-18. The `if` function step, showing the formula text editing control in three stages: at rest, with cursor hover, and with active focus

■ **Note** Inline formula editing is intended as a convenience for quickly entering a short formula but may be impractical for longer formulas. Therefore, this input style is *always* accompanied with an `fx` icon to the right of the active area. Clicking the icon will open a full-sized `Specify Calculation` dialog window into which complex formulas can be entered.

The text area is a fully featured editor like that found in the `Specify Calculation` dialog. It includes auto-complete suggestions and allows for multi-line formula entry and editing, both shown in Figure 34-19.

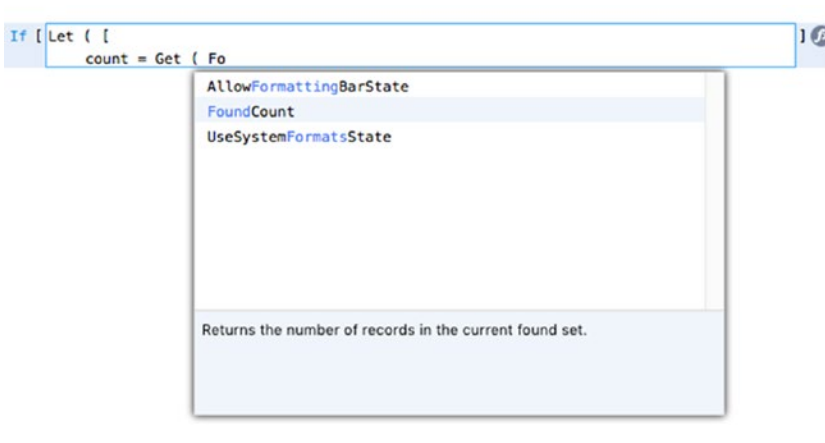


Figure 34-19. An example of a multi-line formula entered directly with the auto-complete suggestion menu visible

After typing compile and save by clicking outside of the formula area or by typing either the Return or Enter key. If a syntax error is detected, the Specify Calculation dialog will automatically open with an error notification dialog, forcing you to correct the error or finish the formula before saving.

■ **Tip** To insert a carriage return in the formula, type Return while holding down the Option key in macOS.

Inline Editing with Toggle Button

An *inline editable toggle button* allows a choice between two possible options and is identified by a labeled value between the square brackets that can logically only possess two states (On/Off) and the appearance of a blue border outlining the current value when the cursor hovers over it. Each of these states is shown in Figure 34-20.

Enter Browse Mode [Pause: Off]

Enter Browse Mode [Pause: **Off**]

Enter Browse Mode [Pause: On]

Figure 34-20. The Enter Browse Mode step, shown in stages: at rest, with cursor hover, and after a click

To change the value from its current state to the opposite, click on it or press the space bar. Also the left and right arrow keys can highlight these buttons.

■ **Tip** When this editing method is available in combination with other methods, such as a dialog or popover, the setting controlled by the toggle button is generally *only* accessible inline and *not* from the corresponding dialog.

Inline Editing with Pop-Up Menu

A step that provides an *inline pop-up menu* to quickly choose between several possible options is identified by a value between the square brackets that isn't inherently limited to two logical states and a blue border outlining the value when the cursor hovers over it. Clicking on the value will reveal a pop-up menu of containing all available options. Each of these states is shown in Figure 34-21.

Go to Record/Request/Page [First]

Go to Record/Request/Page [First]

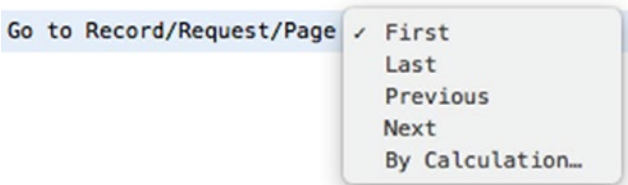


Figure 34-21. The *Go to Record/Request/Page* step, shown in three stages: at rest, with cursor hover, and with active focus

Configuring Steps with an Options Dialog

When a step's options are numerous or just not conducive to inline editing, a dialog window or popover will present the available options. Both can vary greatly depending on the step, including:

- Some dialogs are unique to their script step while others are standards seen throughout the programming interface, such as the *Specify Field* and *Specify Calculation* dialogs.
- Some present a few simple options while others summarize options with buttons that open other dialogs, to manage complexity and allow for less redundancy within the programming interface.
- Some open as independent dialog windows, while others open as a popover dialog, connected to the script step like an expanded floating menu.

The method of opening a dialog will vary from one step to the next. All popovers and most dialogs can be opened by clicking a gear icon that appears to the right of the statement when the cursor hovers over it. Others open a dialog when clicking directly on a parameter or double-clicking anywhere on the statement. Sometimes clicking will provide a shortcut to one standard dialog, which is also accessible with other controls on a dialog open through the gear menu. Many of these interface methods are combined with each other and/or conjoined with inline editing.

Open by Clicking Icon

A script step that has a dialog for the configuration of options will display a gear icon to the right of the statement when the cursor hovers over it. Clicking this icon will open some type of options dialog, as shown in Figure 34-22.

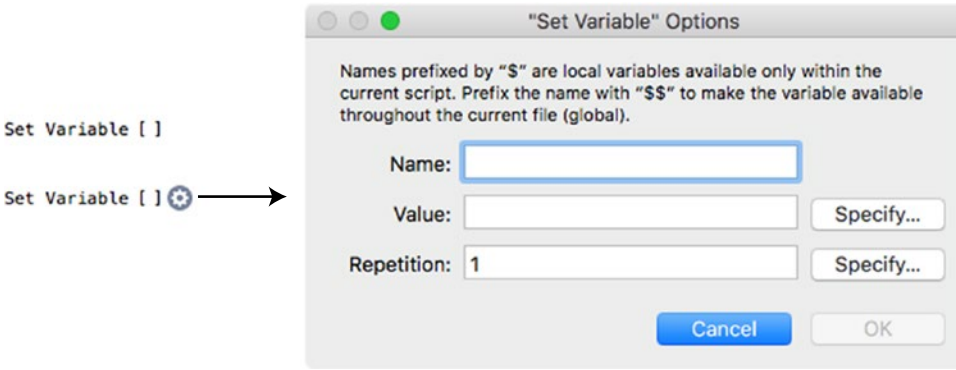


Figure 34-22. An example of an options dialog opened by clicking on the icon that appears when the cursor hovers over the statement

Options dialogs will vary greatly and contain a variety of different controls. They may contain one or more buttons that open additional dialogs. They may include one or more formula fields with adjacent Specify buttons that open a Specify Formula dialog. Once configured, clicking either the Cancel or Save button will close the dialog either discarding or saving changes respectively.

When the options are too complex to be placed inline but not numerous enough to justify a full dialog, a script step may open a small popover dialog when clicking on the gear icon or pressing the space bar, as shown in Figure 34-23.

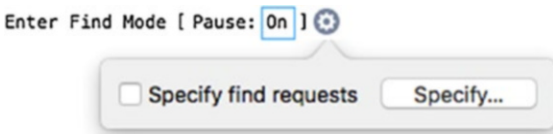


Figure 34-23. An example of an options popover dialog opened by clicking on the icon that appears when the cursor hovers over the statement

Option popovers are almost exactly like dialogs and can contain a variety of different controls. However, unlike dialogs, any changes made to the settings they contain will be saved immediately.

A popover dialog can be closed by clicking on the script workspace outside of the dialog boundaries or typing either Enter or Return.

Open by Clicking Directly

Certain script steps will open a settings dialog when clicking directly on a parameter or double-clicking on the entire script step. Usually these actions are reserved as a shortcut for quick access to one option from the full dialog. Some examples include:

- Clicking directly on a parameter that defines a field or script reference will instantly open the Specify Field or Specify Script dialog.
- Double-clicking anywhere on a step will either open the full dialog of options the same as the gear icon or provide a shortcut to the dialog's primary setting.
- The Perform Script step has no gear icon and will open the Specify Script dialog if you click on the parameter or double-click anywhere on the step's statement.
- Clicking on a field definition parameter of a Go to Field step or double-clicking anywhere on the step will open the Specify Field dialog while clicking on the gear icon will open a popover dialog with a handful of options, including a Specify field button, as shown in Figure 34-24.

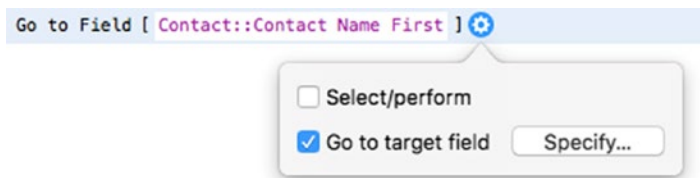


Figure 34-24. The *Go to Field* script step popover contains a handful of options while a direct-click or double-click provides a shortcut for the *Specify Field* dialog

- The *Insert Text* step, which displays several items on a popover, as shown in Figure 34-25, will open a dialog to specify the text to be inserted into the field when double-clicking anywhere on the step. In this case, the *Specify Field* dialog can be accessed *only* through the popover dialog.

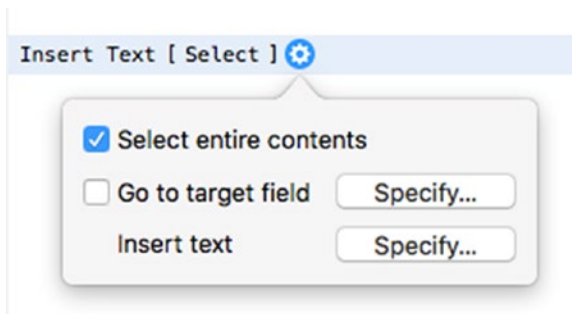


Figure 34-25. The *Insert Text* script step popover contains a handful of options while a direct-click or double-click provides a shortcut for the *Specify text* dialog

For more examples of the specifics of entering options for various steps, see Chapter 35.

Script Comments

A *script comment* is a nonfunctioning step that stores and displays developer-entered text in a script. Comments will appear as a line of text, colored green by default with a pound or hash symbol preceding it, for example:

```
# This is a comment
```

Multi-line comment can be created by typing Return while holding down the Option key in macOS.

Comments are generally used to provide developer documentation by describing the functions of an entire script or of a group of steps within a script. When combined with empty comments, they separate a section of a script into a visually pleasing delimitation of the steps, making reading, evaluating and editing a script a much easier prospect, as shown in Figure 34-26.

```
# This script will demonstrate good commenting practices

# Setup & Declarations
Freeze Window
Allow User Abort [ Off ]
Set Error Capture [ On ]
Set Variable [ $To ; Value: "Joe@FakeCompany.com" ]
Set Variable [ $Subject ; Value: "Example Database Report" ]

# Find open projects
Go to Layout [ "Project" (Project) ]
Enter Find Mode [ Pause: Off ]
Set Field [ Project::Project Status ; "Active" ]
Perform Find [ ]

# Compile the report information
Go to Record/Request/Page [ First ]
Loop
    Set Variable [ $Result ; Value: $Result & Project | Company::Company Name & " : " & Project::Project Name & "¶" ]
    Go to Record/Request/Page [ Next ; Exit after last ]
End Loop

# Email the result
Send Mail [ Send via E-mail Client ; With dialog: Off ; To: $To ; Subject: $Subject ; Message: $Result ]
Go to Layout [ original layout ]
```

Figure 34-26. A script with several comments that document the overall function at the start and then separate and describe groups of steps

■ **Tip** One creative use for script comments is as an outline mechanism. When planning a new script, create a sequence of comments that describe each step or group of steps, using the comments as “pseudo-code.” Then, as you insert and configure actual script steps, revise the comment into a group description or delete them.

Summary

In this chapter, we introduced scripting basics, working in script workspace and various methods of configuring step options.

CHAPTER 35



Common Scripting Tasks

The possibilities for scripting in FileMaker are numerous. There are over 160 script steps that are used to create scripts to perform just about any database task you can imagine. There is no way to exhaustively cover these all in a single chapter or even an entire book. Most steps have a corresponding menu function and should therefore be intuitively accessible with a relative minimum of experimentation.

In this chapter, we will work through a handful of basic real-world examples to provide a foundation upon which to base your own exploration of the remaining script steps.

Allowing User Abort

By default, a running script can be aborted at any time by the user typing Esc or a Period while holding the Command key (macOS). This isn't much of a concern for short, simple scripts that perform one or two quick steps faster than a user can notice. However, with more complex scripts, it introduces a dangerous situation that can lead to a partially completed process, the user stranded on a layout not designed as an interface or leave temporary windows open.

The Allow User Abort script step controls a user's ability to abort, with a choice of two values:

- On — every script step that follows can be aborted by the user.
- Off — every script step that follows will run without allowing interruption from the user.

Once the abort setting is configured, all subsequent script steps in the current script and any sub-scripts will inherit that setting until it is reset. This applies up and down the script stack when scripts call other scripts. So, if a parent script disallows user abort, all subsequent sub-scripts will inherit that setting unless they explicitly change it. Similarly, if a sub-script changes the abort setting, the parent script will inherit that setting when regaining control.

Any formula can check the status of the abort state using the Get (AllowAbortState) function, which returns 0 if abort is disallowed and 1 if it is allowed. These examples demonstrate turning the abort state Off and then On while checking it after each change.

```
Allow User Abort [ Off ]
Get ( AllowAbortState ) // result = 0
Allow User Abort [ On ]
Get ( AllowAbortState ) // result = 1
```

Setting Variables

Variables store information for later access in the database. They can contain any supported data type, can have multiple repetitions, and be local to a script or global to the database file. Local variables allow a script to store a value at one position for use later, which is useful when the interface context changes and fields from a starting context are no longer accessible.

■ **Caution** Local variables are *not* inherited up and down the script stack. So, when a script sets a variable and then runs another script, that sub-script will *not* have access to the local variable set in the parent script and vice versa.

Variables do not need to be initialized. If a formula attempts to access a variable that has not yet been initialized, the result will be a null string. The Set Variable script step can set the value of a local or global variable. To do this, insert the Set Variable script step into a script, then open the Set Variable Options dialog, shown in Figure 35-1, by either double-clicking on the step or clicking the gear icon next to it.

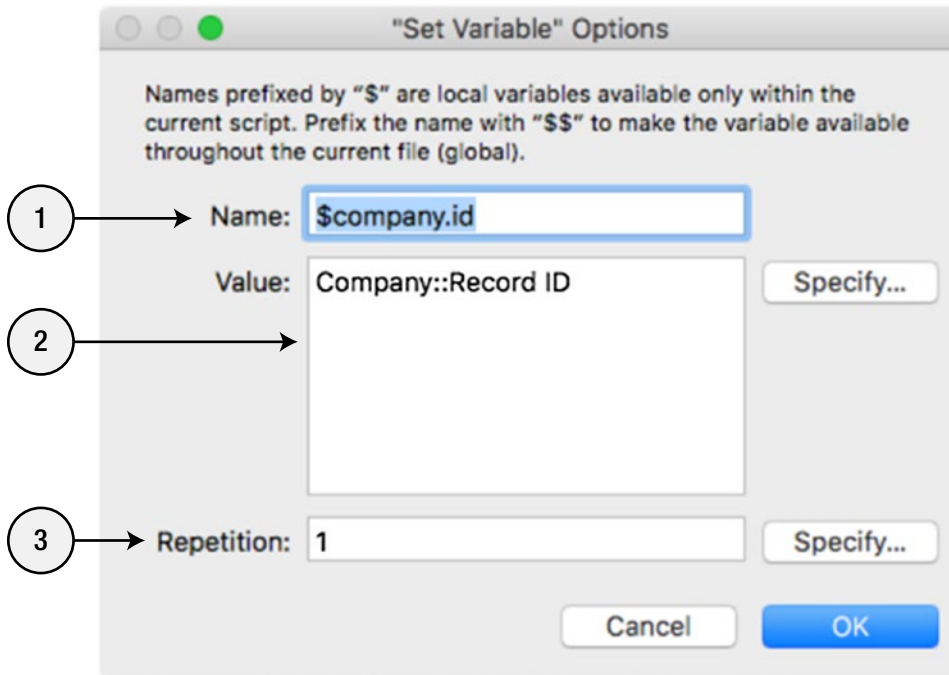


Figure 35-1. The Set Variable Options dialog

To configure a variable, follow these steps:

1. **Name** — contains the name of the variable with a single dollar sign prefix to denote a local variable and a double dollar sign prefix to denote a global variable. If no prefix is entered, a single sign will be added automatically, defaulting the variable to be local.

2. **Value** — contains the actual value stored in the variable: either enter a static value or click the adjacent **Specify** button to enter a formula in a **Specify Calculation** dialog.
3. **Repetition** — a number specifying which repetition of the named variable should receive the specified value. The default will always be 1 for a single repetition variable.

■ **Note** When using the script step to set variables, each repetition must be set in its *own* step.

This example uses the **Set Variable** step to store name of the current company record and a count of the number of contacts associated with it for use later in a dialog. These values are then compiled into a delete warning dialog message that is presented using the **Show Custom Dialog** step. Finally, the button the user clicks is placed into a `$button.clicked` variable that the subsequent script steps can use to determine the proper course of action.

```
-- Load values from fields
Set Variable [ $name ; Value: Company::Company Name ]
Set Variable [ $count ; Value: Count ( Company | Contact::Record ID ) ]

-- Construct the message and present it in a dialog
Set Variable [ $message ; Value:
    "Are you sure you want to delete the company record for " &
    Quote ( $name ) &
    "? There are " & $count & " contact record(s) associated with it!" ]
Show Custom Dialog [ "Confirm Delete Company" ; $message ]
Set Variable [ $button.clicked ; Value: Get ( LastMessageChoice ) ]
```

Creating Navigation Scripts

Scripts are frequently used to change a window's context by navigating to a different layout or record. There are three script steps used to achieve these actions:

- Go to Layout
- Go to Record/Request/Page
- Go to Related Record

Go to Layout

The **Go to Layout** script step, shown in Figure 35-2, is used to change the current layout of the current window.

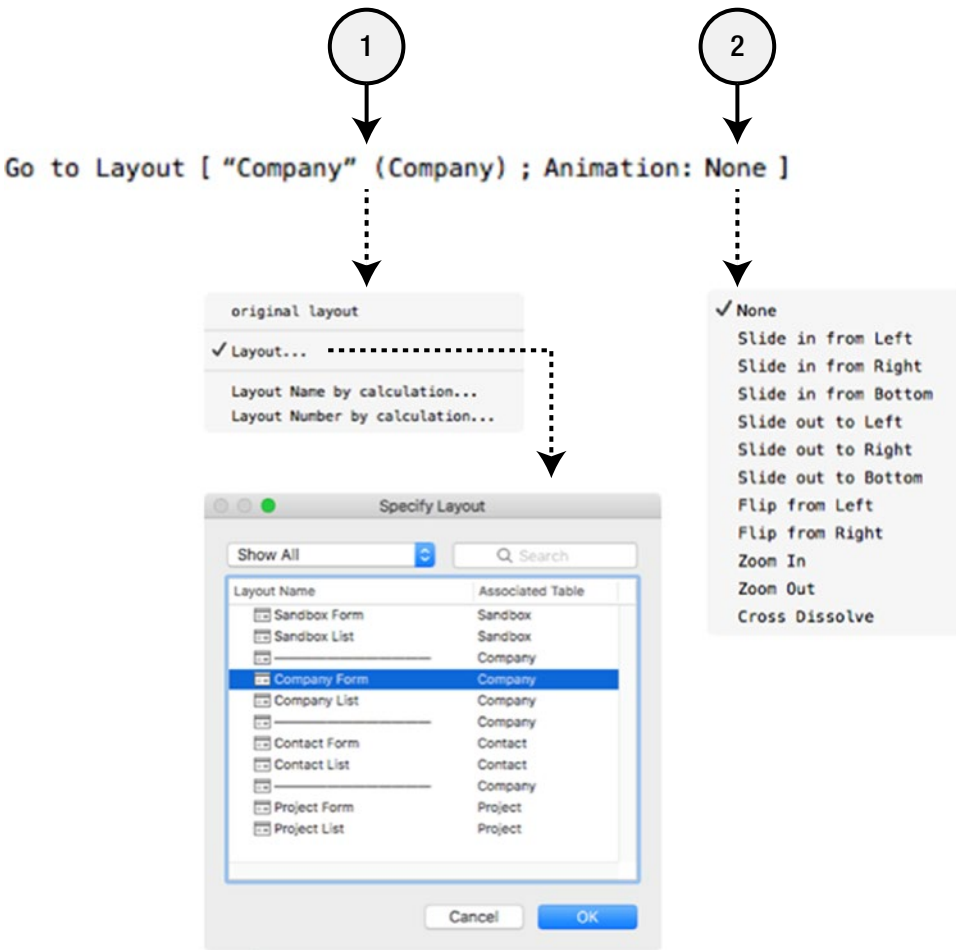


Figure 35-2. The Go to Layout script step with exploded configuration options

To configure the Go to Layout script step, follow these steps:

1. Click on the first embedded parameter to reveal a pop-up menu of the following layout options:
 - Original Layout — this default value instructs the script to navigate back to the layout that was active when the script was imitated, which is used when a script may be triggered from multiple locations in a database and needs to conditionally return the user back to their original location rather than a predetermined location.
 - Layout — opens a Specify Layout dialog in which you can select a specific layout to which the script should navigate.
 - Layout Name by calculation — opens a Specify Calculation dialog into which a formula can be entered that results in the *name* of the desired layout.
 - Layout Number by Calculation — opens a Specify Calculation dialog into which a formula can be entered, which results in the *number* of the desired layout.

2. If the database is used in FileMaker Go, click on the Animation option to reveal a menu of choices for how the layout switch should be rendered. This setting will be ignored when running the script on other platforms.

Go to Record/Request/Page

The Go to Record/Request/Page script step, shown in Figure 35-3, is used to change the content of the current window, depending on the mode as follows:

- In *Browse mode*, it will navigate to the specified record.
- In *Find mode*, it will navigate to the specified find request.
- In *Preview mode*, it will navigate to the specified page.

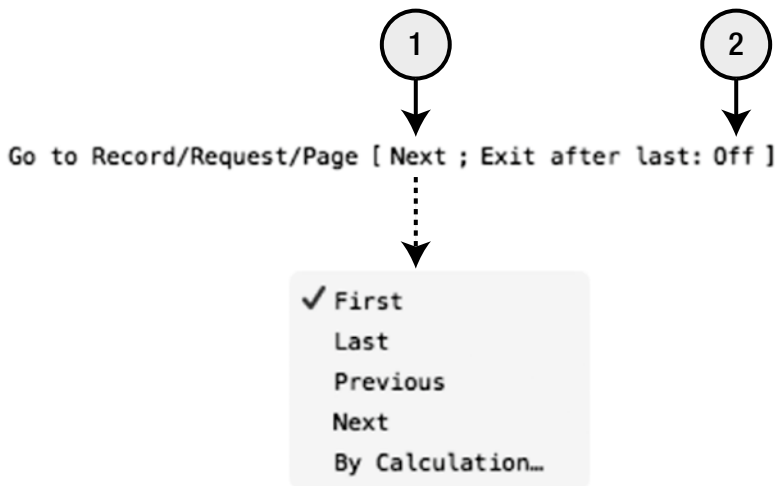


Figure 35-3. The Go to Record script step

To configure the Go to Record/Request/Page script step, follow these steps:

1. Click on the first embedded parameter to reveal a pop-up menu containing the following options:
 - First — go to the *first* record in the found set of records in the current window.
 - Last — go to the *last* record in the found set of records in the current window.
 - Previous — go *backward* one record from the current record in the found set of records in the current window.
 - Next — go *forward* one record from the current record in the found set of records in the current window.
 - By Calculation — opens a Specify Calculation dialog for a formula that results in a number indicating which record in the found set should become the current record.

2. When selecting Previous or Next, an additional Exit after Last option appears, which can be set to On or Off. If the script step is placed within a Loop function in the script and this parameter is set to On, the script will automatically exit the Loop after reaching the last available record in the direction indicated by the first parameter:
 - Previous will exit after reaching the first record in the found set.
 - Next will exit after reaching the last record in the found set.

The following example shows a hypothetical universal script that can be used for four different navigation buttons on every layout in a database. The script reacts to a user clicking on a button indicating a desire to go the first, last, next, or previous record. The buttons identify themselves and the desired navigation with a parameter that the script uses to conditionally select the clause of an If statement to take the appropriate action:

```
Set Variable [ $input ; Value: Get ( ScriptParameter ) ]
If [ $input = "first" ]
    Go to Record/Request/Page [ First ]
Else if = [ $input = "last" ]
    Go to Record/Request/Page [ Last ]
Else if = [ $input = "previous" ]
    Go to Record/Request/Page [ Previous ; Exit after last ; Off ]
Else if = [ $input = "next" ]
    Go to Record/Request/Page [ Next ; Exit after last ; Off ]
End if
```

Go to Related Record

The Go to Related Record step performs the following three functions as a single step:

- Navigating to another layout.
- Performing a find for one or more records related to the current record or to every record in the current found set.
- Optionally creating a new window in which to display the results.

To illustrate the functionality, imagine a user is viewing a Contact record and wants to view the record for that contact's parent Company. Instead of manually changing layouts and performing a find, a button can be added to the layout that performs the Go to Related Record script step or runs a script that includes that step with other actions. With a click of that button, the window changes layouts and displays the Company record assigned to the contact they were previously viewing.

In this example, Contact is called the *source table* and Company is considered the *target table*, each being on one side of the relationship between two records from the perspective of the starting layout's table. The same relationship used to *display* a related company name on a contact layout is the conduit through which the script step will *navigate*. By traveling "through" the relationship, the match fields used to form the relationship act like the search criteria for the record(s) that will be displayed in the target table.

Once the step is added to the script or assigned to a button, the Go to Related Record Options dialog, shown in Figure 35-4, is used to configure the action.

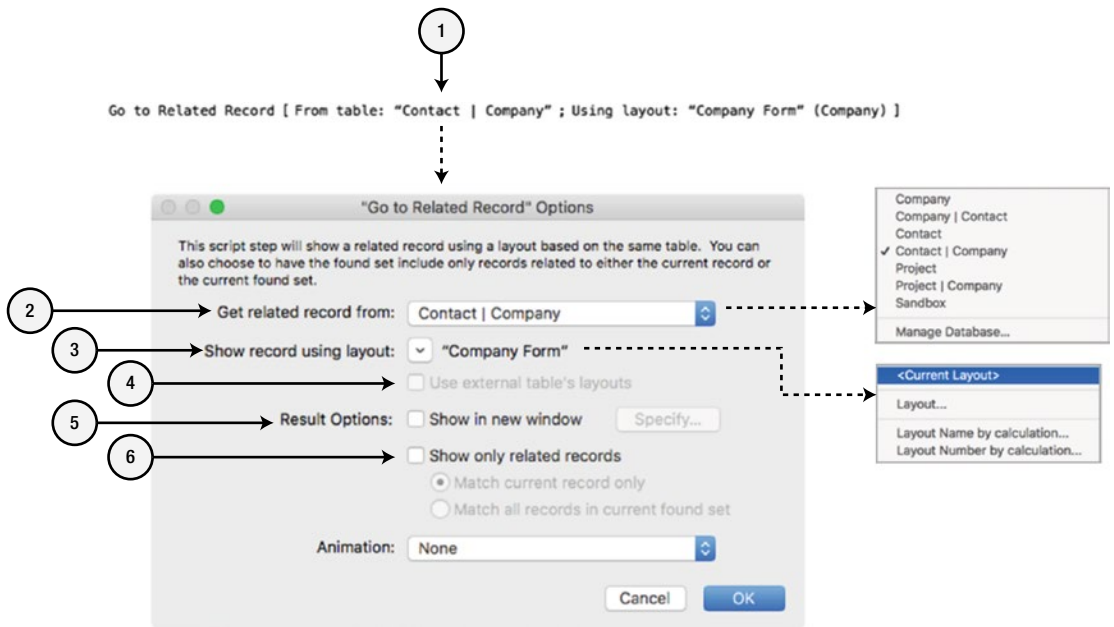


Figure 35-4. The *Go to Related Record* step

To configure the *Go to Related Record* step, follow these steps:

1. Double-click on the step or click on the gear icon to open the *Go to Related Record* Options dialog.
2. The *Get related record from* pop-up menu will list every relationship in the current database file. Select a relationship that connects the current layout's table to the target table that should be used as the conduit for the navigation find. To continue our earlier example in the *Learn FileMaker* database, choose the *Contact | Company* relationship to cause the step to travel from a *Contact* record to its related *Company*.
3. The *Show record using layout* button identifies which target table layout will be used to display the target record(s). It opens a menu with the following options:
 - *<Current Layout>* — leave the user on the starting layout (when the navigation process is using a self-join relationship).
 - *Layout* — opens a *Specify Layout* dialog in which you can pick the desired layout from a list of layouts that are automatically filtered based on the target table.
 - *Layout Name by calculation* — opens a *Specify Calculation* dialog in which a calculation can be entered that results in the *name* of the desired layout.
 - *Layout Number by calculation* — opens a *Specify Calculation* dialog in which a calculation can be entered that results in the *number* of the desired layout.

4. By default, the Specify Layout dialog in Step 3 will display a list any layouts for the target table from the current database file, even if the other side of the relationship is a table from an external database file. If the target table is from an external data source, the Use external table's layouts check box will be enabled. When enabled, the Specify Layout dialog will instead show a list of layouts for the target table from the external database file, and the script step will display the results in that file.
5. The Show in new window check box and adjacent Specify button allow the step to display the results in a new window instead of the starting window. These settings are identical to the New Window script step described later in this chapter.
6. The Show only related records option allows control over the context of records that will be displayed on the target layout. There are three options available for this setting:
 - When the check box is *disabled*, the related record from the target table will be displayed in one of two ways depending on whether it is contained in the current found set on the target layout. Essentially, the step attempts to preserve the current found set in the target layout if possible. If the target record *is* present in that found set, the record will be displayed without changing the found set. If the target record *is not* present in that found set, the record will be displayed in a found set of every record in the database.
 - When the checkbox is *enabled* and the Match current record only radio button is selected, the target record will be displayed in a new found set of only those records matching the criteria to establish a relationship to the *starting record*. This is the equivalent of a *one-to-many* search, where the current record is used to find matching records in the target table.
 - When the checkbox is *enabled* and the Match all records in the current found set radio button is selected, the target record will be displayed in a new found set of every record in the target table that has a match to *any* record in the found set in which the starting record resides in the starting layout. This is the equivalent of a *many-to-many* search, where the current found set of records is used to find matching records in the target table.

Interacting with Fields

There are several script steps that allow different interactions with fields, including:

- Go to Field
- Set Field
- Inserting Contents
- Select Field Contents
- Set Field's Next Serial Number

Go to Field

There are three script steps that allow navigation to a field on the current layout:

- Go to Next Field
- Go to Previous Field
- Go to Field

The first two of these have no parameter options. They will simply move active focus to the *next* or *previous* field on the current layout based on the field tab order. If the current record is not open and no field has focus, these steps will open the record and place focus respectively in the first or last field in the tab order.

The Go to Field step, shown in Figure 35-5, moves focus to a specific field.

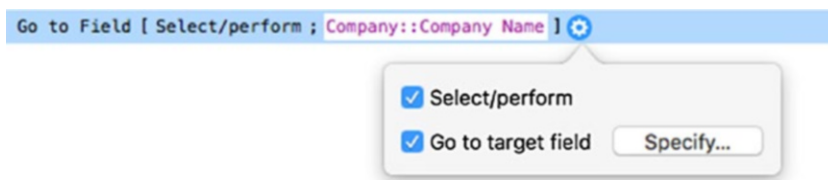


Figure 35-5. The Go to Field step

Click the gear icon to open the panel and configure the following options:

- **Select/perform** — enable this check box to select all field contents and/or to automatically open interface elements such as drop-down lists or calendars when entering the field.
- **Go to target field** — click the Specify button to open a Specify Field dialog to select the desired field that should receive focus.

■ **Caution** The selected field must be present on the current layout and the user must have access to enter it in the current mode. If no target field is selected or if that field is not present or not accessible, the step will exit every field and commit the record.

Set Field

The Set Field step, shown in Figure 35-6, will change the value of a specified field for the current record. The field being set does not need to be in focus or even present on the current layout. The value of any field local to the layout's table or in a related table accessible from the current layout's context can be set using this step.

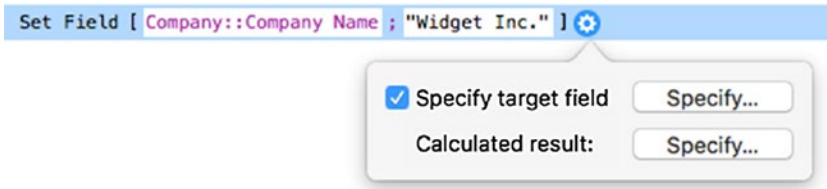


Figure 35-6. The *Set Field* step

After inserting the script step into a script or selecting it as a button’s action, click on the gear icon to open the panel and configure the following options:

- **Specify target field** — click the **Specify** button to open a **Specify Field** dialog to select the desired field that should receive the calculated result.
- **Calculated result** — click the **Specify** button to open a **Specify Calculation** dialog to enter a calculation that results in a value that will be placed into the field.

■ **Caution** If you don’t specify a target field and the record is open, the calculated result will be placed into the current field with focus.

Set Field by Name

The **Set Field by Name** step, shown in Figure 35-7, will change the value of a field specified *by name* for the current record displayed in the window. Unlike the **Set Field** step where a field reference must be hard-coded, this step provides a more flexible function by allowing the field to be specified by name in a calculation so it can vary from one instance to the next.

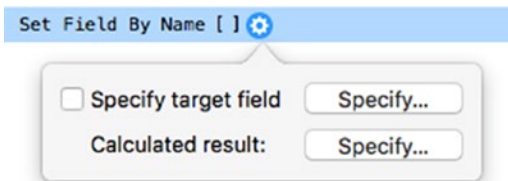


Figure 35-7. The *Set Field by Name* step

Click on the gear icon to open the panel and configure the following options:

- **Specify target field** — click the **Specify** button to open a **Specify Calculation** dialog to enter a formula that results in a text-based reference to the field whose value should be set. This must include the name of the table occurrence to provide the necessary context of locating the field.
- **Calculated result** — click the **Specify** button to open a **Specify Calculation** dialog to enter a calculation that results in a value that will be placed into the field.

For example, to set the **Contact Address State** field for a **Contact** record:

```
Set Field By Name [ "Contact::Contact Address State" ; "NY" ]
```

Since the field reference is text determined by a formula, it can include conditional variations or be constructed more dynamically. This example dynamically builds a reference to a standard Record Notes field for the table of the current layout, which would work in *any* table containing a field of that name:

```
Set Field By Name [ Get ( LayoutTableName ) & "::Record Notes" ; "Hello World" ]
```

Set Selection

The Set Selection step, shown in Figure 35-8, will select a portion of text within a specific field.

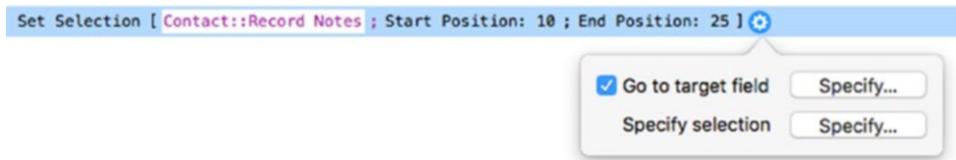


Figure 35-8. The Set Selection step

Click on the gear icon to open the panel and configure the following options:

- **Go to target field** — click the Specify button to open a Specify Field dialog to select which field will receive focus and have text selected. If the check box is left disabled, the content of the field with current focus will be selected.
- **Specify selection** — click the Specify button to open a Specify Set Selection dialog. This dialog has two text areas that can each receive a numeric value or a formula: Start Position and End Position.

When the step is executed, the selected field will become active and the specified characters of the text will be selected, as shown in Figure 35-9.

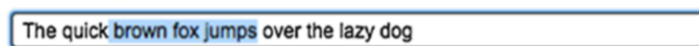


Figure 35-9. An example of a selection of characters 10 thru 25

■ **Tip** To select the *entire* contents of the current field, use the Select All step.

Set Next Serial Value

The Set Next Serial Value step, shown in Figure 35-10, will update a field's Auto Enter > Serial Number > Next Value. Since the auto-enter settings automatically increment the Next Value setting each time a new record is created, this step is used infrequently. However, when importing records as part of a data migration or when resetting a new version of a database for live use after extensive testing serial number fields may need an update to ensure that the next value assigned is uniquely one above the highest value assigned to date.

The change is made to the field definition directly, so the specified field does not have to be visible on the current layout. In many cases, it doesn't need to be part of the current layout's table.



Figure 35-10. The *Set Next Serial Value* step

Click on the gear icon to open the panel and configure the following options:

- **Specify target field** — click the **Specify** button to open a **Specify Field** dialog to select which serial number field should be updated.
- **Calculated result** — click the **Specify** button to open a **Specify Calculation** dialog and enter a formula that returns the next serial value that should be assigned to the field.

A migration import script might include the following function for each table in which data is imported or updated:

```
Go to Layout [ Contact List ]
Import Records [ ]
Show All Records
Unsort Records
Go to Record [ Last ]
Set Next Serial Value [ Record ID ; Contact::Record ID + 1 ]
```

The example shows the pattern for a script that:

- Navigates to the **Contact List** layout.
- Imports records (detail not shown).
- Sets the found set to show every record stored in the table.
- Unsorts those records into creation order.
- Navigates to the last record (with the highest serial number).
- Sets the next serial value of the **Contact::Record ID** field to that last record's serial number value plus one.

Under certain conditions, this pattern may require some changes.

For example, if the serial numbers aren't sequential when the records are sorted in creation order, the **Unsort Records** step might be replaced with a **Sort Records** step that sorts them by the **Record ID** field to ensure that the last record has the highest serial number.

Also, if the **Record ID** field is a text field with leading zeros making the values uniform in length, the formula to determine the next value would need to be changed. Adding one to a "number" as text with leading zeros will result in a numeric value without leading zeros, and the resulting value must be reformatted as text with the appropriate number of zeros to maintain the uniform format. For example,

if the serial number is six digits with leading zeros, the following formula can be used to determine the next value. In this example, the formula standardizes the number of leading zeros on a serial number regardless of its length by adding six zeros to the left of the value and then extracting six digits from the right, so that any starting value of one to six digits will still produce a result of six digits:

```
Right ( "000000" & ( Contact::Record ID + 1 ) ; 6 )
// result = 000015
```

Get Directory

The `Get Directory` step, shown in Figure 35-11, presents the user with a `Choose Folder` dialog prompting them to choose a folder location for use later in the script. This allows user-specified locations for importing, exporting, and saving various resources.

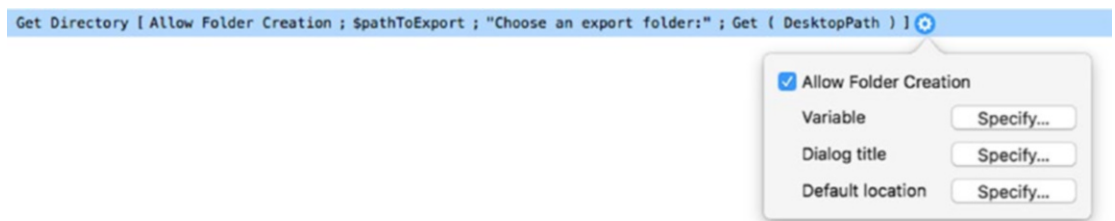


Figure 35-11. The `Get Directory` step

Click on the gear icon to open the panel and configure the following options:

- `Allow Folder Creation` — when enabled, this allows the user to create new folders from within the `Choose Folder` dialog.
- `Variable` — click the adjacent `Specify` button to open a `Set Variable Options` dialog to determine the name of the variable into which the selected folder path should be placed.
- `Dialog title` — click the adjacent `Specify` button to open a `Specify Calculation` dialog in which you can enter a formula for the text prompt instructing the user what to select in the `Choose Folder` dialog.
- `Default location` — click the adjacent `Specify` button to open a `Specify Calculation` dialog in which you can enter a formula that determines the folder location that will be displayed in the `Choose Folder` dialog by default.

The dialog shown in Figure 35-12 shows how the various settings affect the resulting dialog. Once the dialog is open, the user can navigate anywhere on their own computer directory and any network servers that they can access. Once they select a folder and click `Select`, the path to that folder will be placed into the variable specified.

■ **Caution** The exact appearance of the `Choose Folder` dialog will depend on the operating system and the user configuration choices made by the user. For example, the current version of macOS (Sierra) doesn't display the dialog title prompt, which may confuse users by not making clear why they are selecting a folder.

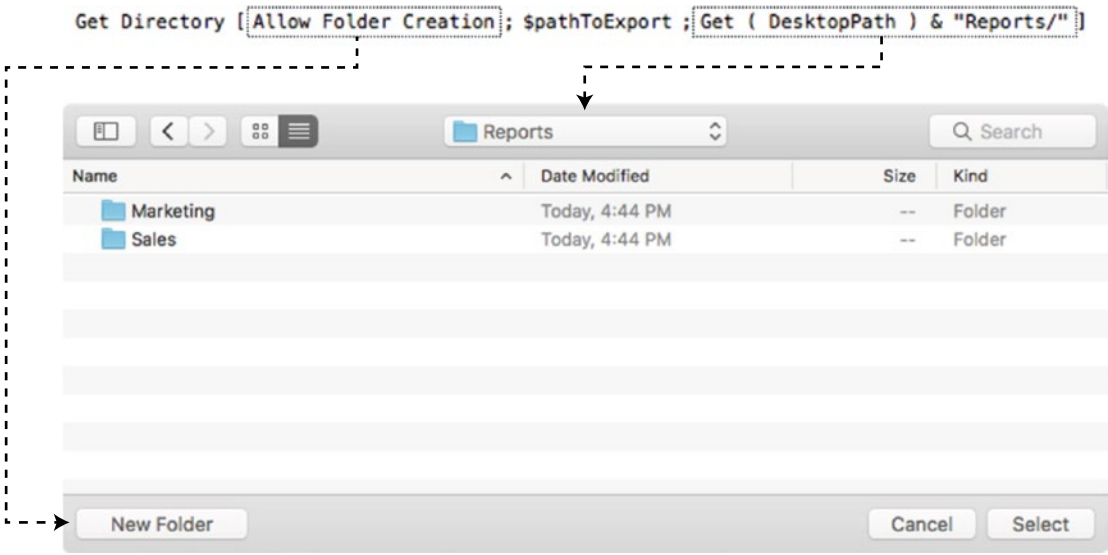


Figure 35-12. A hypothetical `Get Directory` configuration and the resulting dialog

Working with Records

There are many script steps that allow a variety of different interactions with records.

The `New Record/Request` step will create a new record in the current window while the `Duplicate Record/Request` step will create a new duplicate of the current record. These have no configurable options.

The `Delete Record/Request` step will automatically delete the current record in the front window after presenting an optional warning and confirmation dialog. Similarly, the `Delete All Records` step will delete every record in the found set after presenting an optional dialog.

There are steps available to perform many of the actions users can perform through the interface. Scripts can open, commit, copy, and revert records. There are steps for `Save Records as Excel`, `Save Records as PDF`, and `Save Records as Snapshot Link`.

The `Truncate Table` step deletes all records in a specified table, regardless of the current found set or the current layout context.

Also, the following two steps are used to move records in and out of a table:

- `Import Records`
- `Export Records`

Import Records

The `Import Record` step, shown in Figure 35-13, is used to import records into a table with or without human interaction, depending on the configuration settings.

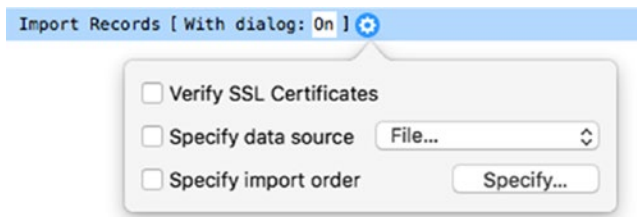


Figure 35-13. *The Import Records step*

Click on the gear icon to open the panel and configure the following options:

- **With dialog** — toggle this embedded parameter setting to instruct FileMaker to turn import dialogs *On* or *Off* when the step is performed. If a data source or import order is *not* defined in the other parameters, this setting will be ignored and the dialogs will appear, requesting the missing information.
- **Verify SSL Certificates** — enable this check box when importing XML data from a server specified with a HTTP request.
- **Specify data source** — specify the data source for the import. You can choose between File, Folder, XML Data, and ODBC Data. The subsequent options dialog will vary depending on your choice.
- **Specify import order** — opens an Import Field Mapping dialog in which to instruct the step about how data in the source should flow into the database.

■ **Note** To see examples of the Import and Export dialogs encountered when configuring script steps and a list of supported file types, see Chapter 5.

Export Records

The Export Record step, shown in Figure 35-14, allows a script to automatically export records with or without human interaction, depending on the configuration settings.

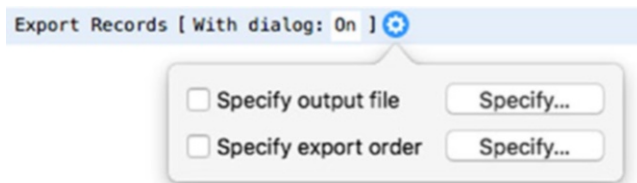


Figure 35-14. *The Export Records step*

After inserting the script step into a script or select it as a button's action, click on the gear icon to open the panel and configure the following options:

- **With dialog** — toggle this setting to instruct FileMaker to turn export dialogs On or Off when the step is performed. If the export output file and field order are *not* defined in the other parameters, this setting will be ignored and the dialogs will appear requesting the missing information.
- **Specify output file** — opens a Specify Output File dialog, choosing the file path into which the data should be saved.
- **Specify export order** — opens a Specify Field Order for Export dialog to instruct the step about how data in the source should flow into the database.

Using Show Custom Dialog

The Show Custom Dialog step is used to generate a dialog consistent with the user computer's operating system. A script may need to display a custom dialog for a variety of different reasons including:

- Presenting an informative notification or warning.
- Confirming a pending course of action with the option to cancel.
- Offering a choice of different courses of action.
- Requesting text input for find criteria or other functions.
- Presenting an instructional prompt for the user to enter input directly into a field through the dialog.

The gear icon opens a Show Custom Dialog Options dialog, with two tabs:

- **General** — Used to configure a standard dialog.
- **Input Fields** — Used to configure field input directly from the dialog.

Configuring a Dialog

The General tab of the Show Custom Dialog Options dialog, shown in Figure 35-15, is used to configure a custom dialog.

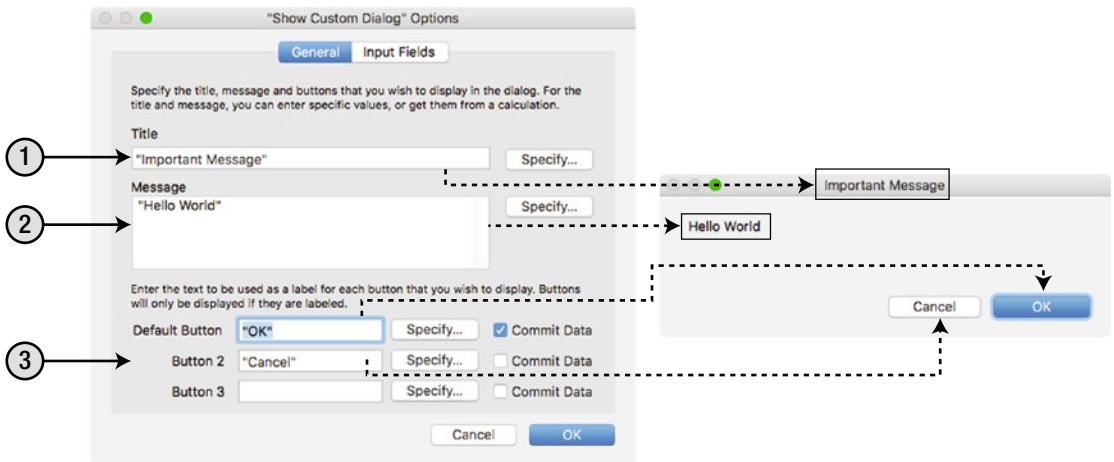


Figure 35-15. The General tab of the Show Custom Dialog Options dialog showing how settings manifest in the dialog being generated

To configure the Show Custom Dialog Options dialog, follow these steps:

1. Enter an optional Title for the dialog.
2. Enter a Message for the body of the dialog.
3. Enter up to three button names for the dialog. These will appear in the dialog from right to left with the Default Button being highlighted and responsive to the user typing the Enter key. If the current record has uncommitted changes at the time the dialog is presented, each enabled Commit Data check box will cause the adjacent button to commit the current record when a user clicks it.

Each of the above components can be entered by either:

- Typing directly into the appropriate text area.
- Clicking the adjacent Specify button to open a Specify Calculation dialog and then entering a formula that results in text.

■ **Caution** Custom dialogs do not automatically resize based on the amount of text entered, so a lengthy Message may be cut off. The user can resize the dialog to read additional information; however, since it is not visually apparent that more text is available or that the dialog can be resized, it is a good idea to try to keep dialog messages concise when using the Show Custom Dialog step.

Configuring a Dialog for Field Input

A custom dialog can include up to three editable fields, which will be connected to the current record at the time the script step is executed. When the dialog is first presented, any value in the fields will be displayed in the corresponding dialog fields. As the user types into the fields through the dialog, the values entered or edited will not be committed until they click a dialog button that is configured to commit. If they click a button not configured to commit, as one might set up a Cancel button, the text they type will be lost.

The Input Fields tab of the Show Custom Dialog Options dialog, shown in Figure 35-16, is used to add up to three editable fields directly into a dialog.

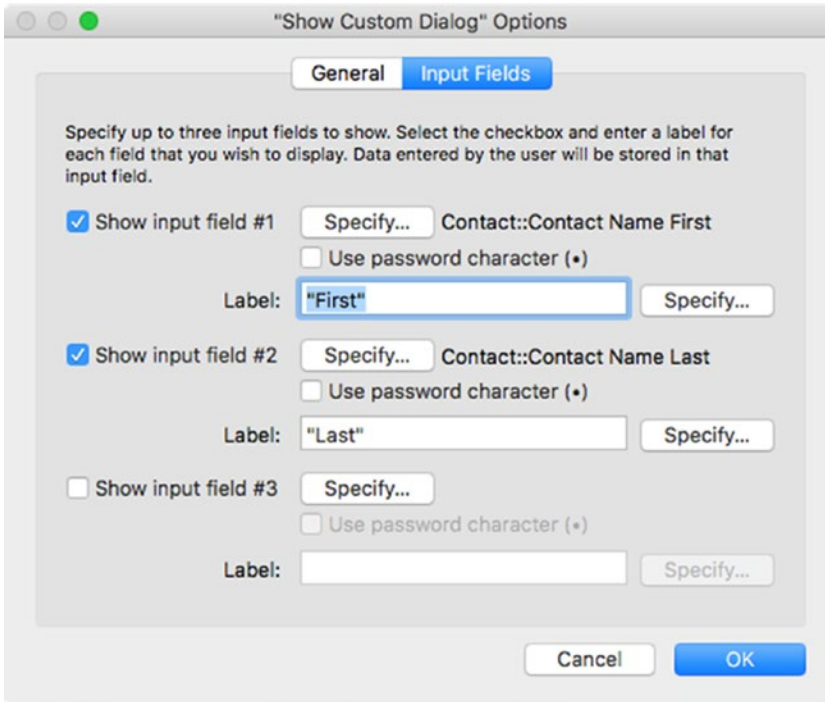


Figure 35-16. The Input Fields tab of the Show Custom Dialog Options dialog

The dialog repeats the controls for up to three fields. Each section, like the one shown in Figure 35-17, can be activated and configured to include one field.

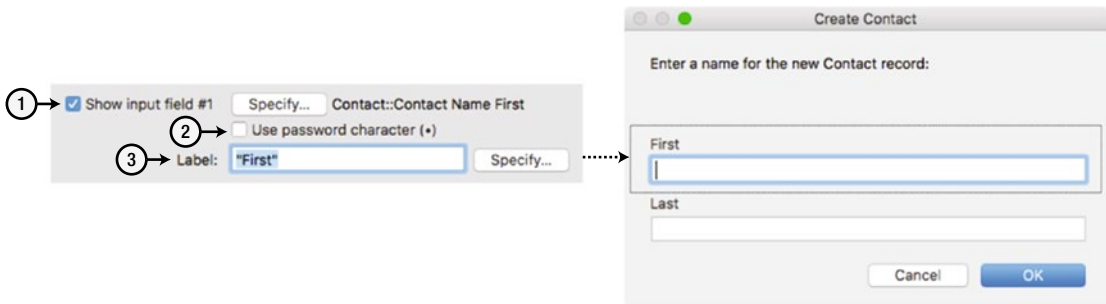


Figure 35-17. One set of field controls showing how field settings appear in a dialog

To configure each field, follow these steps:

1. Check the `Show input` field box. This will open a `Specify Field` dialog allowing the selection of a desired field. To reassign the field later, click the adjacent `Specify` button to reopen the dialog.
2. If the field is used to enter a password or other confidential information, the `Use password character check` box can be enabled to cause the data entered to appear as bullets in the dialog. This is only a display of data and does not alter or encrypt the text entered into the field, which will still be visible on layouts unless it is also configured to display as bullets.
3. By default, the field will be present on the dialog without a label. To add a label, type into the `Label` text area or click the adjacent `Specify` button to enter a formula that results in the text for the label.

■ **Tip** When a dialog requires more than three button choices or needs to contain more than text, use the `New Window` script step to generate a fully custom dialog as a stand-alone window or attached to the current window using the new `Card` option.

Capturing the Button Click

Once a dialog is dismissed by the user, the script can access the index of the button the user clicked using the `Get (LastMessageChoice)` function. This value can be stored in a variable and used later in a script to route the workflow to the appropriate course of action. This example shows a `Show Custom Dialog` step followed by a `Set Variable` step that places the clicked button index into a variable named `$function`:

```
Show Custom Dialog [ "Choose" ; "Please choose an function" ]
Set Variable [ $function ; Get ( LastMessageChoice )
```

In the above example, if there are three buttons — `OK`, `Cancel`, and `Help` — and the user clicks `Cancel`, the `$function` variable will be set to a value of 2.

Convert the button index back into a name with the `Set Variable` function using a `Let` and `Case` statement, as shown in the following formula:

```
Let (
    number = Get ( LastMessageChoice )
;
    Case (
        number = 1 ; "OK" ;
        number = 2 ; "Cancel" ;
        "Help" )
)
```

Conditionally Branching with If Statements

An `If` statement is a set of control steps used to enclose other steps that will only be performed if some conditional criteria are met. There are four different steps available to construct an `If` statement:

- **If** — begins a new conditional statement and set of conditional steps. Contains a formula returning a Boolean that determines if the steps will be performed.
- **Else If** — begins a new set of conditional steps within a conditional statement. Contains a formula returning a Boolean that determines if the subsequent steps will be performed.
- **Else** — begins a set of conditional steps that are performed as a last resort if all preceding conditions were false.
- **End If** — terminates the last set of conditional steps and closes out the overall **If** statement.

Building Simple If Statements

The most basic type of **If** statement includes the establishing **If** step and terminating **End If** step surrounding one or more conditional steps, as shown in this pattern formula:

```
If [ <criteria> ]
    <conditional steps>
End If
```

For example, this statement will display the specified dialog if the current user account is assigned a privilege set that allows full access to the database:

```
If [ Get ( AccountPrivilegeSetName ) = "[Full Access]" ]
    Show Custom Dialog [ "Hello Developer" ]
End If
```

In the above example, if the user does not have the access specified, no action is taken and the script continues performing other steps not contained within the statement (not shown). However, to add an alternative set of steps that are executed if the initial criteria returns a false value, use an **Else** step, as shown in this pattern formula:

```
If [ <criteria> ]
    <conditional steps 1>
Else
    <conditional steps 2>
End If
```

In the following example, a dialog will be displayed with a different message depending on the user's privilege set. If the user has full access, they receive a dialog identifying and greeting them as a developer. If the user has *any* other access level, they receive the greeting in the second dialog step.

```
If [ Get ( AccountPrivilegeSetName ) = "[Full Access]" ]
    Show Custom Dialog [ "Hello Developer" ]
Else
    Show Custom Dialog [ "Hello Regular User" ]
End If
```

Building Compound If Statement

Unlike a simple If statement that contains one set of criteria and two possible outcomes (true or false), a compound statement uses the Else If step to create any number of possible conditions as demonstrated in this pattern formula:

```
If [ <criteria 1> ]
    <conditional steps 1>
Else If [ <criteria 2> ]
    <conditional steps 2>
Else If [ <criteria 3> ]
    <conditional steps 3>
Else
    <conditional steps 4>
End If
```

The statement begins with an If step that contains the first combination of criteria and conditional steps. Each subsequent set of criteria steps is initiated by an Else If step, until finally the statement is terminated with an End If after an optional Else that allows for a final set of steps conditionally run if all the above are false.

The script will work from the top down, evaluating each criteria until one results in a true value. Once it encounters a true result, it will run the enclosed conditional steps and then skip all other criteria and conditions.

In this example, a different hypothetical email script is performed depending on the Status field of an Invoice record:

```
If [ Invoice::Status = "Paid" ]
    Perform Script ["Email Invoice Paid" ]
Else If [ Invoice::Status = "Past Due" ]
    Perform Script ["Email Invoice Past Due" ]
Else If [ Invoice::Status = "Due" ]
    Perform Script ["Email Invoice Due" ]
End If
```

Building Nested If Statement

Multiple sets of If statements can be nested hierarchically so that each top-level criteria triggers a compound If statement, as shown in this formula:

```
If [ <Level 1 Criteria 1> ]
    If [ <Level 2 criteria 1> ]
        <conditional steps 1>
    Else If [ <Level 2 criteria 2> ]
        <conditional steps 2>
    End If
Else If [ <Level 1 Criteria 2> ]
    If [ <Level 2 criteria 3> ]
        <conditional steps 3>
    Else If [ <Level 2 criteria 4> ]
        <conditional steps 4>
    End If
End If
```

Performing Searches

There are several different steps that perform searches. These can be used separately or combined with steps that perform navigation, sorts, compilations, and batch processes.

The following two steps are used to perform searches:

- **Perform Quick Find** — This step is configured with a formula whose result will be used to search within any field on the current layout that has **Quick Find** enabled (Chapter 4, “Searching with Quick Find”).
- **Perform Find** — This step is configured with a **Specify Find Request** dialog in which complex find criteria can be entered (Chapter 4, “Managing Saved Finds”).

The **Enter Find Mode** step, shown in Figure 35-18, is used to initiate a find request, optionally providing some starting criteria.

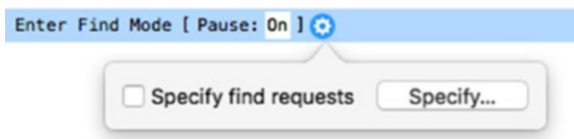


Figure 35-18. The *Enter Find Mode* step

The **Enter Find Mode** step will switch the current window into Find mode and has the following two configuration options:

- **Pause** — When this setting is **On**, the script will enter Find mode and pause to allow a user to enter find criteria or perform other actions. The user will have to press the **Enter** key or click the **Continue** or **Cancel** button in the second level of the status toolbar to continue the script.
- **Specify find requests** — This setting opens a **Specify Find Request** dialog in which complex find criteria can be entered that will be automatically set up after entering Find mode.

Since the step does not perform a find itself, it allows for situations where the user or other script steps add additional criteria. Subsequent **Set Field** steps can be added to set up additional, dynamically calculated criteria beyond the entries made by the user or the **Specify find requests** setting.

A **Perform Find** step must be added at some point to perform the find.

A find script can take many forms. It may perform a find based on static unchanging criteria, build the find based on variables, solicit user criteria, or some combination of these. This example script uses one hard-coded criterion and one user entered. It sets up an invoice search for invoices with a **Status** of “Paid” and then presents a dialog allowing the user to enter a date range into an **Invoice Date** field. Once the user enters a date range and presses the **Enter** key, the resulting search will find all paid invoices that have a date that falls within the range entered by the user. The script includes a conditional dialog that appears if no records are found.

```

Go to Layout [ "Invoice List" ]
Enter Find Mode [ Pause: Off ]
Set Field [ Invoice::Invoice Status ; "Paid" ]
Show Custom Dialog [
    "Search Paid Invoices" ;
    "Enter a date range (m/d/yyyy...m/d/yyyy):" ;
    Invoice::Invoice Date ]
Perform Find
If [ Get ( FoundCount ) = 0 ]
    Show Custom Dialog [
        "No Records Found" ;
        "No paid invoices were found within the date range you entered." ]
End If

```

Working with Found Sets

There are many steps that manipulate found sets of records. Most have a configuration dialog and functionality that are identical to the same functions in the menu bar.

- **Constrain Found Set** — used in lieu of the Perform Find script step to *narrow* the current found set of records based on criteria entered in a Specify Find Requests dialog.
- **Extended Found Set** — used in lieu of the Perform Find script step to *broaden* the current found set of records based on criteria entered in a Specify Find Requests dialog.
- **Find Matching Records** — provides an optional Specify Field dialog and will search for records with a matching value in either the specified field or the field with current focus.
- **Modify Last Find** — enters Find mode with the last criteria in place.
- **Omit Record** — omits the current record from the found set.
- **Omit Multiple Records** — omits one or more records from the found set starting with the current record based on user input in an optional dialog or a number either hard-coded or determined by a calculation formula in an Omit Multiple Records Options dialog on the step.
- **Show All Records** — replaces the found set with all records in the table for the current window while remaining on the current record.
- **Show Omitted Only** — replaces the found set with all records currently omitted.
- **Sort Records** — sorts the records based on user input or sort criteria hard-coded into an optional Sort Records.
- **Sort Records by Field** — provides the ability to quickly sort based on a single field: either the field with current focus or a field selected in the optional Specify Field dialog. There is a choice of sorting methods including Ascending, Descending, and by Associated Value List.
- **Unsort Records** — sorts the found set of records by their creation order.

The Loop step, discussed in the next section, can traverse a found set and compile data or preform actions on records one by one.

Iterating with a Loop Statement

A Loop statement is a set of control steps used to enclose other steps that are repeatedly executed until a certain condition is met. There are three different steps available to construct a Loop statement:

- **Loop** — begins a new conditional statement that will repeat the subsequent steps placed between it and an **End Loop** step.
- **Exit Loop If** — contains a calculation formula that will exit the loop if the result is true. When exiting the loop, the script will skip any steps from the position of this step down to the next **End Loop** step before continuing to execute any remaining steps below it.
- **End Loop** — delimits the bottom of the Loop statement, but does *not* exit the loop. When this is reached during execution, the script will return to the preceding Loop step and repeat the interim steps until a step is executed that explicitly exits.

There are several other script steps that will cause a script to exit a Loop statement. These include **Exit Script**, **Halt Script**, and using the **Exit after last** option of the **Go to Record/Request/Page > Next** and **Go to Portal Row > Next** steps.

Looping Through a Found Set of Records

A frequent use of a Loop statement is to have a script work through a found set, performing one or more tasks for each record. The structure of such a statement begins by navigating to the first record in a found set and then starting a loop. After any task steps are performed, the script moves to the next record until it reaches the last record in the found set.

This example shows the pattern:

```
Go to Record/Request/Page [ First ]
Loop
    <insert task steps>
    Go to Record/Request/Page [ Next ; Exit after last: On ]
End Loop
```

■ **Caution** If the **Exit after last** parameter is turned **Off** and no other steps exit the loop, the script will be caught in an endless loop. The user can only escape an endless loop by typing **Esc** or a **Period** while holding the **Command** key (macOS) key *if* the **Allow User Abort** setting is **On**; otherwise, the application must be force quit.

Building a List of Record Information with a Loop Statement

The task steps you use in a Loop statement will vary from one script to the next and can perform any task imaginable. For example, after a script finds and sorts a list of open invoices, it might include the following code to compile a text summary of those records in a **\$Body** variable, as shown in this example:

```
Go to Record/Request/Page [ First ]
Loop
    # Add body row
    Set Variable [ $Body ; $Body &
        Invoice::Record ID & " " &
        Invoice::Invoice Date & " " &
```

```

Invoice::Invoice Company Name & " " &
Invoice::Invoice Amount & " " &
Invoice::Invoice Status & "¶" ]
Go to Record/Request/Page [ Next ; Exit after last: On ]
End Loop

```

For each record in the found set, the Set Variable step adds a new paragraph to the \$Body variable containing the values stored in five fields. After this script finishes, the \$Body variable will contain a text list of open invoices that might look something like this:

```

1001 04/15/2017 Fantastic Client $1,000 Over 60 Days
1001 04/15/2017 Fantastic Client $1,000 Over 60 Days
1002 05/19/2017 Creative Company $3,000 Over 30 Days
1003 06/1/2017 Fantastic Client $4,500 Open
1003 06/15/2017 Fantastic Client $958 Open

```

Grouping a List of Record Information with a Loop Statement

With a more sophisticated Loop statement, the previous script example can create a *summarized* list of a found set of sorted records. For example, if the found set of open invoices are sorted by Invoice Status, the script can be modified to return a list of invoices organized into groups based on their status, as shown here:

```

Open
1003 06/1/2017 Fantastic Client $4,500
1003 06/15/2017 Fantastic Client $958

Over 30 Days
1002 05/19/2017 Creative Company $3,000

Over 60 Days
1001 04/15/2017 Fantastic Client $1,000
1001 04/15/2017 Fantastic Client $1,000

```

To achieve this result, the following modifications are needed to detect and insert a heading every time a status change is detected and then store that status to avoid heading duplication:

- A variable named \$Last.Status will be initialized at the start with an empty string and then used to store the last status heading that has been encountered and added into the results.
- At the beginning of the Loop statement, the Invoice Status field value will be placed into a variable named \$Status to shorten it for easier use repeatedly in a new conditional statement.
- An If statement will be added to handle the insertion of the status heading into the result and tracking this fact. A new status is detected when \$Status is not equal to \$Last.Status.

- Within the If statement are two Set Variable steps:
 - The \$Body variable is modified to include a Case statement that conditionally adds a carriage return if the body is not currently empty and then the new \$Status value is added followed by another carriage return.
 - The \$Last.Status variable is modified to contain the new \$Status so that another heading isn't added until a different status value is encountered.

Taking all this into consideration, the new script should appear as follows:

```

Set Variable [ $Last.Status ; "" ]
Go to Record/Request/Page [ First ]
Loop
  Set Variable [ $Status ; Invoice::Invoice Status ]
  If [ $Status ≠ $Last.Status ]
    # Add header row
    Set Variable [ $Body ; $ Body &
      Case ( $Body ≠ "" ; "¶" ) & $Status & "¶" ]
    Set Variable [ $Last.Status ; $Status ]
  End If
  # Add body row
  Set Variable [ $Body ; $ Body &
    Invoice::Record ID & " " &
    Invoice::Invoice Date & " " &
    Invoice::Invoice Company Name & " " &
    Invoice::Invoice Amount & " " &
    Invoice::Invoice Status & "¶" ]
  Go to Record/Request/Page [ Next ; Exit after last: On ]
End Loop

```

Looping Through Lists of Data

A Loop statement can be used to work through a carriage return-delimited list of values one by one. This involves setting up a couple of control variables to store the *count* of values and keep track of which among them is the *current* value. For example, assume a script contains a list of several state abbreviations, placed them into a variable named \$States and needs to use them as the criteria for a multi-value search of contacts residing in one of the included states. If the entire list of states is entered into a single find request, there will be no resulting matches since that would instruct FileMaker to look for a record with all the states in the field. To get a result, multiple find requests need to be created, one for each state in the list.

The following script enters Find mode and then loops through the list of states, creating one find request for each state listed. Once they are all set up, the find is performed.

```

Set Variable [ $Count ; ValueCount ( $ States ) ]
Set Variable [ $Current ; 1 ]
Enter Find Mode [ Pause : Off ]
Loop
  Set Field [ Contact::Contact Address State ; GetValue ( $States ; $Current ) ]
  Set Variable [ $Current ; $Current + 1 ]
  Exit Loop If [ $Current > $Count ]
  New Record/Request
End Loop
Perform Find [ ]

```

By dynamically counting the number of state values in the list and exiting when the next `$Current` value exceeds that count, the script will work for *any* number of states listed in the variable. It continues to repeat on to the next value until it reaches the end of the list and the `Exit Loop If` condition is true.

Confirming a Dialog Field Value

Another use of a `Loop` statement is to confirm that a user has entered a value into a field in a dialog and circling back with a warning prompt if they fail to do so. For example, a script that searches invoices based on status, would require that the user enter a value into the `Invoice Status` field. To avoid errors or confused results, a `Loop` statement can postpone performing the find until the user complies with that requirement.

The script will begin as shown below by entering `Find` mode and asking the user to enter an `Invoice Status` in a dialog, as shown in Figure 35-19. If they click the `Cancel` button, the script returns to `Browse` mode and exits the script. The following code will perform these steps:

```
Enter Find Mode [ Pause: Off ]
Show Custom Dialog [ "Search" ; "Enter a status:" ; Invoice::Invoice Status ]
If [ Get ( LastMessageChoice ) = 2 ]
    Enter Browse Mode
    Exit Script
End if
```

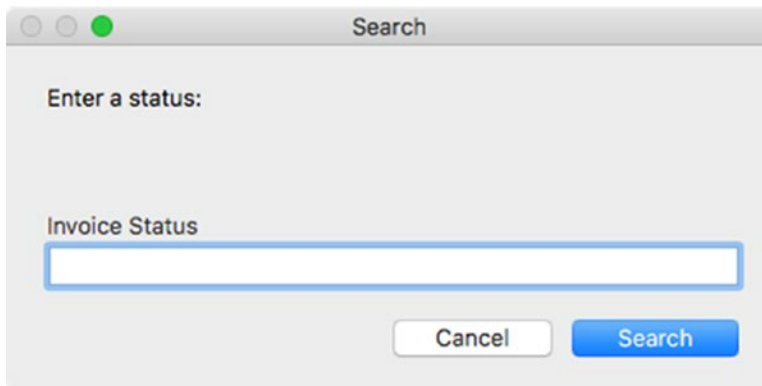


Figure 35-19. A dialog prompting the user to enter an invoice status that will be used to perform a search

If the script continues executing beyond this point, we know that the user has clicked the `Search` button. In the next section, a `Loop` statement is used to confirm that a value was typed into the field. This is accomplished by performing the following functions:

- The first step within the `Loop` statement uses an `Exit Loop If` step to immediately exit the statement if the user followed the instructions and did, in fact, enter a value. If they did not enter a value, the expression returns a false value and the script stays within the statement.
- The `Bleep` step causes the computer to sound an audio alert.
- A dialog identical to the original but with a slightly enhanced message is opened, explicitly stating that a status is required.

- When the user clicks a button on the dialog, one of three actions occur:
 - If they click the Cancel button, the script will return to Browse mode and exit.
 - If they click the Search button *without* enter a value, the Loop statement repeats, presenting another dialog.
 - If they click the Search button *with* a value entered, the Loop statement starts to repeat but exits with the Exit Loop If step immediately at the top.

The preceding steps are expressed in the following script steps:

```

Loop
  Exit Loop If [ Invoice::Invoice Status ≠ "" ]
  Beep
  Show Custom Dialog [ "Search" ; "A status is required:" ; Invoice::Invoice Status ]
  If [ Get ( LastMessageChoice ) = 2 ]
    Enter Browse Mode
    Exit Script
  End if
End Loop

```

Once the loop is exited, the script can continue, performing the find and taking whatever additional steps are necessary to complete its task.

Enhancing the Confirmation Process

The formula in the Exit Loop If step could be enhanced to validate the text enter, confirming that it is the appropriate data type or is within a certain range of values. In this example, the loop will only exit if the value entered is one of seven hard-coded options:

```

Exit Loop If [
  Invoice::Invoice Status ≠ "Paid" and
  Invoice::Invoice Status ≠ "Open" and
  Invoice::Invoice Status ≠ "Past Due" and
  Invoice::Invoice Status ≠ "Over 30 Days" and
  Invoice::Invoice Status ≠ "Over 60 Days" and
  Invoice::Invoice Status ≠ "Over 90 Days" and
  Invoice::Invoice Status ≠ "Over 120 Days"
]

```

To avoid hard-coded values, the formula could instead get a list of acceptable values from the same value list associated with the status field being searched. This example assumes the existence of a value list named Invoice Status that contains all the available options for the field and uses that to confirm that the value entered in the dialog is appropriate:

```

Let (
  Values = ValueListItems ( Get ( FileName ) ; "Invoice Status" ) ;
  Not IsEmpty ( FilterValues ( Values ; Invoice::Invoice Status ) )
)

```

■ **Tip** An even more sophisticated option is to generate a *fully* custom dialog using the New Window step described later in this chapter. This would allow the inclusion of a pop-up menu of choices to relieve users of the need to remember and type valid values and instead simply select a value from the list.

Scripting Portal Functions

Previously, we discussed how to create and configure portals on a layout and mentioned some examples of basic functionality for including buttons attached to hypothetical scripts (Chapter 30). In this section, we explore some examples of scripts used to enhance a portal by providing basic navigation and record management functionality.

Navigating to a Portal Row's Record

Portals are like miniature list layouts, where every row displayed is a related record that exists in the table serving as the portal's data source. To be used as a navigation conduit, they require a button connected to a script step or a script that will navigate the user to the related record on the appropriate layout.

The Go to Related Record script step can be attached *directly* to a portal row button to serve this purpose. However, that function will be limited to the options: isolating a set of records in the target table, switching layouts, and even opening a new window with specified properties. That may be sufficient in most cases, but consider the following limitations:

- There is no integrated option to sort records so that the records displayed in the destination layout sort in the same order as they were displayed in the portal. This can be disorienting to users, especially when they move to view the next or previous record.
- The step allows opening a *new* window with specified dimensions and a new name. However, if you prefer to maintain a single window environment, there is no option to *change* the size, position, or name of the current window after the layout change.

There is a way to address these limitations and run additional functions after switching layouts. You *can* set up an OnRecordLoad script trigger on the destination layout to run a script that performs additional steps like sorting and resizing or renaming the window. However, those functions will run *every* time a user encounters a record there and not just the first instance of navigating from a portal. Also, this creates a fragmented technical design that can become confusing in a complex database.

Therefore, the better option is usually to have the portal navigation button perform a script that contains *all* the steps required, including going to the related record(s), window setup, sorting, and any other necessary tasks.

The following example demonstrates the basic setup of such a script and navigates from a Company record to a Contact record through a portal:

```
Go to Related Record [
    From table: "Company | Contact" ;
    Using layout: "Contact Form" (Contact)
]
Sort Records [ Restore ; With dialog: Off ]
<Insert Other Steps>
```

Remember that a script attached to a button in a portal row will operate based the context of the *row* the user clicks on. Therefore, be sure to perform context-sensitive steps within the script *before* performing steps like committing a record or changing layouts.

When running a script by clicking a button on a Contact portal row in a portal on a Company layout, any references to any field in the related Contact table will be pulled from the row clicked until an action is taken that commits the current record. Once the record is committed, any references to related fields will be pulled from the first row. Similarly, once the script has performed the Go to Related Record step, the context will change to the destination table occurrence so the fields selected in the Sort Records step must be from that context and not the original context.

Adding a Portal Row

When allowing direct creation of related records (Chapter 30, “Creating Records in a Portal”) isn’t desirable or adequate, a button can be added above or below the portal and attached to a script that performs the following functions:

1. Initialize variables with the values of all the match fields used to establish the relationship between the current layout’s table (starting) to the table acting as the portal’s data source (target).
2. Freeze the window to avoid user distraction and performance drag (Freeze Window command is described in the next section).
3. Switch to a layout that displays records from the target table.
4. Create a new record in the target table.
5. Enter the match field values from the starting record into the match fields from the target table to establish a relational connection between the new record in the target table and the parent record in the starting table.
6. Commit the new record.
7. Switch back to the original layout.

This example demonstrates a script that will create a new Contact record attached to the current Company record:

```
Set Variable [ $Company.ID ; Company::Record ID ]
Freeze Window
Go To Layout [ "Contact - Form" (Contact) ; Animation: None ]
New Record/Request
Set Field [ Contact::Record Parent ID ; $Company.ID ]
Commit Records/Requests [ With dialog: Off ]
Go to Layout [ original layout ; Animation: None ]
```

When this script runs, from the user’s perspective, nothing will have changed except for the appearance of a new row in the portal.

Managing Windows

There are numerous script steps that allow for the manipulation and management of windows, including:

- **Adjust Window** — adjust the visibility or size of a window based on the following options:
 - **Resize to Fit** — adjust the dimensions of the window to the minimum size possible while continuing to display the entire content area.
 - **Maximize** — resize the window to the full size of the computer screen on which is it currently displayed.
 - **Minimize** — shrink the window down to an icon stored in the Dock (macOS) or taskbar (Windows).
 - **Restore** — returns the window to the size and position as it was at the beginning of the script.
 - **Hide** — hide the window from view, placing it into the Window > Show Window menu from where it can be reopened.
- **Arrange all Windows** — arranges all the open windows by the selected method:
 - **Tile Horizontally**
 - **Tile Vertically**
 - **Cascade**
 - **Bring All To Front**
- **Select Window** — bring a specified window to the front of every other window.
- **Move / Resize Window** — adjust the Height, Width, Distance from Top, and Distance from Left for a specified window.
- **Close Window** — close a specified window.
- **Freeze Window** — lock the active window from refreshing during the remainder of a script. This helps to speed up a scripted process and hides layout and record changes from being a distraction to the user.
- **Refresh Window** — update the active window's record content.
- **Scroll Window** — scrolls the window view by the selected method:
 - **Home**
 - **End**
 - **Page Up**
 - **Page Down**
 - **To Selection**
- **Set Window Title** — change the title of a specified window.

Creating a New Window

The New Window step, shown in Figure 35-20 will create a new window based on the current window's type, layout, size, positions, and other properties unless these are specifically modified in the options dialog.

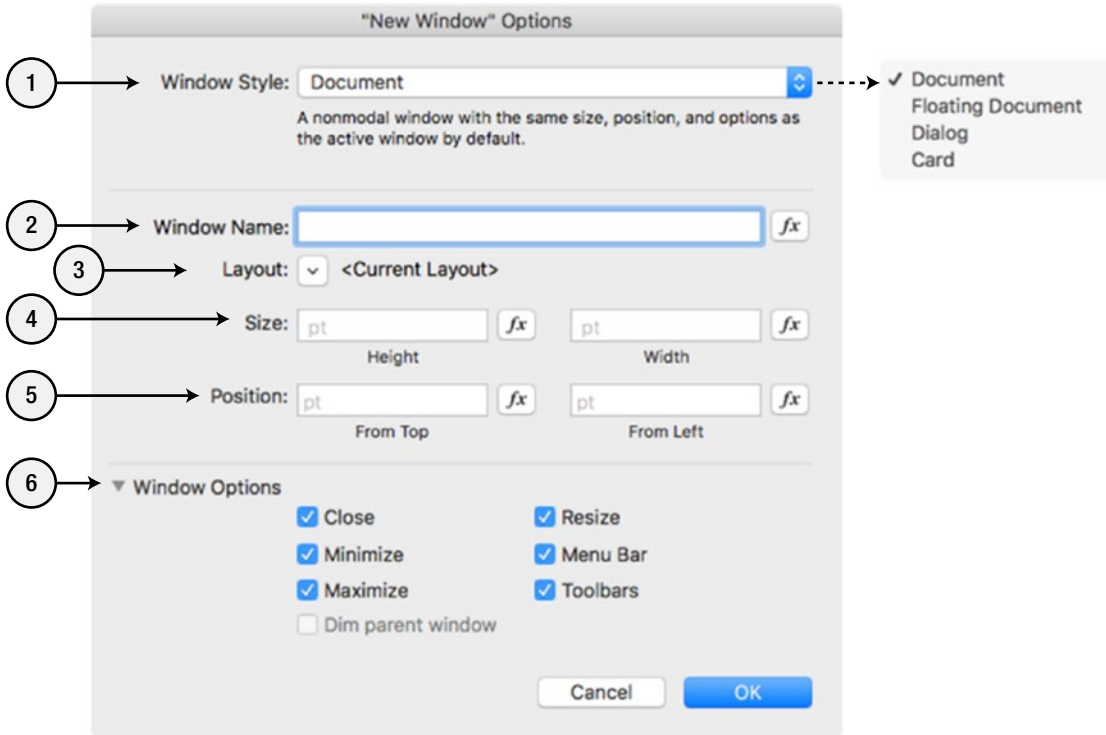


Figure 35-20. The New Window step

To configure the New Windows Options dialog, follow these steps:

1. Select a Window Style from the following choices:
 - Document — A standard non-modal FileMaker window.
 - Floating Document — A non-modal window that will remain in front of other FileMaker windows.
 - Dialog — A window that is modal to the application.
 - Card — A window that is modal to the parent window.

■ **Note** A window is considered *modal* when it forces the user to interact with and/or dismiss it before they can resume work in the application or window.

2. Optionally, enter a `Window Name` into the text area or click the `fx` button to enter a formula to dynamically calculate a name. If no name is specified, the new window will be named the same as the current window with a numeric suffix.
3. Optionally, select a `Layout` that should be displayed in the new window.
4. Optionally, enter a `Height` and `Width` to control the `Size` of the new window by typing directly into the text areas or clicking the `fx` button to enter a formula to dynamically calculate them.
5. Optionally, enter a `From Top` and `From Left` value to control the `Position` of the new window by typing directly into the text areas or clicking the `fx` button to enter a formula to dynamically calculate them.

■ **Tip** Developers often create “staging” windows that have a large negative value for the `From Left` and `From Top` properties to keep them hidden offscreen. These are ideal when a script performs several preparatory steps before presenting a window to the user or when performing a complex sequence of automated steps that would be a distraction to a user if in full view.

6. Select from the list of available `Window Options` to control the following properties of the new window:
 - `Close` — enables a standard close window control.
 - `Minimize` — enables a standard minimize window control.
 - `Maximize` — enables a standard maximize window control.
 - `Resize` — allows the user to manually resize the window after it is created.
 - `Menubars` — controls if the menubar is visible at the top of the window (Windows only).
 - `Toolbars` — controls if the status toolbar is visible.
 - `Dim Parent Window` — Dims the parent window (Card only).

■ **Caution** If you present a modal window without access to the close function and without providing a button that closes the window, the user will be stuck and must force quit the application to continue.

Creating a Delete Confirmation Dialog

The `New Window` step can create a fully custom dialog window based on a layout specifically designed to serve that purpose, with a more robust interface than is possible using the built-in `Show Custom Dialog` step. A custom dialog can exceed the limit of three buttons and input fields, and can include any other interface elements such as pop-up menus, radio buttons, check boxes, calendars, charts, etc.

To illustrate this with a simple example, create a custom deletion script that presents a dialog that includes the name of the table for which the record belongs and requires the user to type the word “Delete” into a text box before it will allow the record to be deleted. This script could be attached to buttons or as a replacement function for the `Delete Record` menu if using custom menus.

Creating a Table for the Dialog’s Resources

Begin by creating a new table with fields for the dialog. Since this dialog will be used when deleting any record in any table, it makes sense to have a dedicated table rather than add the required fields into an existing table. So, create a new table named `Dialog` and then add the following global text fields without any options:

- `Dialog Body` — This field will be used to store text that will be displayed as the *body* of the dialog, allowing it to change depending on the type of record being deleted.
- `Dialog Confirmation Entry` — This field will be used as a confirmation that the user is paying attention to the task they are about to undertake to avoid mistakes. They must type the word “Delete” into it before the `Delete` button will become enabled.
- `Dialog Confirmation Prompt` — This field will be used to inform the user to enter the word “Delete” into the confirmation field.
- `Dialog Button 1` — This field will be used to store the name of the first dialog button.
- `Dialog Button 2` — This field will be used to store the name of the second dialog button.

■ **Tip** Notice that we have named these fields generically and included two fields that the script will use to specify the name of the button. This will allow the table and fields to be expanded in the future and be used to generate any number of additional dialog types.

Creating a Layout for the Dialog

Now create a new form layout called `Dialog`, assigned to the newly created `Dialog` table. The layout should have all the fields assigned to the table, arranged and formatted like they are shown in Figure 35-21.

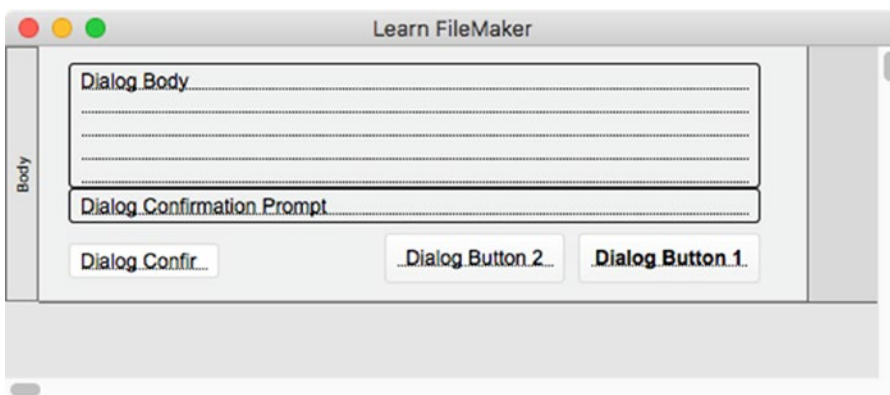


Figure 35-21. An example of a *Dialog* layout

Since this dialog will pull field values as the button names and since a field object can be converted into a button, either of the following two options can create the buttons:

- Create buttons and use a merge field for the name property: <<fieldname>>.
- Create the fields on the layout along with a temporary button. Then use the Format Painter tool to copy and paste the style settings from the actual button to the text fields. The resulting fields will appear to be buttons, as shown.

Enter Browse mode and create a new empty record in the Dialog table.

Creating a Script for the Dialog

Now create a new script named `Delete Record {function}`. This script will have a parameter so that it can handle new dialog requests as well as cancellation requests from the dialog itself. See Figure 35-22.

```

1 # Declarations
2 Set Variable [ $function ; Value: Get ( ScriptParameter ) ]
3 Allow User Abort [ Off ]
4 Freeze Window
5
6 # Handle cancellations
7 If [ $function = "Cancel" ]
8     Close Window [ Current Window ]
9     Exit Script [ Text Result: ]
10 End If
11
12 # Establish dialog values
13 Set Variable [ $Body ; Value: "Are you sure you want to delete this " & Get ( LayoutTableName ) & " record?" ]
14 Set Variable [ $Prompt ; Value: "Type " & Quote ( "Delete" ) & " to confirm:" ]
15 Set Variable [ $Button1 ; Value: "Delete" ]
16 Set Variable [ $Button2 ; Value: "Cancel" ]
17
18 # Present dialog
19 New Window [ Style: Dialog ; Name: "Really Delete Record?" ; Using layout: "Dialog" (Dialog) ; Height: 175 ; Width: 451 ;
    Top: ( Get ( WindowDesktopHeight ) / 2 ) - ( 175 / 2 ) ; Left: ( Get ( WindowDesktopWidth ) / 2 ) - ( 451 / 2 ) ]
20 Set Field [ Dialog::Dialog Body ; $Body ]
21 Set Field [ Dialog::Dialog Button 1 ; $Button1 ]
22 Set Field [ Dialog::Dialog Button 2 ; $Button2 ]
23 Set Field [ Dialog::Dialog Confirmation Prompt ; $Prompt ]
24 Set Field [ Dialog::Dialog Confirmation Entry ; "" ]
25 Go to Field [ Select/perform ; Dialog::Dialog Confirmation Entry ]
26 Pause/Resume Script [ Indefinitely ]
27
28 # Continue with deletion

```

Figure 35-22. The complete `Delete Record` script

The script starts with some declarations in lines 2-4:

- Placing any script parameter into a `$function` variable.
- Turning off `Allow User Abort` so that the user can't exit the script using a method other than the buttons provided in the dialog.

- Using the Freeze Window command to lock the interface until it needs to be refreshed to avoid unnecessary distractions.

Then, an If statement encloses lines 8-9 so that they only run if the script parameter contains “Cancel” indicating that the user is already interacting with a dialog generated from a previous execution of the script and has clicked the Cancel button:

- Using the Close Window step, the dialog is closed, returning the user to the window they started in.
- The Halt Script step is used to stop the script, returning the interface to normal use.

If the script parameter did *not* indicate a cancellation process, the script assumes we are generating a new dialog and continues to lines 13-16. Here it initializes variables that will be used to populate fields in the dialog. The \$Body variable is the only one of these that contains dynamic text and must be initialized before the window changes layouts since it includes the name of the table in which the user is attempting to delete a record. Using the Get (LayoutTableName) to generate the dialog body message assumes that the layout table is named as a singular descriptive version of the type of records in that table. For example, a table named Contact or Company will make sense using the formula shown, but a table named in the plural (Invoices) or named abbreviated (Invo Recs) will not make sense when placed into sentence form.

The window is created in line 19 by the New Window step configured as shown in Figure 35-23.

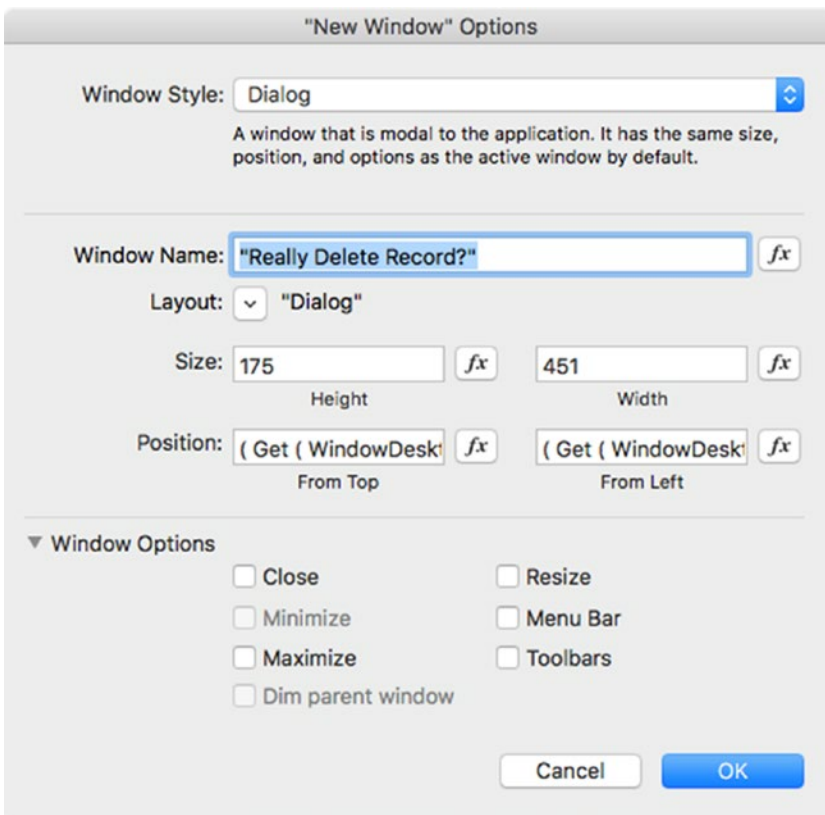


Figure 35-23. The New Window step configured to display a Dialog

The New Window step is configured as follows:

- `Window Style` is set to `Dialog` to generate a modal window that will lock up the application functions until the user dismisses it by making a choice.
- The `Window Name` is set to a static text value of “Really Delete Record?”
- The `Layout` menu and subsequent dialog is used to select the `Dialog` layout as the desired layout for the new window.
- The `Size Height` is set to 175 and the `Size Width` is set to 451, based on the dimensions of the layout’s content area plus room for the window title bar.
- The `Position From Top` contains the following formula that will center the new window vertically: $(\text{Get} (\text{WindowDesktopHeight}) / 2) - (175 / 2)$.
- The `Position From Left` contains the following formula that will center the new window horizontally: $(\text{Get} (\text{WindowDesktopWidth}) / 2) - (451 / 2)$.
- Finally, all the `Window Options` are unselected to keep the new window looking like a dialog and forcing the user to interact with the buttons provided rather than the standard close button.

Once the `New Window` step is run, the dialog will be the front window displaying the one record. Lines 20-24 fill in the field values for the message, button names, and confirmation prompt. Line 25 puts focus into the `Dialog Confirmation Entry` field, and line 26 pauses the script and waits for the user’s choice. When the user clicks the `Delete` button, the script resumes, closes the dialog, and deletes the record from the original window.

Finishing the Layout Configuration

Now that the script is configured, the following final configuration steps must be performed on the `Dialog` layout to connect buttons to the script and to confirm objects display properly on the layout.

Select the `Dialog Button 1` field and perform these two steps:

- Select the `Format > Button Setup` menu item and set the `Action` to perform a `Single Step of Resume Script` and check the `Change cursor to hand over button` option, as shown in Figure 35-24.
- In the `Data > Behavior` section of the `Inspector`, enter the following formula into the `Hide object when text area` so that the `Delete` button is inaccessible until the user enters the word “Delete” into the confirmation field.

```
Dialog::Dialog Confirmation Entry ≠ "Delete"
```

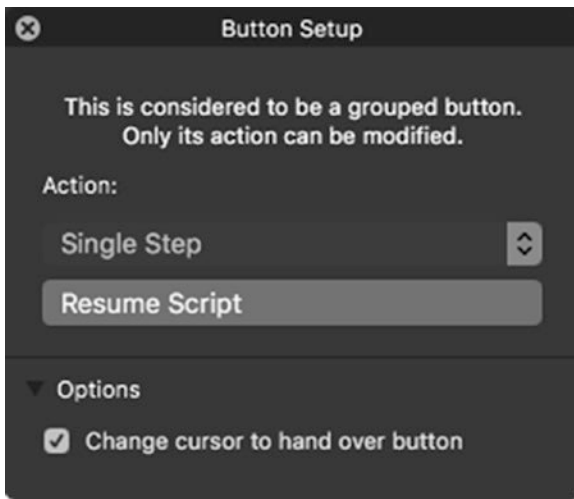


Figure 35-24. The configuration of the *Dialog Button 1* field on the *Dialog* layout

Select the *Dialog Button 2* field and configure it to run the *Delete Record {function}* script by following these steps:

- Select the *Format > Button Setup* menu item and set the *Action* to perform a *Perform Script*.
- In the *Specify Script* dialog, select the *Delete Record {Function}* script.
- Enter “Cancel” into the *Optional script parameter* text area.
- Click *OK* to close the *Specify Script* dialog.
- Select *Halt Current Script* in the *Options* area of the *Button Setup* dialog.
- Also, check the *Change cursor to hand over button* option. See [Figure 35-25](#).

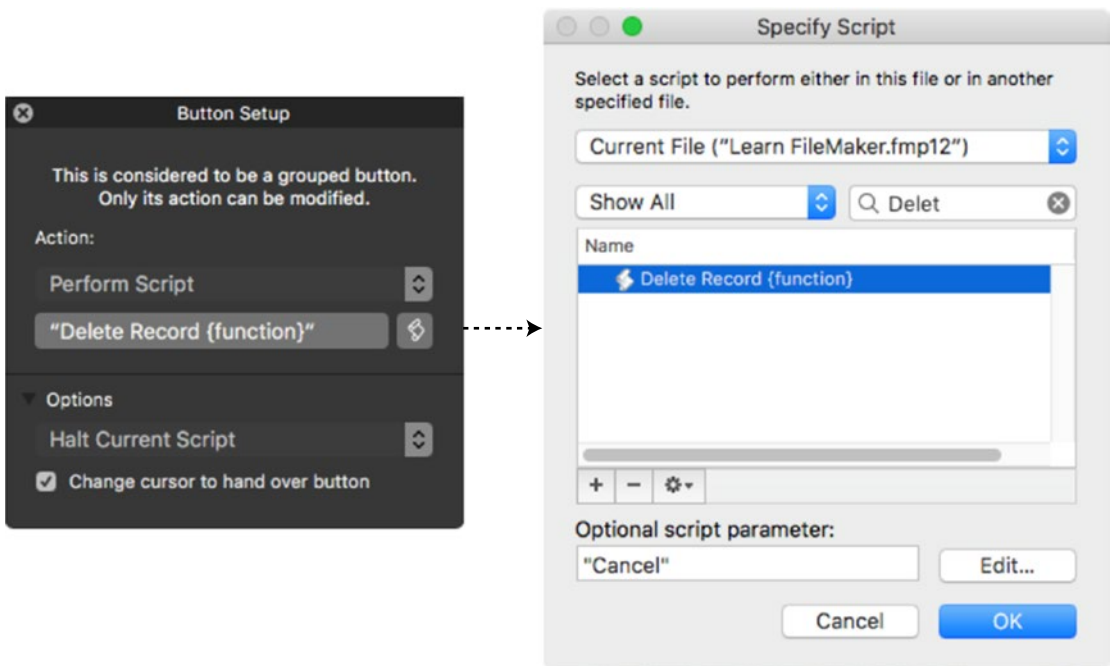


Figure 35-25. The configuration of the Dialog Button 2 field on the Dialog layout

In the Data > Behavior section of the Inspector, confirm that all the fields on the layout except Dialog Confirmation Entry have no Field Entry options selected.

Finally, select the Layouts > Set Tab Order and click Clear All in the Set Tab Order dialog. Then click the arrow pointing to the Dialog Confirmation Entry field and then on the arrow pointing to the right side of the Dialog Button 1 field/button, as shown in Figure 35-26. This will allow the user to type into the confirmation field and then tab to jump to the Delete button. Click OK to save the new tab order.

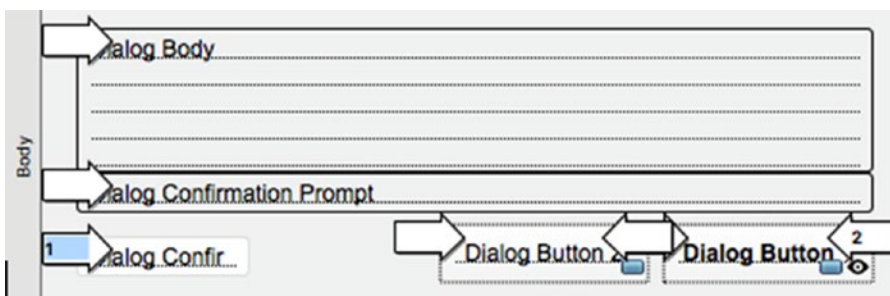


Figure 35-26. The tab order to allow tabbing between the confirmation text field and the first dialog field/button

Testing the Script

Once configured, it is time to test the script. Navigate back to a layout for a non-dialog table such as Contact Form. Eventually, this script should be connected to the Delete Record custom menu. However, for testing purposes, make it visible in the Scripts menu or add a button to the layout and attach it as the action. When run, you should see a dialog open like the one shown in Figure 35-27.

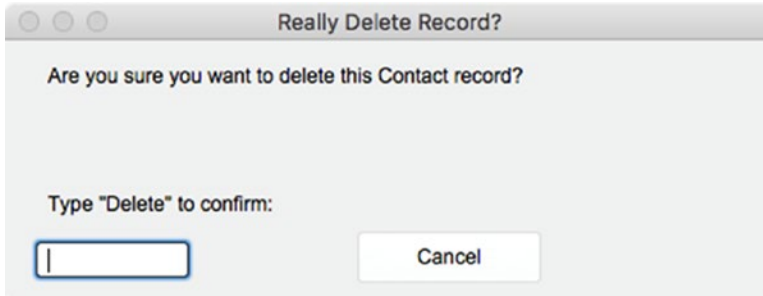


Figure 35-27. The delete dialog when the script is run from a layout for the Contact table

Type the word “Delete” into the confirmation box and hit the Tab key. That should make the Delete button visible, as shown in Figure 35-28.

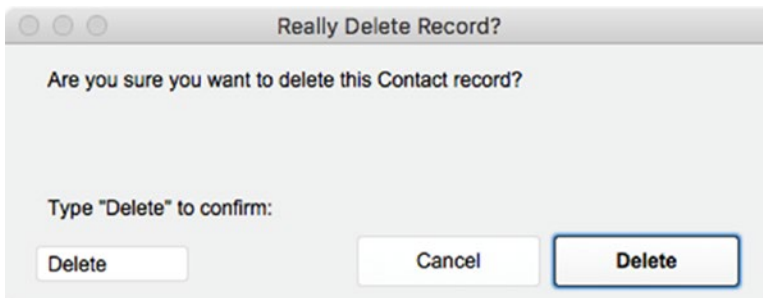


Figure 35-28. The delete dialog after the user types into the confirmation field

■ **Tip** This dialog and script can be evolved to include various enhancement. For example, the message could include a value from a standard Record Name field and could be expanded to handle a Delete All Records request.

Introducing the Card Window

A *card* is a modal window overlay displaying a second layout within the current window on top of its content, as shown in Figure 35-29. The parent window’s content becomes inactive and inaccessible while the card is open although, depending on the size of the card, that content can still be seen in the background of the window in an optionally diminished light. Unlike a dialog window, which is modal to the entire FileMaker application, a card only locks up one window leaving the user free to work in other windows at the same time it is displayed.

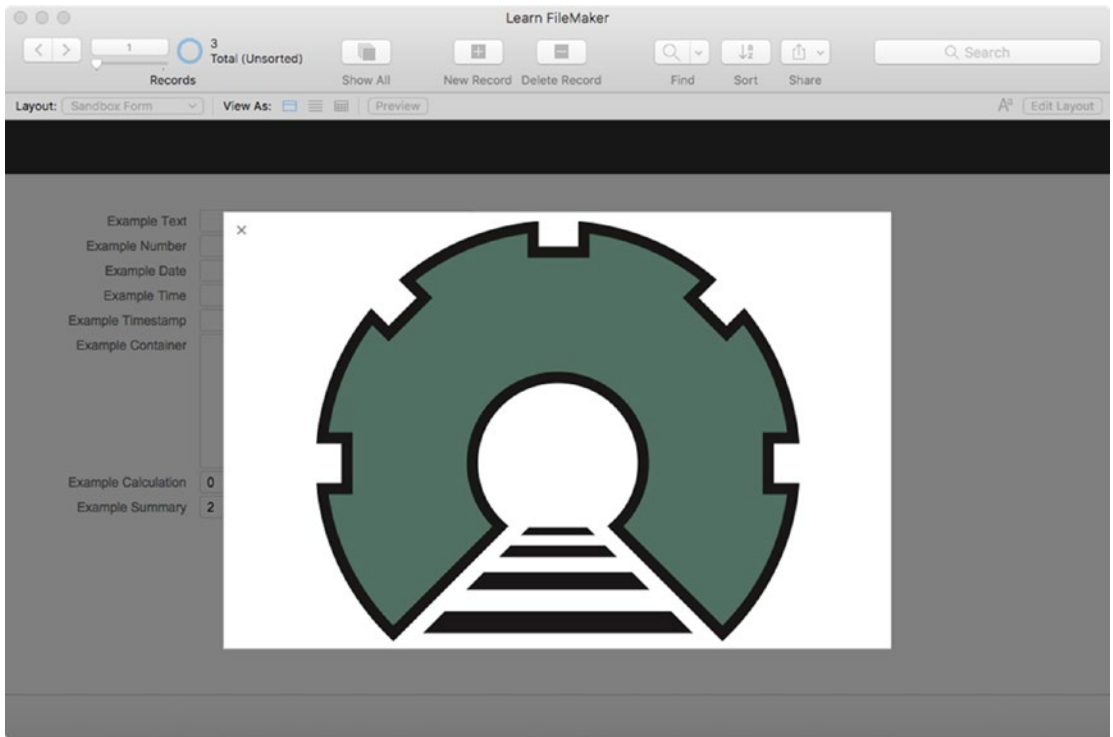


Figure 35-29. An example of a card superimposing a graphic layout on top of the Sandbox Form layout

Cards are similar in appearance and behavior to modal windows found in other environments. For example, a macOS dialog can open as a “sheet,” attached to the top of a window. Also, various types of objects are often displayed atop the content of a web page, including advertisements and images, which are sometimes loosely referred to as a “Lightbox” effect.

Cards are great for presenting navigation menus, custom toolbars, dialogs, and more. Since the card layout has its own context that can be based on a table occurrence other than that of the parent window’s layout, a card can provide universal services regardless of the parent window’s context.

■ **Tip** When configuring a Card in the New Window Options dialog, leave the Size and Position values empty to have the layout centered in the parent window displaying its content at full size.

Converting the Delete Confirmation Dialog to a Card

With the following adjustments to the New Window script step, the delete dialog window from the previous example can be converted into a card that appears to be attached to the top of the parent window like an macOS window dialog:

- Change the Window Style to Card.
- Change the Position > From Top to 0.

- Change the Position > From Left to a null (empty) value.
- Select the Dim parent window check box option.

Once saved, when the script is run, the dialog should appear attached to the top of the window, as shown in Figure 35-30.

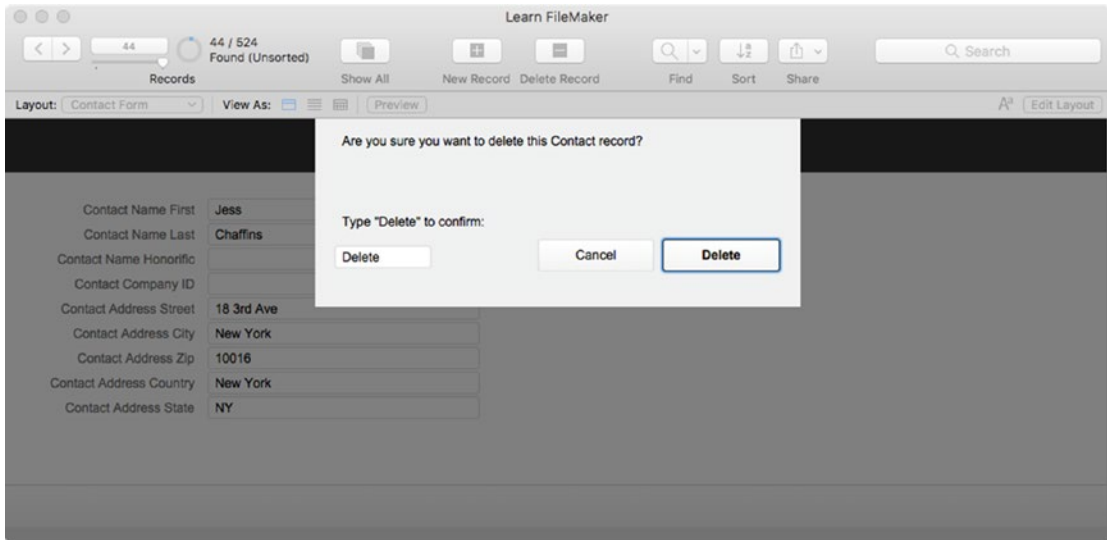


Figure 35-30. The dialog presented as a Card, attached to the top of the window

Caution When the Close Window option is disabled, the card will not include a Close button in the upper-left corner. You must include at least one button on the layout that allows the user to close the card or it may become necessary to force quit the application to release the window, especially if the script that opened the window is paused and has Allow User Abort turned Off.

Managing Errors

When an error occurs, a dialog may appear informing the user of the problem and temporarily pause the running script. The message in this dialog may or may not make sense to a user depending on the error that occurs or their knowledge of the script functions and of FileMaker development.

A common scripting error that presents a dialog is an attempt to use the Set Field step to set a value for a field from the wrong context. For example, if browsing a Company record, setting a field in the Contact table directly (not through a relationship) will result in the error message stating, “This operation could not be completed because the target is not part of the related table.” Since this is a programming error, a user would have absolutely no idea what this means or what to do to reach a resolution.

To suppress error displays, use the Set Error Capture step with a parameter of On. This will cause the script to ignore any errors it encounters and continue processing. The Get (LastError) function can be used at various points in the script to determine if an error has occurred. The function will return a numeric value indicating either no error (0) or the number representing the error. An If statement can take conditional action in the event an error is detected.

As an illustration of this technique, the example below shows a script attempting to set a the `Contact::Name Last` field and displaying a dialog. If this script is run from a different table, say a `Project` table, the `Set Field` step will generate a 103 error and display the dialog specified.

```
Set Error Capture [ On ]
Set Field [ "Contact::Name Last; "Smith" ]
If [ Get ( LastError ) = 103 ]
    Show Custom Dialog [ "Error 103: Set Field context error" ]
End If
```

Generally, it isn't necessary to put an error check and dialog on a granular level as shown in the preceding example, which is intended to illustrate the pattern of how errors can be captured, detected, and reported.

For a complete list of errors, search for "FileMaker Pro error code reference guide" on FileMaker.com.

Summary

In this chapter, we explored many examples of scripting tasks that can be used to automate activity within a database.

CHAPTER 36



Debugging Scripts

Troubleshooting an issue occurring within a simple script doesn't pose much of a challenge. As a database grows in complexity, several factors can adversely impact how easily a developer identifies and locates the cause of a scripting problem. A script can call a sub-script, which can call another in a branching hierarchy. Scripts are triggered from numerous interface elements such as buttons, layouts, menus, records, and more. Scripts can jump from one layout context to another, which can be difficult to keep track of during complex processes that are woven into a mesh of interconnected scripts. To ease the burden of figuring out what is going wrong and where, modern development environments have a feature called a script debugger.

A *script debugger* is an interactive tool used by developers to troubleshoot complex scripted processes, often those involving functions that range across multiple scripts. FileMaker Pro Advanced has a built-in script debugger that makes it easy to monitor variables, step through scripts and sub-scripts, set and clear custom breakpoints, pause on errors, and temporarily disable all script triggers to help locate and correct scripting problems.

In this chapter, we will introduce basic debugging, covering the following topics:

- How script debugging works
- Exploring the debugging interface
- Using the debugger
- Using custom breakpoints

How Script Debugging Works

The FileMaker script debugger is a powerful feature that can replace primitive troubleshooting techniques. For example, without a debugger, developers used to insert temporary dialogs and pauses to help identify which step had been reached prior to the point of error, what information is in variables or fields at that time, and what the current context is within the interface. These are incredibly crude methods of troubleshooting and require a lot of experimentation to locate and diagnose a problem. Also, when finished, they must be cleaned up. A formal debugger replaces all of this with powerful features that can be activated or deactivated as needed without modifying a single script.

Once debugging is enabled, the script being performed will be loaded into the **Script Debugger** window and pause at the first step it is about to perform. You can then choose how the script and any nested subscripts will be performed by choosing from a set of debugging functions that control where and when the process will be paused next. Essentially, debugging allows you to slow down the action of a script, breaking it into a sequence of pause-delimited functions, providing an opportunity to analyze the interface context and variable content at different points within a script or the sub-script stack.

Uses for the Script Debugger

The debugger has a variety of uses including:

- Discovering previously unknown scripting errors that are suppressed by uses of the `Set Error Capture` script step but may be adversely affecting database functionality (using the debugger's `Pause on error` feature).
- Locating the exact position of a programming error within a script.
- Moving slowly through the hierarchy of subscripts for troubleshooting or structural analysis of a complex database.
- Visualizing the interface context and variable content at various points within a script.
- Disabling script triggers temporarily to provide an opportunity to work around catastrophic issues that occur immediately when a file is opened.

Defining Basic Debugging Concepts

Before delving into debugging features, take a moment to become acquainted with the following terminology:

- *Breakpoint* — This is position within a script at which the debugger will pause and await instructions about how to continue. FileMaker will automatically pause at the beginning of a direct script launch and on an error step, if the option to do so is selected. But you can manually insert custom breakpoints anywhere within a script to fine-tune the troubleshooting experience.
- *Step* — This is one of a set of functions that define the distance to the next automatic breakpoint. When a script pauses at the start of the debug process, you have the option to take steps through it rather than allowing it to run quickly to the end. FileMaker offers the following step options:
 - *Step Over* — This function will execute the next step of the current script before pausing but will not enter and pause in sub-scripts. The sub-scripts will be performed but not debugged.
 - *Step Into* — This function will execute the next step of the current script before pausing and will enter sub-scripts and pauses them in the debugger.
 - *Step Out* — When the current script is a sub-script, this function will execute the remaining steps and then return to the parent script and pause on the line after the `Perform Script` step that called the sub-script.
 - *Set Next Step* — This function allows you to skip ahead from the last executed step to the selected step *without performing the interim steps*.
- *Continue* — When paused, this option will continue the script to the next breakpoint, error, or the end of the script.
- *Halt* — This option will abruptly abort the script.

Activating the Debugger

Script debugging is active whenever the Script Debugger window is open. This can be achieved in one of three ways:

- Open the Script Debugger window by selecting the Tools > Script Debugger menu item. While open, any script that runs will automatically be displayed in the window and run through the debugger. The first script to run will pause and allow you to control subsequent functions, including whether to step through subscrip
- Turn on Pause on error by selecting the check box at the bottom of the Script Debugger window or menu item under the Tools > Script Debugger. Once activated, close the debugger window and work normally. Anytime an error is encountered the Script Debugger window will open and will pause at the script step that is generating the error. This includes any errors that are automatically captured and suppressed with the Set Error Capture script step, allowing you to identify errors that you may not be aware are occurring.
- Click the Script Debugger button in toolbar of the Script Workspace window (Chapter 34) while viewing a script. This will open the Script Debugger window and immediately run the selected script through the debugging process.

Exploring the Debugging Interface

The debugging interface includes three items:

- The Tools > Debugging Controls submenu.
- The Script Debugger window.
- The Data Viewer window.

Debugging Options Under the Tools Menu

FileMaker's debugging features are accessed at the top of the Tools menu, as shown in Figure 36-1.

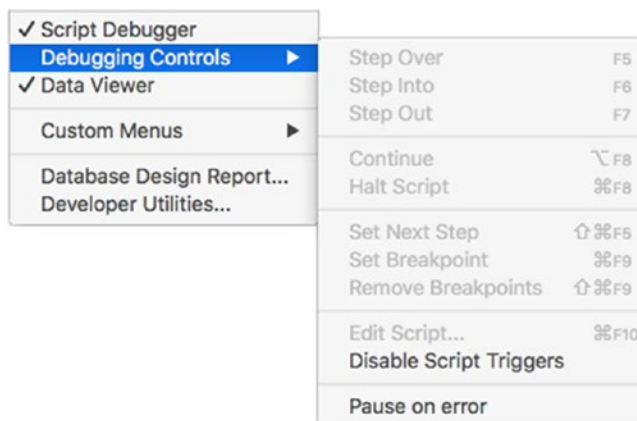


Figure 36-1. The Script Debugger menu items and Debugging Controls submenu

The following controls are available at the top of the Tools menu:

- **Script Debugger** — This menu item will toggle the visibility of the Script Debugger window, with a check mark indicating that it is currently open.
- **Debugging Controls** — This submenu provides key command access to various debugging functions (described later in this chapter) and displays the status of the **Disable Script Triggers** and **Pause on error** functions.
- **Data Viewer** — This menu item will toggle the visibility of the Data Viewer window, with a check mark indicating that it is currently open.

Exploring the Script Debugger Window

The Script Debugger window, shown in Figure 36-2, allows monitoring and interacting with the script execution progress.

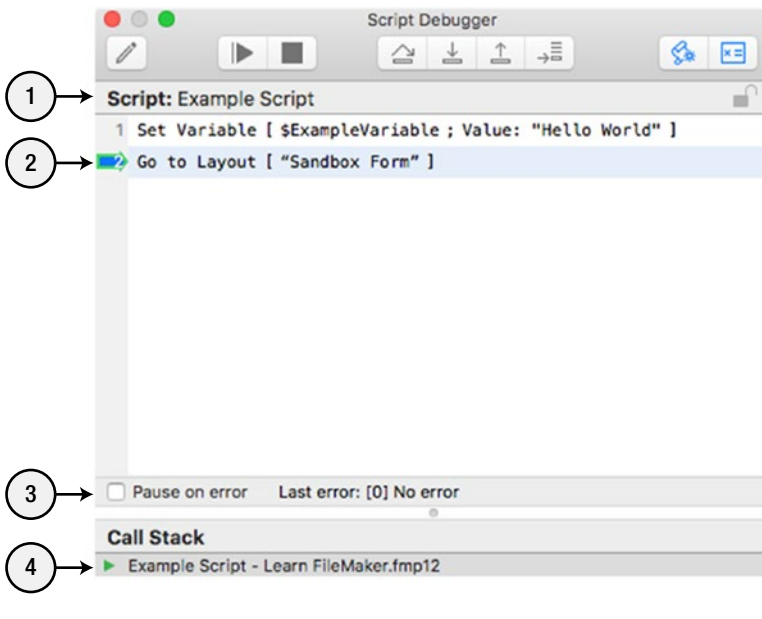


Figure 36-2. *The Script Debugger window*

The main portion of the debugger window contains the following interface elements:

1. **Script Name** — This area displays the name of the current script being displayed.
2. **Script Step** — This area displays the script steps that make up the current script. If the debugger is paused on a highlighted step, that indicates it has encountered a natural breakpoint, custom breakpoint, or error and is waiting for you to choose to stop, continue, or take some other action.

3. **Pause on error** — When this check box is enabled the Script Debugger will pause any time an error occurs, opening the dialog if necessary.
4. **Call Stack** — This area contains the “stack” of script calls based on the script paused in the view. If viewing a sub-script that was called by other scripts, the disclosure triangle can be clicked to expand the hierarchy above the current script, tracing back the path to the first script in the stack.

Status Toolbar Items (Debugger Window)

The non-customizable status toolbar for the Script Debugger window is shown in Figure 36-3.

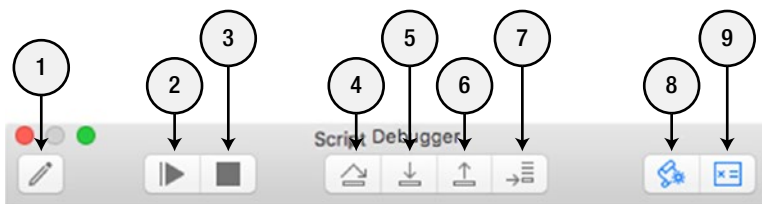


Figure 36-3. The Script Debugger window toolbar

The toolbar contains the following buttons, each connected to a debugging function that is also available under the **Tools > Debugging Controls** submenu:

1. **Edit Script** — Opens the current script in the Script Workspace window.
2. **Continue** — Continues running the script currently paused in the window.
3. **Halt Script** — Stops the script and any other scripts in the call stack.
4. **Step Over** — Continues running the script *without* debugging sub-scripts.
5. **Step Into** — Continues running the script *with* debugging sub-scripts.
6. **Step Out** — Continues the current sub-script without debugging until control is returned to the parent script.
7. **Set Next Step** — Jump ahead to the script step that is selected in the window, skipping the interim steps.
8. **Disable Script Triggers** — This button will toggle the setting to disable script triggers throughout the database file.
9. **Open Data Viewer** — This button will open the Data Viewer dialog.

Exploring the Data Viewer Window

The *data viewer* is a dialog that displays a list of variables and their current value, as shown in Figure 36-4. The dialog can be used at any time to review global variables. When used in conjunction with the debugging feature, it allows you to see the current value of both global variables and any current local variables defined when the script was paused.

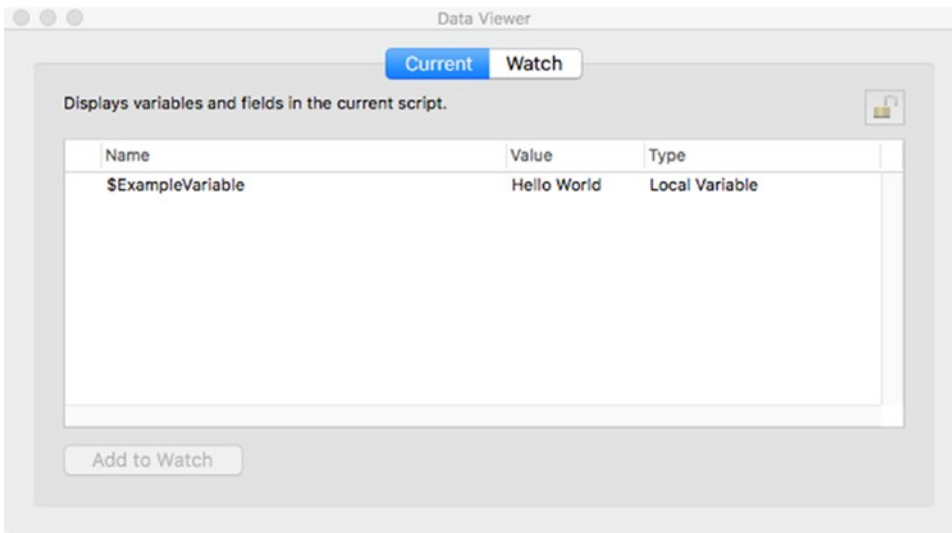


Figure 36-4. The *Data Viewer* dialog is a diagnostic tool that is used to monitor the value of global and local variables when a script is paused

The dialog is divided into the following two tabs:

- **Current** — This tab contains a list of every variable currently defined. Global variables will show up all the time while local variables, being transitory, will only be present when a script is paused. Double-click on a variable to open a **Current Value** dialog to view the contents of lengthy values. Select a variable and click **Add to Watch** to add that variable to the **Watch** tab's list of monitored expressions. When the **Current Value** dialog is open, you can modify the value in the variable while a script is paused.
- **Watch** — This tab is used to monitor any number of factors. A set of buttons allows you to add, duplicate, edit, and delete the list of expressions. Use this to test formulas before integrating them into a database formula or custom function.

Using Custom Breakpoints

Custom breakpoints are developer specified pauses within a single script that help pinpoint the location of an error.

To add a breakpoint to a script, open the script in the workspace then click on the step number, which is running in a column to the left of the steps. A breakpoint is indicated when the number is illuminated in a blue arrow, as shown in [Figure 36-5](#).

```

Hello World
1
2 # Extract input parameter
3 Set Variable [ $State ;
   Value: Let ( [ input = Get ( ScriptParameter ) ; state = Case ( input ≠ ""_ ]
4
5 # Find all contacts by state
6 Go to Layout [ "Contact" (Contact) ]
7 Enter Find Mode [ Pause: Off ]
8 Set Field [ Contact::Contact Address State ; $State ]
9 Perform Find [ ]
10
11 # Compile a list of contact information for the found set
12 Go to Record/Request/Page [ First ]
13 Loop
14   Set Variable [ $Names ;
     Value: $Names & Contact::Contact Name First & " " & Contact::Contact N... ]
15   Go to Record/Request/Page [ Next ; Exit after last ]
16 End Loop
17
18 # Send list as email
19 Send Mail [ Send via E-mail Client ; With dialog: On ;
   To: "jane.accounting@widget.com" ; Subject: "Contacts from " & $State ;
   Message: $names ]

```

Figure 36-5. A script with two breakpoints added

Once in place, a script run through the debugger will pause at the beginning and then at each of the two breakpoints.

Breakpoints can be removed by any of the following methods:

- Within a script by clicking on each blue arrow.
- When a script is paused in the debugger, all breakpoints can be removed by selecting the Tools > Debugger Controls > Remove Breakpoints menu item.

Summary

In this chapter, we explored the benefits and features of the script debugger, which is a valuable tool to assist in troubleshooting databases with complex scripting.

CHAPTER 37



Using Script Triggers

A script connected to a button or one made visible in the Scripts menu will run in response to a user's click. While this gives users easy access to a time-saving and error-avoiding sequence of predefined functions, it has one glaring limitation. The click-to-run approach to scripting requires the user to consciously intend to perform the script function, knowing when and how to invoke it. So, while the script will *automatically* perform the steps defined within, it must be *manually* triggered. As a developer, you may not want to always rely on a user to remember to take the right action at the appropriate time. Script triggers solve this problem.

A *script trigger* is a mechanism that connects a specific interface event to a script so that any time a user or other script performs the event, the script runs automatically. This feature is like most application development environments, triggering a function in reaction to an interface event. FileMaker has over two-dozen triggers for a variety of file, layout, and object events. For example, a script can be configured to run after the database file opens or before the database file closes. Similarly, a script can run when a layout is entered or exited; when a field is entered, edited or exited; when a window changing modes; and more.

Every trigger runs in response to an event. However, events trigger a script either before or after the user's intended action is completed. For example, when modifying a field or navigating to a layout, the script is triggered after that event is completed. However, some scripts are triggered *before* the event is completed. In those cases, if the script returns a false value of 0, the script triggering action is not completed.

Scripts executed by a trigger are the same as any other script in FileMaker. In fact, events can trigger scripts that are connected to a button or available in the Scripts menu. Different events or the same event from different objects can each have their own script, or they can share a script depending on each developer's preference or the practicality of doing so within a specific context.

In this chapter, we will introduce script triggers, covering the following topics:

- Script trigger reference
- Working with script triggers

Script Trigger Reference

Script Triggers are available in the following three areas:

- File option triggers
- Layout triggers
- Layout object triggers

File Option Triggers

A *file option trigger* will run a script in response to a file level event. These triggers are available in the Script Triggers tab of the File Options dialog, shown in Figure 37-1, which can be accessed by selecting the File > File Options menu.

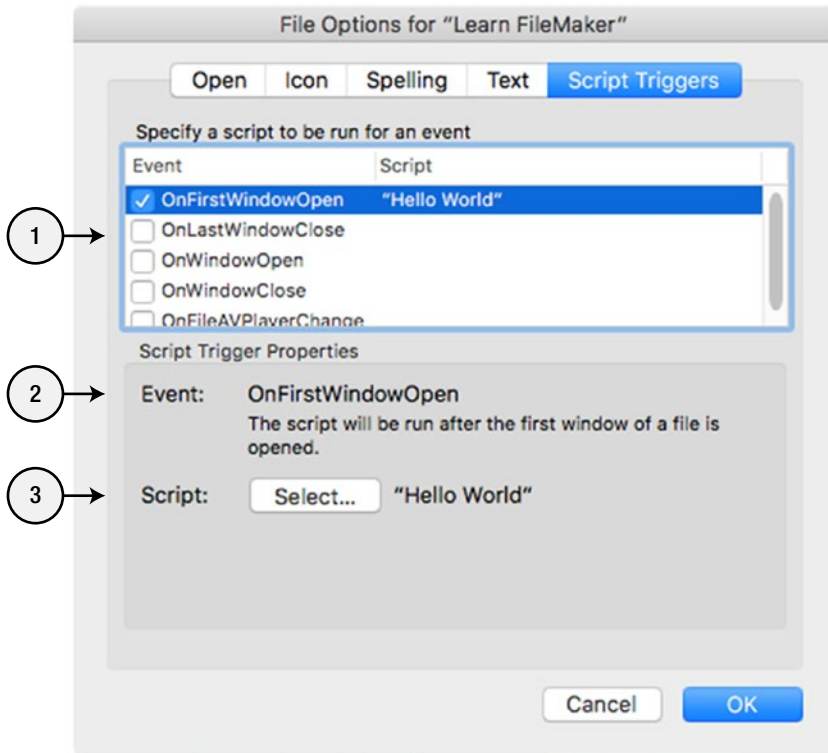


Figure 37-1. The Script Triggers tab for the File Options dialog

The dialog contains the following controls:

1. Events — The area contains a list of all events that can be configured to trigger a script. The check box indicates that the event is active and will trigger the script displayed on the right.
2. Event — The selected event name will be displayed here with a brief description.
3. Script — If a script is assigned to the selected event, it will be displayed here. Click the Select button to specify which script should be assigned to the event.

The following file triggers are available:

- OnFirstWindowOpen
- OnLastWindowClose
- OnWindowOpen
- OnWindowClose
- OnFileAVPLayerChange

OnFirstWindowOpen

The OnFirstWindowOpen event occurs when the *first* window is created after a database is explicitly opened by a user or Open File script step in another file. It provides an opportunity to perform any configuration or setup routines for the file itself at startup. For example, this event might trigger a script that initializes global variables, automatically opens other windows, or performs default finds and sorts, etc.

■ **Caution** The OnFirstWindowOpen event does *not* occur if the file opens as a hidden window due to being automatically opened by a relationship to it from another database file.

The event will trigger a script *after* the event has been processed and the window has been opened.

OnLastWindowClose

The OnLastWindowClose event occurs when the *last* layout window for the file is closed by a user or a Close Window script step. It provides an opportunity for a script to perform any last-minute validation functions or maintenance routines.

■ **Note** This event is based on the last non-developer window being closed and will not consider other windows such as the Script Workspace.

The event will trigger a script *before* the event has been processed. If the script returns a false value of 0, the Close process will be halted and the window will remain open.

OnWindowOpen

The OnWindowOpen event occurs after any new window is created in response to:

- A user selects the Window > New Window menu item.
- A script runs that uses the New Window step.
- A window for a hidden file is brought into focus by a user or script step for the first time.

This event provides an opportunity to perform any configuration or setup routines for the window. For example, this event might trigger a script that performs a default find and sort.

The event will trigger a script *after* the event has been processed.

OnWindowClose

The OnWindowClose event occurs before any window is closed.

This event provides an opportunity for a script to perform any last-minute validation functions or maintenance routines to the record(s) viewed within the window.

The event will trigger a script in non-Layout modes *before* the event has been processed. If the script returns a false value of 0, the process will be halted and the window will remain open.

OnFileAVPlayerChange

The OnFileAVPlayerChange event occurs whenever a user or script changes the playback state of a media file playing in a field or URL. For example, the event will trigger the script when a user pauses, plays, or stops the media or when the media stops upon reaching the end.

The event will trigger a script in Browse and Find modes *after* the event has been processed.

■ **Caution** This event is only supported in FileMaker Go.

Layout Triggers

A *layout trigger* will run a script in response to an event while a user is accessing records on a particular layout. These triggers are available in the Script Triggers tab of the Layout Setup dialog, shown in Figure 37-2, which can be accessed by selecting the Layout Setup from the layout's contextual menu or by clicking the pencil icon to the right of the Layout menu in the lower row of the window toolbar.

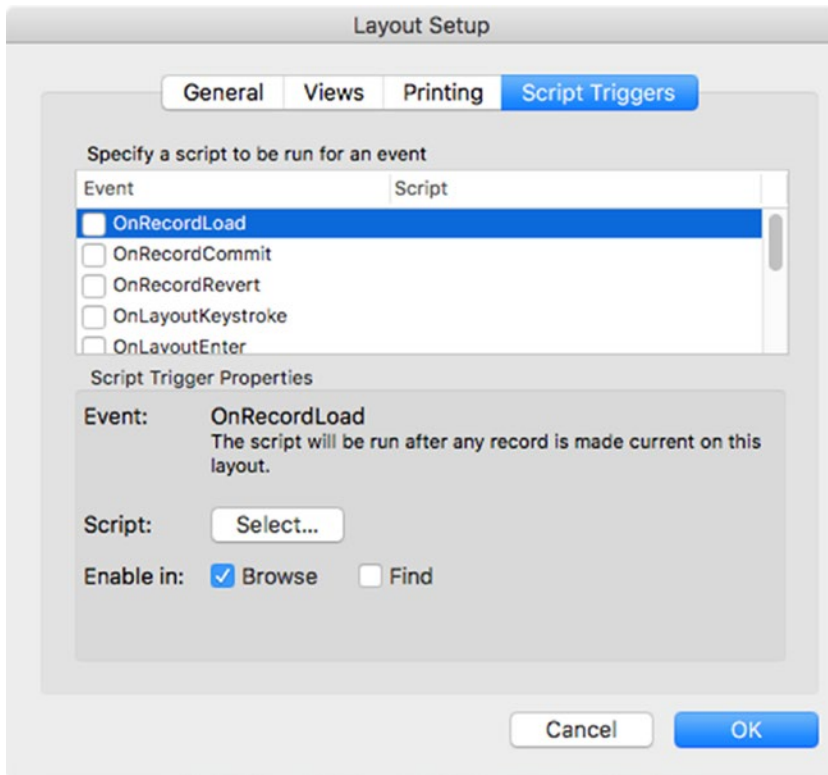


Figure 37-2. *The Script Triggers tab for the Layout Setup dialog*

The dialog controls are the same as those on the File Options dialog with one addition:

- **Enable in** — These check boxes allow the selected script trigger to be enabled or disabled for both Browse or Find window modes.

The following layout triggers are available:

- OnRecordLoad
- OnRecordCommit
- OnRecordRevert
- OnLayoutKeystroke
- OnLayoutEnter
- OnLayoutExit
- OnLayoutSizeChange
- OnModeEnter
- OnModeExit

- `OnViewChange`
- `OnGestureTap`
- `OnExternalCommandReceived`

OnRecordLoad

The `OnRecordLoad` event will occur after a record becomes current, providing an opportunity for a script to perform actions just prior to a user or script viewing or modifying a record.

The event will trigger a script *after* a user or script performs any of the following record-loading events:

- Navigating to a new record within a found set
- Navigating to a different layout
- Creating a new record
- Deleting a record in a found set of more than one record
- Creating a new window
- Displaying the results of a Find operation

This event will occur anytime a window displays a record that wasn't currently displayed in the moment before on a layout that has a script assigned to this trigger.

OnRecordCommit

The `OnRecordCommit` event will occur when an attempt is made to commit a modified record by a user or a script performing any of the following actions:

- Navigating to a different record
- Typing the Enter key
- Clicking anywhere on the layout outside of the fields of the active record
- Changing records; navigating, deleting, creating, duplicating, etc.
- Performing a script containing the `Commit Records/Requests` step
- Opening the Manage Database dialog window
- Closing the file

This event allows a script to perform validation steps or other actions prior to saving changes made to the record.

The event will trigger a script *before* the event has been processed. If the script returns a false value of 0, the `Commit` process will be halted and the uncommitted changes will remain in place on the record, which will still be in focus.

OnRecordRevert

The `OnRecordRevert` event will occur when an attempt is made to revert a modified record to its previously committed state by a user selecting the `Records > Revert Record` menu or a script is run with the `Revert Record/Request` step.

This event allows a script to provide a custom dialog message warning the user to perform any necessary actions prior to removing recent changes and restoring the record to its last saved state.

The event will trigger a script *before* the event has been processed. If the script returns a false value of 0, the Revert process will be halted, and the uncommitted changes will remain in place on the record, which will still be in focus.

OnLayoutKeystroke

The OnLayoutKeystroke event will occur when any character is typed on the keyboard except those handled by the Operating System or FileMaker. For example, the event does occur when typing navigational keys such as Tab and arrows but is not sent when typing a command by holding down the macOS Command key or the Windows Alt and Control keys, or function keys like volume buttons.

The script triggered by this event can use various functions to determine the keystroke that triggered it and what, if any, field has focus and is being typed into. Some useful functions here include:

- Get (TriggerModifierKeys)
- Code
- Char
- Get (ActiveFieldName)

The event will trigger a script *before* the event has been processed. If the script returns a false value of 0, the keystroke will not be processed.

■ **Note** See OnObjectKeystroke later in this chapter.

OnLayoutEnter

The OnLayoutEnter event will occur after a layout has loaded. It will trigger a script *after* the event has been processed.

OnLayoutExit

The OnLayoutExit event will occur when a user or script attempts to navigate away from the current layout or to close a window. It will trigger a script *before* the event has been processed. If the script returns a false value of 0, the layout change will be halted, and the window will continue to display the previously displayed layout.

OnLayoutSizeChange

The OnLayoutSizeChange event will occur when the window has changed size due to a user or script performing any of the following actions:

- Manually resizing the window by dragging
- Running a script that contains a Resize Window step
- Hiding or showing the window toolbar or formatting bar

- Opening a window
- Rotating the iOS device (FileMaker go only)

The event will trigger a script *after* the event has been processed.

OnModeEnter

The OnModeEnter event will occur after the window mode is changed by a user or a script step. It will trigger a script *after* the event has been processed.

OnModeExit

The OnModeExit event will occur when a user or script attempts to change the window mode. It will trigger a script *before* the event has been processed. If the script returns a false value of 0, the mode change will be halted, and the window will remain in whatever mode it was previously displaying.

OnViewChange

The OnViewChange event will occur when a user or script changes the window view between Form, List, or Table. It will trigger a script *after* the event has been processed.

OnGestureTap

The OnGestureTap event will occur when a tap gesture is received on a layout under the following conditions:

- FileMaker go and windows only (except Windows 7)
- Browse and find modes only
- Tap is not made in active web viewers or active edit boxes
- Single tap is one, two, or three fingers (iOS)
- Double tap with one finger (iOS), which will trigger the event twice
- Tap is with two fingers (Windows)

The triggered script can use the Get (TriggerGestureInfo) function to determine details about the gesture event.

The event will trigger a script *before* the event has been processed. If the script returns a false value of 0, the gesture will not be processed.

■ **Caution** This event is only supported in FileMaker Go and Windows.

OnExternalCommandReceived

The OnExternalCommandReceived event will occur when a user presses a button to control playback functions (play, pause, stop, next, previous, etc.) on an iOS device lock screen or on an external device. The triggered script can use the Get (TriggerGestureInfo) function to gain information about the external event.

The event will trigger a script *before* the event has been processed.

■ **Caution** This event is only supported in FileMaker Go.

Layout Object Triggers

A *layout object trigger* will run a script in response to an event performed to or within a specific layout object. These triggers are available in the dialog shown in Figure 37-3, which can be accessed by selecting a layout object and then clicking on the Format > Set Script Triggers menu or from the object's contextual menu. The dialog controls are the same as those on the Layout Setup dialog.

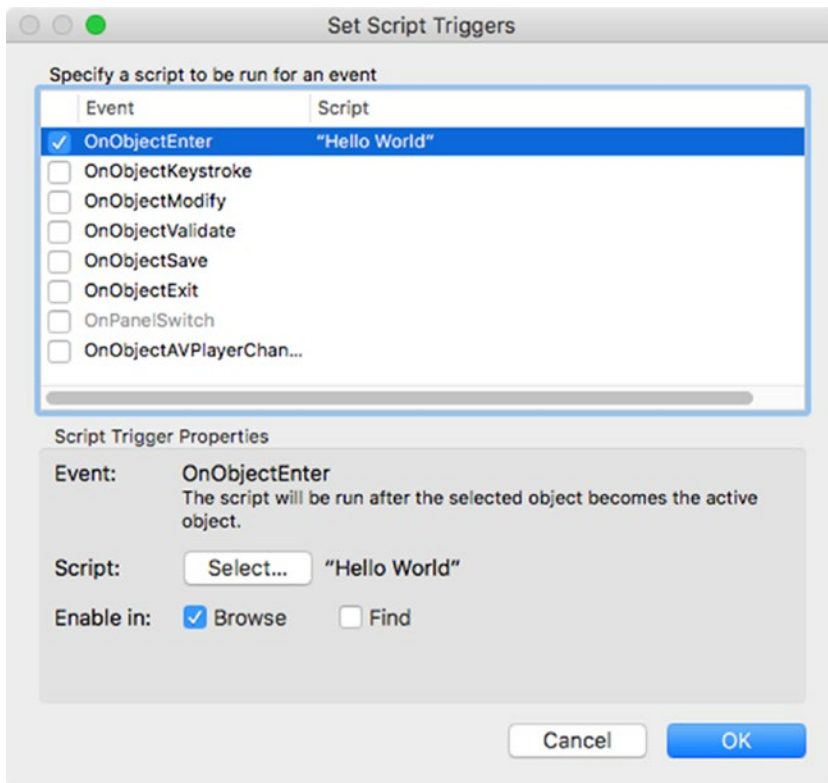


Figure 37-3. The Set Script Triggers dialog for a selected layout object

The following layout triggers are available through this dialog:

- OnObjectEnter
- OnObjectKeystroke
- OnObjectModify
- OnObjectValidate
- OnObjectSave

- OnObjectExit
- OnPanelSwitch
- OnObjectAVPlayerChange

OnObjectEnter

The OnObjectEnter event will occur when a layout object becomes entered and receives active focus. This can occur when a user clicks on the object, tabs into the object, or when a script is run that contains a step causing an object to be entered: for example, the Go to Object or Go to Next Field steps.

The event will trigger a script *after* the event has been processed.

OnObjectKeystroke

The OnObjectKeystroke event will occur when any character is typed on the keyboard into an object with focus except those handled by the operating system or FileMaker.

The event will trigger a script *before* the event has been processed. If the script returns a false value of 0, the keystroke will not be registered and the character will not appear in the object.

■ **Note** See OnLayoutKeystroke earlier in this chapter.

OnObjectModify

The OnObjectModify event will occur when the value of an object has changed because of any edit events including:

- Typing keystrokes into the object
- Selecting a new value in a value list formatted field (check box, radio button, etc.)
- Using Cut, Paste, or Clear functions
- Changing the value of the object with a script step such as Insert Text
- The selection of a new tab on a panel control

The event only occurs in response to interface-related actions. It does *not* occur when a window updates values that are populated from an external data source, when a field is updated by an auto-enter formula, or when the Set Field script step is used to update the value of a field.

The event will trigger a script *after* the event has been processed.

OnObjectValidate

The OnObjectValidate event will occur when a field value has changed and is about to be validated prior to being saved. This occurs when a user tabs to a new field or commits a record after making changes to the field with active focus.

The event will trigger a script *before* the event has been processed. If the script returns a false value of 0, the validation and exit events will be halted and the current object will retain focus.

OnObjectSave

The `OnObjectSave` event will occur after a modified field is in the process of being exited when focus is moving to a new field or a record is being committed.

The event will trigger a script *after* the event has been processed. If the script returns a false value of 0, the exit object event will be halted and the current object will retain focus; however, the changes will have already been saved.

OnObjectExit

The `OnObjectExit` event will occur when an object is about to lose active focus:

- Move from one object with focus to another
- Moving to another portal row
- Moving to another repetition in a repeating field
- Any action that causes the record to be committed
- Opening a dialog that causes a field to be exited

The event will trigger a script *before* the exit event has been processed. If the script returns a false value of 0, the event will be halted and focus will remain active on the current object.

OnPanelSwitch

The `OnPanelSwitch` event will occur when the current tab of a panel control is about to change due to a user selecting a different tab or a script step selecting an object on a different tab.

Use the `Get (TriggerCurrentPanel)` and `Get (TriggerTargetPanel)` functions to determine which individual panel of the control is being exited from and which has been selected to receive active focus.

The event will trigger a script *before* the event has been processed. If the script returns a false value of 0, it will cause the panel change to be halted.

OnObjectAVPlayerChange

The `OnObjectAVPlayerChange` event will occur when the media in a layout object changes state. For example, the event will trigger a script when a user pauses, plays, or stops a video in a field or when the video stops upon reaching the end.

The event will trigger a script *after* the event has been processed.

■ **Caution** This event is only supported in FileMaker Go.

Working with Script Triggers

There are a few topics to consider that will help to improve the efficiency of working with triggers and ensure that the results match your expectations:

- Finding script triggered objects
- Remembering event precedence

- Using caution with script parameters
- Exceptions to event triggers

Finding Script-Triggered Objects

When the View > Show > Script Triggers menu setting is active, objects with script triggers assigned will display a small script icon badge when viewed in layout mode, as shown in Figure 37-4.



Figure 37-4. The object badge visible on a field with one or more script triggers assigned

■ **Note** Object Badges are helpful in identifying various features used by a layout object. They are discussed further in Chapter 15.

Remembering Event Precedence

When a single interface event triggers multiple events, FileMaker will execute the script triggers in an order of precedence. To help can clarify the options for precisely triggering the script you want at a specific time and avoiding duplicate executions, here are a few examples of the order of precedence for different actions.

When *opening a database file*, the following sequence of events will be triggered in this order:

- OnFirstWindowOpen
- OnWindowOpen
- OnLayoutEnter
- OnModeEnter
- OnRecordLoad

When *committing a record with unsaved changes*, the following sequence of events will be triggered in this order:

- OnObjectValidate
- OnObjectSave
- OnObjectExit
- OnRecordCommit
- OnRecordLoad
- OnObjectEnter

When *opening a new window*, the following sequence of events will be triggered in this order:

- OnWindowOpen
- OnLayoutEnter
- OnModeEnter

- OnRecordLoad
- OnLayoutSizeChange

When *changing layouts*, the following sequence of events will be triggered in this order:

- OnLayoutExit
- OnLayoutEnter
- OnRecordLoad

■ **Caution** If the layout change is initiated from a window that has unsaved changes, the sequence listed here will be preceded by the entire record commit sequence listed previously.

Using Caution with Script Parameters

When assigning a script to an event trigger, the standard Specify Script dialog allows for the inclusion of a script parameter. Like with buttons and the Perform Script step, using parameters as part of an event trigger can help keep the target scripts flexible and open ended by pushing in specific information rather than encoding it within the script. However, with script triggers it is a good idea to use parameters cautiously.

Many events trigger a script *before* the event is completed. This means that the action the user is taking has not yet been realized on the interface. When the script runs, the interface will reflect the starting point of the event rather than the intended state that will be in place after the action is fully processed. In these cases, the entire time the script is running, it will be as if the user has taken no action at all because that action has not yet been processed by FileMaker. Due to this, it is important to maintain an awareness of exactly *when* an event will perform the assigned script and what is and is not safe to push in a script parameter.

The OnObjectKeystroke event provides a great example. When the user types a character into a field, the event will trigger the assigned script *before* the character is entered in a field. If the call to the script includes a script parameter with the contents of the field, it will not include the newly typed character.

Some triggers performed before an event is processed have special functions that allow you to extrapolate the intended action or target. In the example above, the Get (TriggerKeystroke) function will return the character typed but not yet entered into the field. The following trigger functions can only be used within a script that is triggered by an event and should not be used as script parameters:

- Get (TriggerCurrentPanel)
- Get (TriggerExternalEvent)
- Get (TriggerGestureInfo)
- Get (TriggerKeystroke)
- Get (TriggerModifierKeys)
- Get (TriggerTargetPanel)

Since triggers often occur in succession, if one event doesn't accommodate your needs, another may be used instead. For example, OnObjectExit doesn't offer a way to know which layout a user is requesting. In this case, use OnObjectEnter instead, which is triggered after the layout change occurs.

Script parameters should never be completely excluded from use with event triggers. However, take care when using them and be certain to have a clear understanding of what can be pushed in a parameter and when.

Exceptions to Event Triggers

Under some circumstances, events will not *always* trigger scripts so be careful when relying on script triggers to perform critical functions.

For example, most `Spelling` and `Find/Replace` functions, especially when performed across found sets, will not trigger various events such as `OnObjectModify`, `OnObjectValidate`, and `OnObjectSave`. Also, many functions `Import Records`, `Replace`, and `Relookup` modify table data directly and won't trigger events even though these must operate from a specific layout context. Similarly, script steps like `Set Field`, `Set Field by Name`, and `Set Web Viewer` will not active many script triggers.

Be sure to consider all possible actions available to a user or scripts and, where necessary, lock down functions or use alternative script steps to ensure that critical functionality is triggered. For example, the `Insert Text` or `Insert Calculated Result` script steps will trigger script steps when `Set Field` will not. When non-triggering steps can't be avoided, add steps to programmatically trigger other scripts or to explicitly perform the desired functionality.

Summary

In this chapter, we continued our discussion of scripting, specifically triggering scripts in reaction to interface events.

CHAPTER 38



Extending Features with Plugins

A *plug-in* is a software package developed with the C or C++ programming language that extends the capabilities of FileMaker Pro by adding new functions accessible through the Specify Calculation dialog and new script steps.

This chapter covers basic plug-in concepts including:

- Purchasing plug-ins
- Installing plug-ins
- Accessing plug-ins

Purchasing Plug-Ins

Many companies sell plug-ins for FileMaker Pro. A few prominent examples are included here for your convenience.

360 Works

360 Works sells about two dozen different FileMaker plug-ins, including:

- *360Works Email* — This plug-in allows you to send and receive email messages directly from within a database, supporting SMTP, POP, and IMAP protocols.
- *360Works Plastic* — This plug-in allows credit card processing directly from within a database supporting both Authorize.net and Verisign/PayPal Payflow.
- *360Works Web Services Manager* — This plug-in publishes FileMaker scripts as XML Web Services that can be accessed by SOAP (Simple Object Access Protocol)-compatible software and can be used to power public Web Services.

For more information visit the 360 Works website: 360works.com.

Productive Computing

Productive Computing sells several useful FileMaker plug-ins, including:

- *Address Book Manipulator* — This plug-in allows bidirectional data exchange between a database and the macOS Contacts application.
- *iCal Manipulator* — This plug-in allows bidirectional data exchange between a database and the macOS Calendar application.

- *File Manipulator* — This plug-in allows a database to copy, rename, move, delete, compress, and encrypt files or folders in local or networked directories.
- *Biometric Fingerprint Reader* — This plug-in adds the ability to incorporate fingerprint security and script control options to a database.

For more information visit the Productive Computing website: productivecomputing.com.

Trio Automatisering

Troi Automatisering sells many different FileMaker plug-ins, including:

- *Troi Dialog* — This plug-in allows a database to display dynamic dialogs with an icon, a progress bar, up to four buttons, up to fifteen input fields, and more.
- *Troi File* — This plug-in allows files stored anywhere in local or networked directories to be accessed from within a database. Create files, get data from fields, manipulate files, and more.

For more information visit the Troi Automatisering website: troi.com.

Finding Other Plug-ins

There are a several sites that provide index listings of available FileMaker plug-ins. These include:

- *FileMaker’s Website* — solutions.filemaker.com/made-for-filemaker
- *FileMaker Plug-ins* — filemaker-plugins.com
- *FileMaker World* — filemakerworld.com/pages/plugins1.html

Locating Installed Plug-Ins

FileMaker Pro plug-ins are installed in different folder locations depending on the operating system and product, as shown in Table 38-1.

Table 38-1. *The FileMaker plug-in folder*

Product	OS	Plug-In Folder
FileMaker Pro	macOS	Macintosh HD/Users/<user>/Library/Application Support/FileMaker/FileMaker Pro/16.0/Extensions
	Windows	C:\users\<user>\AppData\Local\FileMaker\FileMaker Pro\16.0\Extensions\
FileMaker Pro Advanced	macOS	Macintosh HD/Users/<user>/Library/Application Support/FileMaker/FileMaker Pro Advanced/16.0/Extensions
	Windows	C:\users\<user>\AppData\Local\FileMaker\FileMaker Pro Advanced\16.0\Extensions\

Installing Plug-Ins

Every computer that will access a database that uses plug-in functions must have the plug-in installed locally. Plug-in installation resources can be embedded in and installed from a database file, either a separate installer database or directly within a database solution. When purchased from a professional developer, the plug-in will most likely be delivered to you in an installer database that you can run to install it.

To prepare a database to install a plug-in, create a container field(s) for the plug-in file(s). Plug-ins can only be installed if they are the correct architecture for the operating system and version of FileMaker, so a mixed environment will require one field for each version of the plug-in. Next, create a script that uses the `Install Plug-in File` script step with references to the appropriate container field. For example:

```
If [ Get ( ApplicationArchitecture ) = "i386" ]
    Install Plug-in File [ Resources::Plugin 32bit ]
Else if [ Get ( ApplicationArchitecture ) = "x86_64" ]
    Install Plug-in File [ Resources::Plugin 64bit ]
End
```

■ **Note** In Windows, plug-ins will have an extension of either `.fmx` (32-bit version of FileMaker) or `.fmx64` (64-bit version of FileMaker). In macOS, all plug-ins will have an extension of `.fmplugin`.

The script should check for errors to ensure that the installation process was successful. For example, the `Allow Solutions to Install Files` application preference *must* be enabled for the script to work (Chapter 2, “Application Preferences”) and will return an error if it is disabled. Also, it is a good idea to use the `Get (InstalledFMPlugins)` function to confirm that the correct version of the plug-in is installed and enabled.

Accessing Plug-In Functions

Once a plug-in is successfully installed on a computer, it will be available from within the `Functions` pane of any `Specify Calculation` dialog under the `External Functions` category, as shown in Figure 38-1. They are used in the same manner as built-in functions (Chapters 13-24).

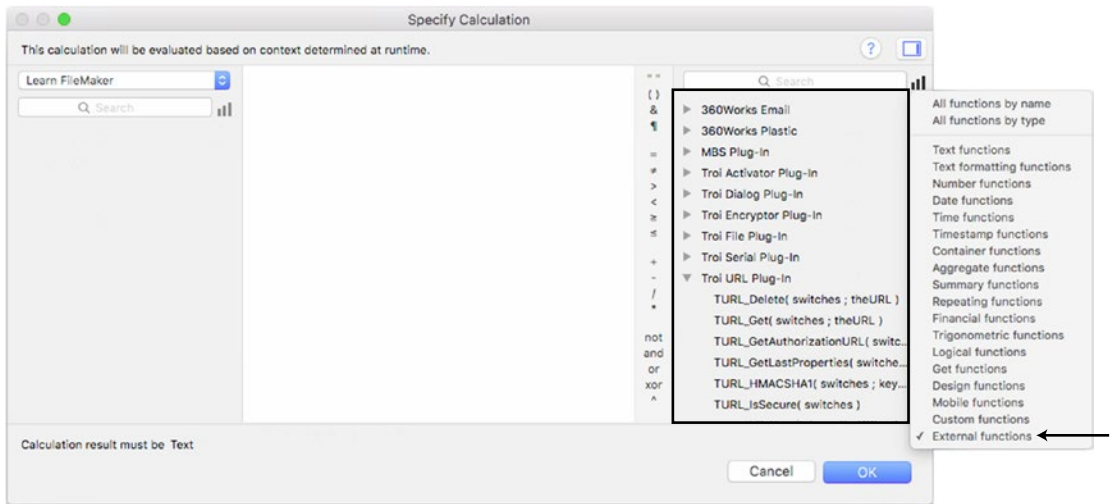


Figure 38-1. Plug-in functions are accessible under the External functions category of the Functions pane in the Specify Calculation dialog

Starting in FileMaker 16, plug-ins can add new script steps that are accessible through the Script Workspace.

Summary

In this chapter, we discussed the basic process for finding, installing, and accessing plug-ins.

PART VI



Sharing and Networking

Sharing databases on networks increases their usefulness but also introduces a need for security. These chapters cover the basics of network sharing, security, and some advanced analysis tools, and how to connect with the author and find more help online.

39. *Introducing Network Sharing* — Covers how to host a database on a network for sharing in a multi-user environment.
40. *Securing Database Access* — Discusses the options for configuring security settings to control who has access to which features.
41. *Advanced Sharing and Analysis Tools* — Explores the options for analyzing a database's structure with the Database Design Report and the utilities available for manipulating files and creating runtime solutions.
42. *Getting Connected* — A compilation of URLs from throughout the book, various FileMaker-related resources, and the official sites for the author and book.

CHAPTER 39



Introducing Network Sharing

FileMaker databases are not only great tools for individuals, they can be shared across a network. Network sharing allows multiple users to gain simultaneous access to one or more databases. Each user can view records, perform data entry tasks, and even perform development tasks simultaneously. Networking databases opens a new dimension for how teams collaborate and communicate.

In this chapter, we explore network sharing and discuss topics such as:

- How network sharing works
- Setting up peer-to-peer database sharing
- Using a FileMaker server
- Accessing databases from outside filemaker

How Network Sharing Works

Sharing a FileMaker database on a network is *not* the same as sharing a document file. Certainly, a database file *can* be placed in a shared directory on a file server where multiple people can see and access it. However, not only is this not what is meant by database network sharing, it is *not recommended*.

A database file sitting in a server directory can be opened by more than one user at the same time, as illustrated in Figure 39-1. Each copy of FileMaker that opens the database will perform read and write operations to the file continuously as the users perform data entry tasks, which will cause corruption and data loss. If this happens, the file must be run through the Recovery process and may be damaged beyond repair (Chapter 6, “Understanding File Damage”).

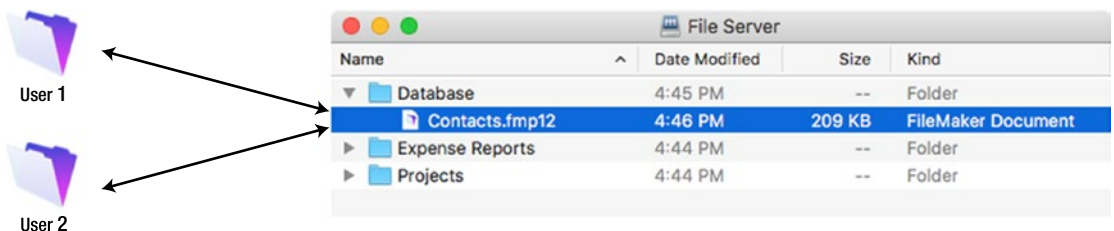


Figure 39-1. A database file sitting in a shared server directory, accessed by multiple people simultaneously will result in corruption and data loss

The correct way to share a database is to have it open and hosted by a version of FileMaker software that allows sharing. This can be done using peer-to-peer sharing with either FileMaker Pro or FileMaker Pro Advanced, or the preferred method of using client-server sharing with FileMaker Server.

When hosted in this manner, the physical file is installed somewhere on the host computer in a non-shared folder, out of site from all users. The application, acting as a network host, makes the file available to other users across the network while managing all read/write activity to the physical file itself, as shown in Figure 39-2.

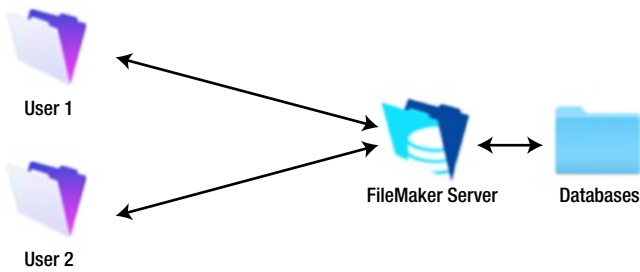


Figure 39-2. A database file hosted by a copy of FileMaker Pro running on a computer where the file is installed but visible to the network. Users connect to the host application instead of directly to the file

When configured correctly, a user accesses a shared database through the Hosts tab of the Launch Center, as shown in Figure 39-3, instead of through an Open File dialog as they would to open a database file stored locally on their computer.

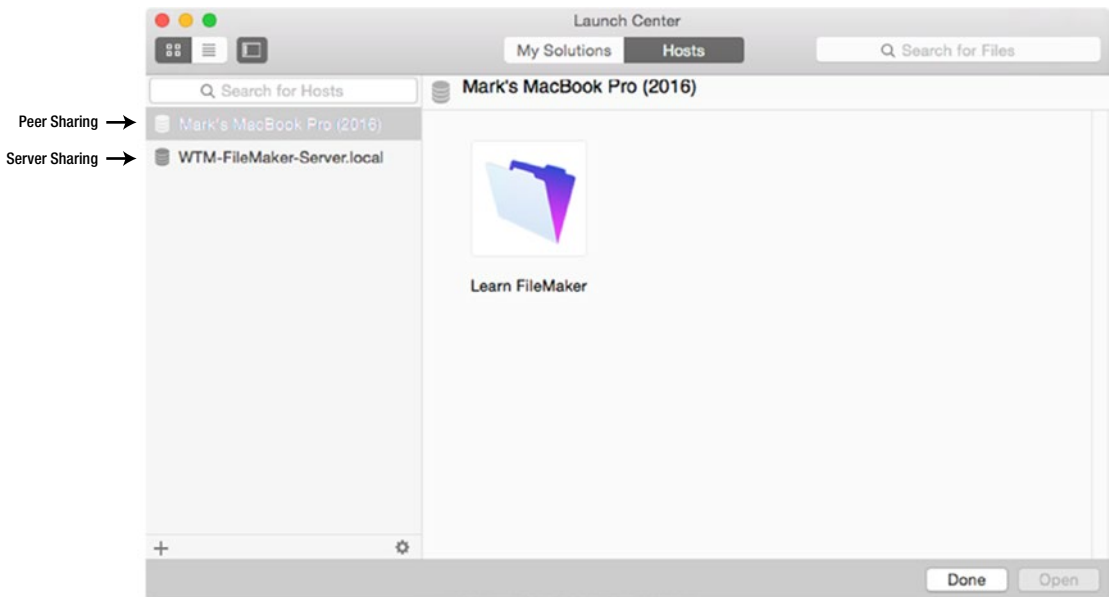


Figure 39-3. The Launch Center showing databases shared by a peer and a server

To open a database, select the appropriate host on the left and the desired file from the list on the right. Then click Open or double-click on the file icon.

Limitations to Sharing

There are some limitations enforced by FileMaker to avoid conflicts between multiple users when working in a hosted file. The following actions are limited to a single user at a time:

- Editing the *same* record.
- Modifying the *same* layout.
- Modifying the *same* script.
- Modifying *any* database scheme; tables, fields, relationships.
- Modifying *any* value list.
- Modifying *any* data source.
- Modifying *any* security access privileges.

The first three actions listed above can be undertaken at the same time if the users are each modifying a *different* record, layout, or script. The remaining development actions are limited to a single user at a given time working on *anything* within a specific developer dialog.

Setting Up Peer-to-Peer Database Sharing

The *peer-to-peer database sharing* option involves one user of a database acting as a host and sharing it with another user across a network. The two are considered peers because they are both using a version of the client software, either FileMaker Pro or FileMaker Pro Advanced. One peer is *acting as a host* even though their computer is a user workstation and not running any special server software.

Any computer running a database in the client software can act as a peer host. The host is limited to 125 databases (depending on various hardware factors) and a maximum of *five* concurrent remote client connections. Since the peer host is running a client version of the software, both the host and connected users can perform data entry and development tasks in the files, depending on their access privileges.

To configure peer-to-peer sharing, open the FileMaker Network Settings dialog, as shown in Figure 39-4. This dialog can be opened using any of these methods:

- Select the File > Sharing > Share with FileMaker Clients menu item.
- Click on the Sharing button in the toolbar and choose the Share with FileMaker Clients option.
- Run a script using the Open Sharing script step.

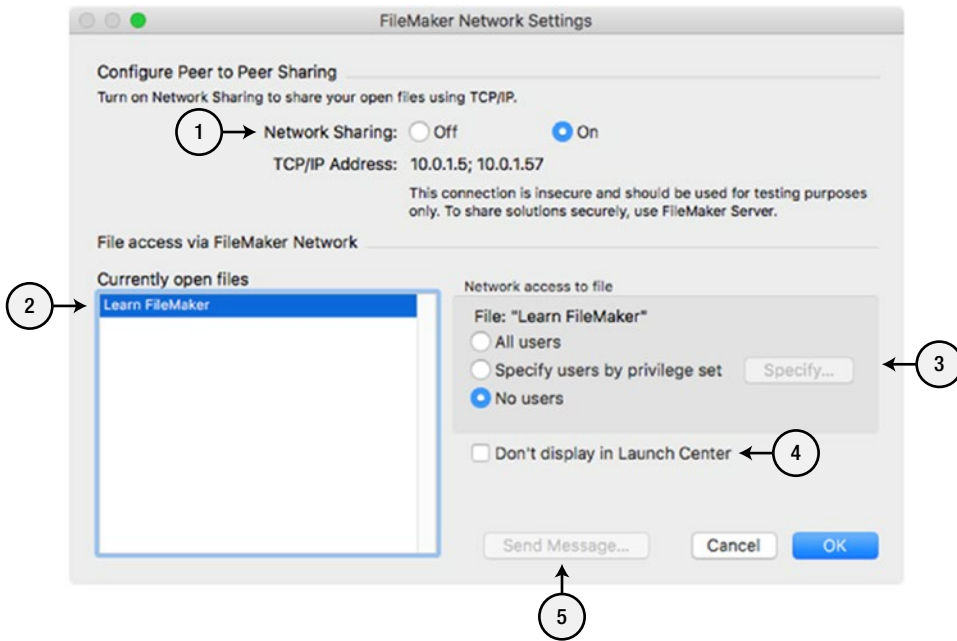


Figure 39-4. The FileMaker Network Settings dialog for configuring peer-to-peer sharing

The following controls are available on the FileMaker Network Settings dialog:

1. Network Sharing — These radio buttons are used to turn the FileMaker application’s peer-to-peer sharing on or off.
2. Currently open files — This area lists all the files open with the user’s copy of FileMaker. Select a file to configure its access settings in the adjacent panel.
3. Network access to file — This area contains the following radio buttons that control which, if any, users can connect to the selected file remotely:
 - All Users — Choose this option to allow any user to connect to the file. The user will still be required to enter a valid username and password if security is enabled on that database.
 - Specify users by privilege set — Choose this option to select the privilege set(s) to which a username and password must belong to connect to the file. When first selected, a dialog will open with a list of available privilege sets. It can be reopened later by clicking the adjacent Specify button.
 - No users — Choose this option to deny all access to the selected file, that is, the file will not be shared.

4. **Don't display in Launch Center** — Enabling this check box will cause the selected shared file to not be publicly displayed in the Launch Center for remote users. This means that any user will need to know the name and address of the file to access it or can only access it indirectly by opening another database containing a relationship that causes the file to open automatically.
5. **Send Message** — This button opens a dialog allowing a message to be entered and sent to all remote users of the selected database. This is helpful in the event the host needs to close a file and warn users who are actively connected.

Once the database is configured, click OK to save the changes. From then on, anytime the user opens the database on their local computer, it will be available to remote users across the network.

Using a FileMaker Server

FileMaker Server is a fast, reliable, and scalable server software package that can securely share database files across a network. The server provides many powerful features and benefits including:

- Reliable, full-time availability of databases to users of any of the FileMaker product line running on macOS, Windows, or iOS.
- Multiple, flexible licensing options that can be mixed and matched to suit the needs of your individual team members.
- Share databases on the Web without web programming using FileMaker WebDirect or use Custom Web Publishing to create data-driven websites with PHP or XML.
- Use ODBC (open database connectivity) and JDBC (java database connectivity) to read from and write to hosted databases with up to 50 simultaneous remote connections.
- Robust scalability without user limits except those imposed by the hardware, operating system, or licensing program.
- Monitor and manage the server locally or remotely using either the Admin Console through the web interface or the `fmsadmin` command line.
- Automate administration tasks:
 - Backups can run automatically, even when data is being accessed by users.
 - Schedule database scripts and server scripts to automatically perform routine and repetitive functionality.
- Database scripts can offload tasks from a user's computer to the server using the Perform Script On Server step (Chapter 34) to save processing time and allow users increased performance and efficiency.

Understanding Client License Options and Limitations

FileMaker offers three different options for how users connect to a FileMaker Server. These options may seem unnecessarily complex at first, however, they are designed to offer flexibility based on how individual team members access the server and can help to avoid unnecessary costs associated with a traditional approach.

The three options, which can be used in any combination, are:

- Legacy license
- User connection license
- Non-licensed

Legacy License

The *legacy license* is a traditional, computer-centric option where a copy of the client application is purchased for each computer upon which it will be installed. This license type applies *only* to copies of FileMaker Pro and FileMaker Pro Advanced.

Once purchased and installed, the software can be used to work on databases locally or to connect to databases hosted on a server. When connecting to a FileMaker Server, the number of legacy connections is theoretically unlimited, except for any limitations imposed by hardware specifications, database design, or the operating system of the host computer. However, this has only been officially tested up to 250 connections.

This license type is best suited for users who may work with databases locally outside of an office network without access to a server or for those who just prefer the traditional licensing model. However, using the legacy license, a user might end up purchasing multiple licenses if they typically divide the work between an office workstation, a laptop, and a home computer. Each of these computers would require an individual license.

For more information about legacy licensing or to make a purchase, visit the FileMaker store: <http://store.filemaker.com/legacy>.

User Connection License

The *user connection license* is a new, more user-centric option, in which a FileMaker Server is required and special licenses are purchased for each concurrent user connection. FileMaker calls this approach *FileMaker Licensing for Teams* and offers it as a simple way for teams of five or more people to license software.

Using this option, each user requires a user connection license and can then freely connect to hosted databases using any software including FileMaker Go, FileMaker WebDirect, or FileMaker Pro (for User Connections) license. All this software is “free” and can be installed on all the users’ devices since it requires a connection license on the server to access a hosted database or, in the case of the desktop application, to run locally. The user software can be installed on more computers than the number of user connection licenses purchased; however, the number of concurrent users cannot exceed the number of licenses purchased.

FileMaker Server includes one user connection license for testing purposes. FileMaker limits the number of user connections purchasable on their web store to 100. However, additional licenses can be purchased by contacting a sales representative.

Copies of the client desktop applications purchased under the legacy license can be used in conjunction with user connections and do not count toward the available user connections on the server.

For more information about legacy licensing or to make a purchase, contact a sales representative or visit the FileMaker store: <http://store.filemaker.com/teams>.

■ **Note** When using more than seven concurrent webDirect users, FileMaker recommends a second machine for web publishing, separate from the FileMaker Server. Also, the RAM and CPU requirements for the web server increase with each block of additional users. See “Recommended Hardware Configurations for FileMaker WebDirect” available at http://help.filemaker.com/app/answers/detail/a_id/15663.

Non-Licensed Connections

When creating your own client-side connections, there are no licensing requirements besides the purchase of a FileMaker Server.

Custom Web Publishing has a theoretical limit of 2000 concurrent connections but has only been officially tested to 200. ODBC/JDBC clients are theoretically unlimited, but have been tested to 50 concurrent connections and may be limited by hardware and operating system.

Exploring FileMaker Server Basics

The FileMaker Server software is far too complex to cover thoroughly. However, since databases are usually shared using the product, it is worth reviewing some basics, such as:

- Optimizing for server performance
- Installing the server software
- Defining installed resources
- Deploying a server
- Accessing the admin console
- Exploring the admin console
- Uploading files to the server
- Opening and closing files
- Restarting a FileMaker Server
- Using `fmsadmin` in the command-line interface

■ **Note** For more detailed information than the basics described in this section, consult the FileMaker Server product support section located at: <https://www.filemaker.com/support/product/documentation.html>.

Optimizing for Server Performance

There are several recommendations for the host computer that may improve the performance of a FileMaker Server including:

- Use the most powerful computer you can afford with a fast processor and plenty of RAM. Also, a modern, solid state drive can greatly improve the read/write speeds.
- Connect the computer to your network with a wired, high speed, dedicated connection.
- Assign a static IP address to the server computer if possible.
- To allow for the high levels of processor, hard disk, and network capacity, the software should be installed on a *dedicated* server computer used exclusively for hosting databases. It is *strongly recommended* that it not be installed on a user's primary workstation or any other type of server including file, email, or Web.

- To avoid corruption, the FileMaker Server folder should *not* be backed up with operating system or third-party backup software, including the macOS Time Machine. Instead, use FileMaker Server’s built-in scheduled backup feature, ideally to a separate hard drive.
- Disable all screen savers and energy-saving hibernation or sleep features. Instead, turning off or diming the monitor is fine.
- Either turn off indexing services such as the macOS Spotlight feature or exclude the entire FileMaker Server folder.
- Exclude the entire FileMaker Server folder from anti-virus scans.

Installing the Software

FileMaker Server is installed in much the same way as a version of the desktop application. On macOS, this involves open a standard installer disk image file and running the installer (Chapter 2, “Installing and Launching FileMaker Pro”). Before installing, it is recommended that you disable anti-virus scanning.

The installer has the following three installation options:

- **Single Machine** — use this option for most installations; all the server components will be installed on one master computer.
- **Multiple Machine** — use this option to deploy server components across multiple machines to improve overall performance in environments where the server will be used frequently by clients with WebDirect, XML, or PHP.
- **Uninstall** — Use this option to remove all FileMaker Server resources. A computer can have only one version of the server software installed at a given time so you must uninstall past versions before installing a new one. This process will remove server resources only and not any of your databases.

Defining Installed Resources

FileMaker Server isn’t a traditional application and will not appear in your /Applications folder. Instead, the following resources will be installed:

- *Database Server* — This software hosts the databases, shares them with FileMaker Pro and FileMaker Go clients, and publishes them on the Web.
- *Web Publishing Engine* — This software handles the Custom Web Publishing and WebDirect services by managing traffic between the Web Server and the Database Server.
- *Web Server* — In macOS, the server uses its own copy of the Apache web server, which is automatically installed and enabled. In Windows, the Internet Information Services is enabled during installation and used as the web server.
- *PHP Engine* — This software is used by Custom Web Publishing with PHP to route calls for the FileMaker API to the Web Publishing Engine.
- *Admin Console* — This software runs in a web browser on the server or any client computer and is used to configure and administer the server.

- *Command-Line Interface Executable* — The `fmsadmin` commands are used to administer the FileMaker Server through a command-line interface (described later in this chapter).
- *User Account* — A user account under which FileMaker Server will run must be specified. The default choice on macOS is a new `fmserver` user created by the installer; on Windows, the `Local System`. However, you can specify a custom account to use instead if it provides adequate access to the computer services.
- *User Group* — A new user group called `fmsadmin` is created and must be assigned to databases to allow them to be hosted by the server.

Deploying a Server

After the installation is complete, you have the option to open the `Deployment Assistant`, a seven-step dialog that will walk you through entering basic information about the deployment, including:

- Entering a user name, password, and PIN for accessing the admin console.
- Entering a server name, description, and owner information.
- Choosing to enable ODBC/JDBC and Web Publishing.

Once finished, the server will be configured and the `Admin Console` will be opened.

Accessing the Admin Console

The `Admin Console` is used to administer the server and is accessible through a web browser from the server computer and any other computer with access to it from across the network.

■ **Note** If you opted for it during the install process, a `.webloc` file containing a bookmark to the local host will be found on the server computer's desktop.

To access the console from the server computer, open a web browser and enter:

```
http://localhost:16001/admin-console
```

To access the console from another computer, open a web browser and enter either one:

```
https://<ServerAddress>:16000/admin-console
https://<ServerAddress>/admin-console
```

For example, if the server is running on a computer with an address of `10.0.1.20`, it can be accessed with one of these addresses:

```
https://10.0.1.20:16000/admin-console
https://10.0.1.20/admin-console
```

In either case, this should open a login page, as shown in Figure 39-5.



Figure 39-5. The FileMaker Server login page

Enter the administrator name and password you defined during deployment to log in and access the Admin Console.

Exploring the Admin Console

The *Admin Console* organizes server administration controls in seven screens, each accessible in a list on the left side of the screen:

- *Status* — Contains a summary of the server information, status, logs, and a control to turn the server on or off.
- *Activity* — Contains two tabs:
 - *Databases* — Shows a list of all installed databases and their status. Click on a database to see the clients actively connected. Also contains controls to take actions on a database such as open, close, pause, resume, etc.
 - *Clients* — Shows a list of all active clients. Click on a client to see a list of databases in use.
- *Schedules* — Contains a list of all schedules and controls for managing them.
- *Statistics* — Contains information about server usage and client activity. Includes a graph with your choice of eleven different data points.
- *Log Viewer* — Contains a detailed log of events with filter controls.
- *General Settings* — Contains various settings divided into seven tabs:
 - Server Information
 - Email Notifications

- Admin Console
- Startup
- ODBC/JDBC
- Administrator Groups
- Connections
- *Database Server* — Contains various server settings divided into seven tabs:
 - FileMaker Clients
 - Databases
 - Security
 - Folders
 - Logging
 - Server Plug-Ins
 - Directory Services

Uploading Files to the Server

There are two methods available for adding a database to the server:

- Moving the file into the databases folder
- Uploading a file from a client computer

■ **Note** To avoid a security risk, FileMaker requires that a database file has an account and password assigned before uploading to a server.

Moving the File into the Databases Folder

The traditional method of installing files onto a FileMaker Server is to copy them to the host computer and then move them into the `/Data/Databases/` subfolder of the FileMaker Server folder. If you chose the default installation location, this folder will be in one of two locations depending on the operating system.

On macOS, the folder can be found at:

```
/Library/FileMaker Server/Data/Databases
```

On Windows, the folder can be found at:

```
[drive]:\Program Files\FileMaker\FileMaker Server\Data\Databases\
```

To install files, copy them into the folder. You can create subfolders if desired and place files within them. It is important that the files be assigned to the `fmsserver` user and `fmsadmin` group.

■ **Caution** Never open files located in the Databases folder directly with a client version of the software by double-clicking them or choosing them in an Open File dialog, especially when they are being actively hosted on the network. To be safe, files should first be closed and then removed from the folder before any direct access with a client version of FileMaker.

Uploading a File from a Client Computer

An open FileMaker file can be uploaded to a server from anywhere on the network from within FileMaker Pro or FileMaker Pro Advanced.

Selecting and Signing into a Server

To begin, open the Upload to FileMaker Server dialog, as shown in Figure 39-6. This dialog can be opened by either of the following two methods:

- Select the Sharing > Upload to FileMaker Server menu item.
- Click on the Sharing icon in the toolbar of a database window and choose Upload to FileMaker Server from the menu.

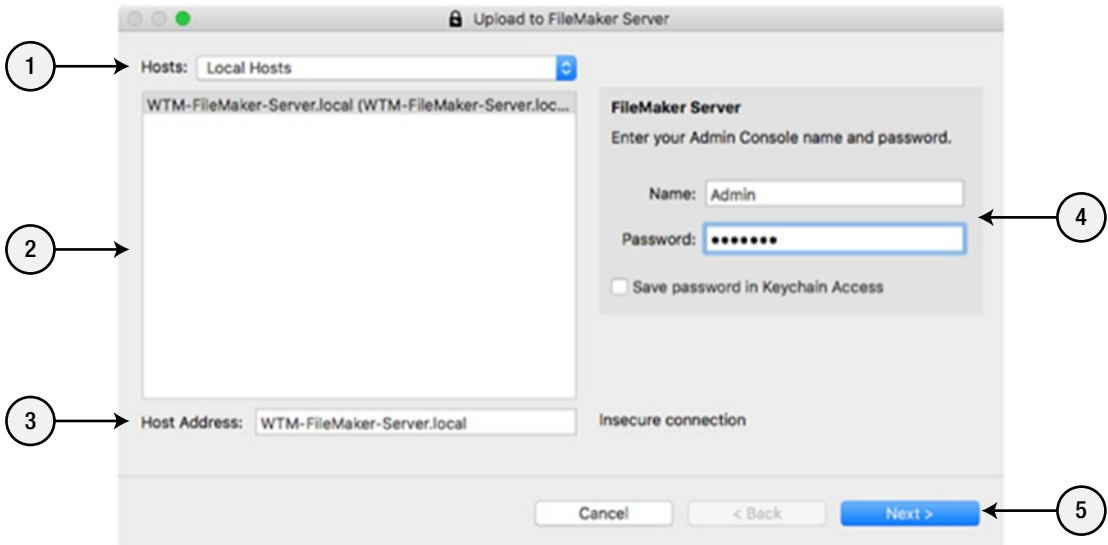


Figure 39-6. The Upload to Filemaker Server dialog is used to copy an open database to a FileMaker Server

If you have a database file open, a dialog will inform you that it will be closed before continuing.

This dialog allows you to select a FileMaker Server and sign on as an administrator using the following controls:

1. *Hosts* — This pop-up menu contains the following three options for selecting a host category:
 - *Local Hosts* — Choose this option to automatically search your local network for FileMaker Servers.
 - *Favorite Hosts* — Choose this option to display a list of FileMaker Servers that you have entered and made favorites in the Launch Center (Chapter 2).
 - *Hosts Listed by LDAP* — Choose this option to display a list of FileMaker Servers from a lightweight directory access protocol server.
2. *List* — This area lists all the available FileMaker Servers from the selected host category.
3. *Host Address* — This area will display the selected or manually entered FileMaker Server's address.
4. *Access Settings* — Enter the Name and Password for the selected server's admin console.
5. *Next* — Click this button to continue to the next step.

Selecting Files to Upload

The second panel of the Upload to FileMaker Server dialog, shown in Figure 39-7, is used to specify the database file(s) to upload and other options.

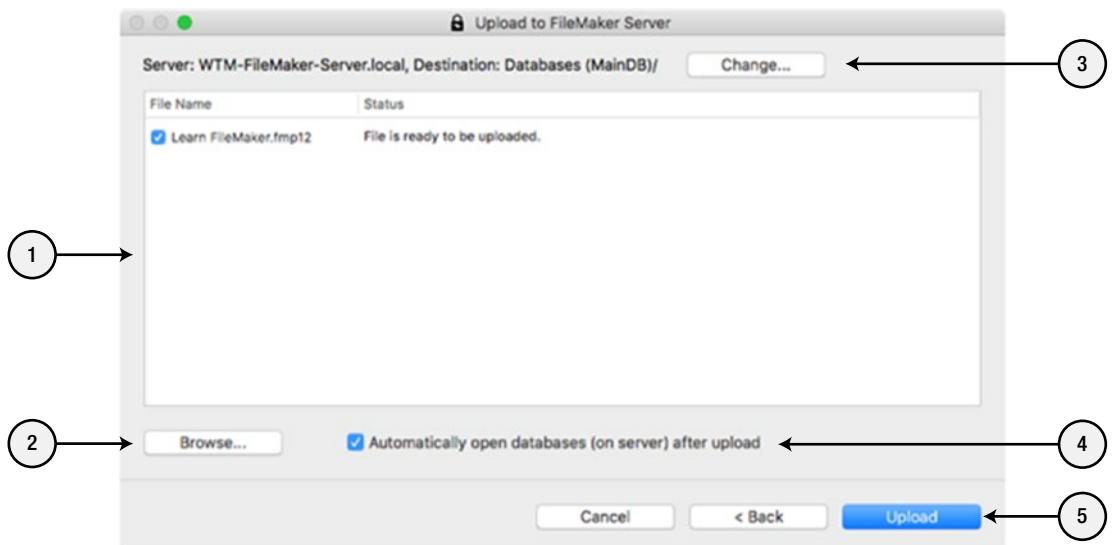


Figure 39-7. The second panel of the Upload to FileMaker Server dialog allows specification of which database file(s) to upload

This dialog contains the following controls:

1. *Databases* — This area contains a list of databases to upload. If this process was started from an open database, it will automatically appear in the list. The check box indicates the file(s) that will be uploaded and the status indicates any issues.
2. *Browse* — This button will open a standard Select File dialog allowing the selection of one or more database files to add to the list.
3. *Change* — This button will open a Change Folder dialog, shown in Figure 39-8, in which you can select or create a database subfolder for the uploaded files.

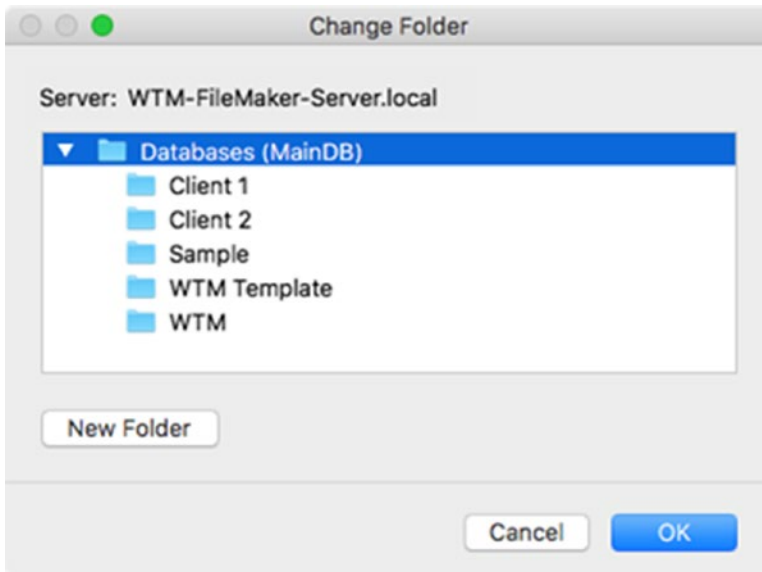


Figure 39-8. The Change Folder dialog is used to select or create a database subfolder for the files being uploaded to the server

4. *Automatically open databases* — Enable this check box to automatically open and host the databases on the server after the upload.
5. *Upload* — Click this button to continue the process of uploading the selected files into the desired folder on the server.

Managing Database Files

The Admin Console provides many different tools for interacting with databases. See Figure 39-9.

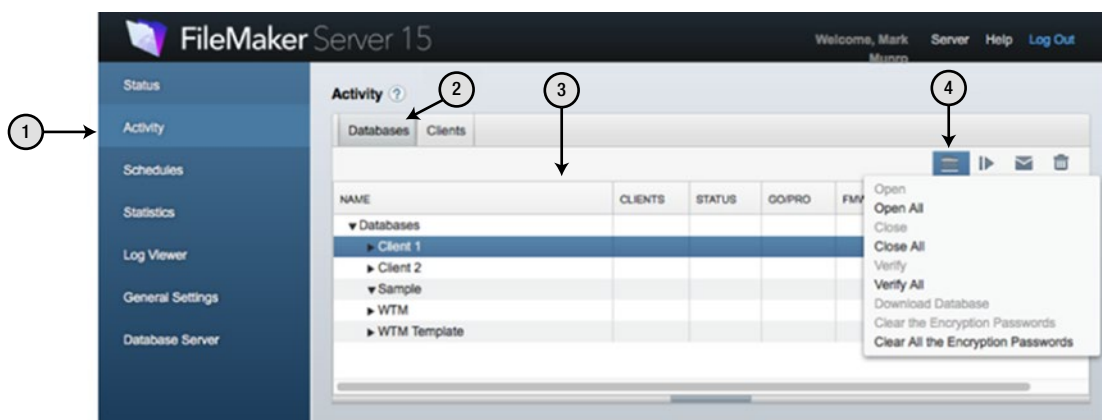


Figure 39-9. The Manage Database menu of the Database tab of the Activity panel of the console

To access these tools, log into the Admin Console and perform the following actions:

1. Click **Activity** in the panel on the left side of the window.
2. Click on the **Databases** tab.
3. Select one or more databases or folders.
4. Click on the folder icon on the upper right of the Databases tab to reveal a drop-down menu.

The following commands are available in this menu:

- **Open** — This command will open and begin hosting the selected database file(s).
- **Open All** — This command will open and begin hosting all files regardless of the selection.
- **Close** — This command will close and cease hosting the selected database file(s).
- **Close All** — This command will close and cease hosting all files regardless of the selection.
- **Verify** — This command will close, verify, and reopen the selected database file(s).
- **Verify All** — This command will close, verify, and reopen all files regardless of the selection.
- **Download Database** — This command will download a copy of the selected closed database file(s) to the local hard drive. In macOS, this feature will not work if pop-up windows are blocked by the browser preferences!

- Clear the Encryption Passwords — This command will remove the encryption password from the selected database file(s).
- Clear All Encryption Passwords — This command will remove encryption passwords from all databases regardless of the selection.

Removing Files from the Server

Before moving files out of the FileMaker Server folder, it is important to first close them through the Admin Console to avoid file corruption. To help avoid errors, FileMaker has a `Remove` command. This command is the only option available in the menu accessed through the trash icon, shown in Figure 39-10. The menu item is only enabled when one or more *closed* database files are selected.



Figure 39-10. The `Remove` option is used to safely remove files from the list of available databases and place them into a subfolder of the `Databases` folder for moving and copying elsewhere

Once selected, the console will present a dialog confirming that you want to continue moving the file out of the server. If you click OK in this dialog, the file(s) will disappear from the list of available databases. The file(s) will be placed inside of a new `Removed_by_FMS` subfolder within the `Databases` folder, which will not be visible from within the Console. It is now safe to move or copy the file(s) out to another directory on your server or elsewhere on the network.

Restarting a FileMaker Server

Understanding the proper shutdown procedure is critical to avoid file corruption. When the server is actively hosting files, it may be communicating with clients, holding recent changes in cache, or be in the middle of writing those changes to disk. Therefore, it is *vitally important* that the FileMaker Server is properly shut down prior to any hardware restart or shutdown.

Unsafe Methods of Shutting Down a Server

The following methods should be *avoided* since they can cause *major file corruption*:

- Using the operating system's Shut Down or Restart commands.
- In macOS, using the Activity Monitor app to Force Quit the server process (`fmserverd`).

- A forced hardware reboot by any of the following methods:
 - On Windows, pressing the power on/off button.
 - On macOS, pressing and holding the power button.
 - Running the server without a UPS (uninterruptable power supply) during a power outage resulting in an abrupt hardware shutdown.

If any of these actions occur, it will most likely require a Recover process that may still result in some loss of data. In a worst-case scenario, the file might be irreparable.

Safe Methods of Shutting Down a Server

In addition to using a UPS device that can power the computer for an adequate length of time to allow a proper manual or scripted shutdown during a power outage, the following methods are considered safe to prepare a server for restart or shutdown:

- Logging into the Admin Console and shutting off the database server, with or without first closing all database files.
- On macOS, using the Activity Monitor app to Quit the server process (`fmservd`).
- On Windows, using the Administrative Tools > Services app to Restart or Stop the FileMaker Server service.

■ **Note** Back in FileMaker Server 5.5, there was a problem that would corrupt files if you shut down the server software without first closing all databases. Due to this, as a precaution, some developers still follow that procedure.

Using `fmsadmin` in the Command-Line Interface

FileMaker Server installs the `fmsadmin` tool for administering the FileMaker Server through the command-line interface using the Terminal app (macOS) or the command prompt (Windows). These commands can be used in scripts or batch files.

Locating the `fmsadmin` Executable

When the server is installed in the default location, the executable `fmsadmin` is in the following platform-specific folder:

- macOS: `/Library/FileMaker Server/Database Server/bin/fmsadmin`
- macOS (symbolic link): `/usr/bin/fmsadmin`
- Windows: `[drive]:\Program Files\FileMaker\FileMaker Server\Database Server\fmsadmin.exe`

Formatting an `fmsadmin` Command

The formula for an `fmsadmin` command is:

```
fmsadmin <command> <options>
```

For example, to get a list of all hosted databases:

```
fmsadmin list files
```

Certain functions require authentication with the admin username and password. These prompts will immediately follow once a command is entered. Optionally, you can include the administrator name and password as follows:

```
fmsadmin <command> -u <user> -p <password>
```

For example, to get a list of all hosted databases in a single command line, try:

```
fmsadmin list files -u Admin -p J56TF3
```

Available `fmsadmin` Commands

The `fmsadmin` tool contains the following twenty-three commands, each with various options available:

- `autorestart` — Get or set the auto-restart feature of the `adminserver`:
- `backup` — Back up one database or every database in a folder.
- `clearkey` — Remove saved database encryption passwords.
- `close` — Close one or more databases.
- `certificate` — Manage SSL certificates.
- `disable` — Disable schedules or statistics logging.
- `disconnect` — Disconnect one or more clients.
- `enable` — Enable schedules or statistics logging.
- `help` — Get help regarding a specific command or get a list of available commands.
- `list` — List clients, databases, plug-ins, or schedules.
- `open` — Open one or more databases.
- `pause` — Temporarily stop the database server.
- `remove` — Move databases out of hosted folder or remove empty folder.
- `resetpw` — Reset the admin user password.
- `restart` — Restart the server, `adminserver`, `FMSE`, `WPE`, or `XDBC` process.
- `resume` — End a temporary pause, making databases available again.
- `run` — Run a schedule.
- `send` — Send a message to connected users.

- `standby` — Manage standby server connections, roles, and updates.
- `start` — Start the server, `adminserver`, FMSE, WPE, or XDBC process.
- `status` — Get the status of clients or databases.
- `stop` — Stop the server, `adminserver`, FMSE, WPE, or XDBC process.
- `verify` — Check the consistency of databases.

Getting Detailed `fmsadmin` Command Help

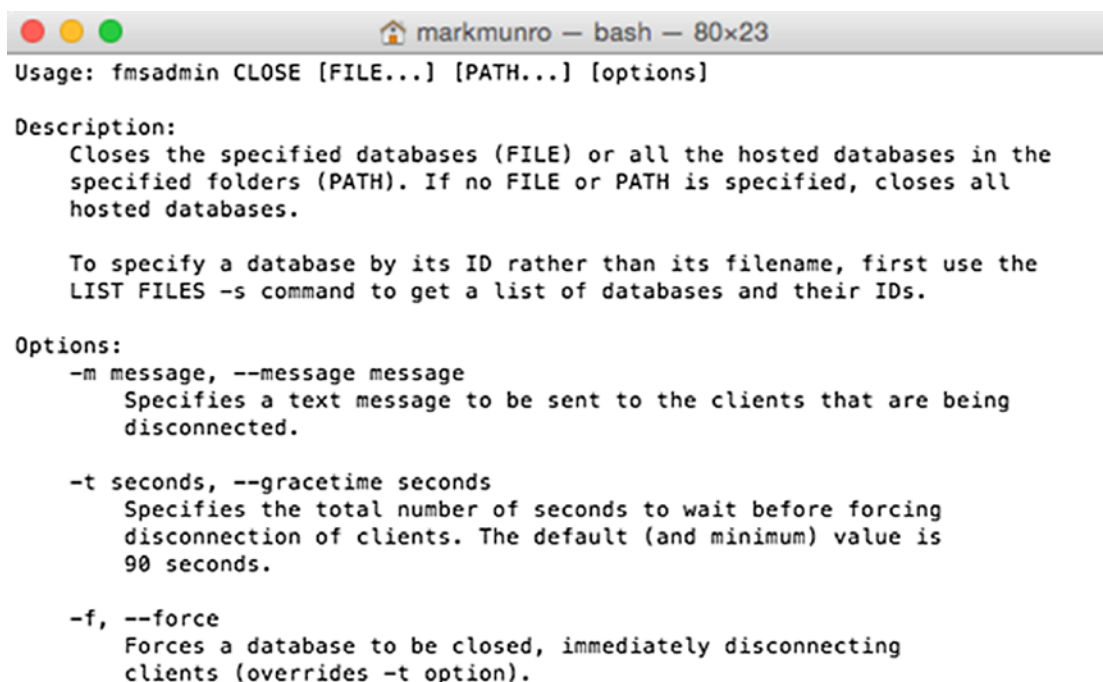
To get additional information about a specific `fmsadmin` command, use the `help` command followed by the name of the desired command:

```
fmsadmin help <commandName>
```

For example, to explore the `close` command, type:

```
fmsadmin help close
```

The results will include a description of the command and details about the available options, as shown in Figure 39-11.



```

markmunro — bash — 80x23
Usage: fmsadmin CLOSE [FILE...] [PATH...] [options]

Description:
  Closes the specified databases (FILE) or all the hosted databases in the
  specified folders (PATH). If no FILE or PATH is specified, closes all
  hosted databases.

  To specify a database by its ID rather than its filename, first use the
  LIST FILES -s command to get a list of databases and their IDs.

Options:
  -m message, --message message
    Specifies a text message to be sent to the clients that are being
    disconnected.

  -t seconds, --gracetime seconds
    Specifies the total number of seconds to wait before forcing
    disconnection of clients. The default (and minimum) value is
    90 seconds.

  -f, --force
    Forces a database to be closed, immediately disconnecting
    clients (overrides -t option).
```

Figure 39-11. The integrated help guide for the `Close` function

Hosting Databases with FileMaker Cloud

FileMaker Cloud is a new service that provide reliable access to databases from anywhere in the world. Databases are managed by the FileMaker Cloud web service hosted through Amazon Web Services (AWS), which essentially places a special version of FileMaker Server on a LINUX server in the cloud.

Some important differences between FileMaker Server and FileMaker Cloud:

- There are no hardware or energy costs associated with the cloud services. However, you must have an active AWS account. The FileMaker Cloud service will ask you for that information during installation.
- All interactions with the server must be performed through the AWS Management Console and the FileMaker Cloud Admin Console. There is no way to access the folder structure directly.
- Because there is no way to access the folder directory on the cloud server, the only method available to upload databases to the server is using the Upload to FileMaker Server option accessible in the Sharing menu or by clicking the Sharing tool in the toolbar of a database window.
- To protect your information, all databases must be encrypted when uploaded to the cloud. During the upload process, you will be prompted to enter an encryption password for any files that are not already encrypted.
- When enabled, the FileMaker Cloud service will perform various maintenance tasks such as daily backups, software upgrades, automatic restarts during non-business hours, and other maintenance tasks.
- Since AWS manages snapshot backups of your files, there is no script schedule interface although this has been promised in a future release.
- AWS is responsible for the physical security of the server hardware.
- ODBC and JDBC are both supported and drivers are provided by the service.
- Custom Web Publishing with PHP and XML is not supported.

■ **Note** For help setting up a FileMaker Cloud server, check out this detailed, step-by-step guide on the beeswax.net blog: <https://goo.gl/8tBKBN>.

Connecting to the Cloud Server

Once FileMaker Cloud is installed on your AWS account, you can connect to the Admin Console or access databases using the following URL formula:

<https://<HostName>.fmi.filemaker-cloud.com>

For example, if the host name you assigned the server is widgetdatabases, then the URL would be:

<https://widgetdatabases.fmi.filemaker-cloud.com>

Accessing Databases from Outside FileMaker

There are a few interesting methods available to access databases from outside of FileMaker. These include:

- Sharing bookmarks with snapshot links
- Using the FileMaker URL
- Using AppleScript to automate FileMaker

■ **Note** These methods of access don't necessarily *require* a FileMaker Server to work and can work with databases open locally on a user's computer. However, to share these with other users, the databases must be shared either peer to peer or hosted on a server.

Sharing Bookmarks with Snapshot Links

A *snapshot link* is an XML file that stores information about a record or a found set of records and is used to re-create a previously existing found set or single record. They act kind of like a bookmark to a found set of records. Once saved, double-clicking on the snapshot link file will automatically open the same database, navigate to the same the layout, and restore the found set of records, thereby re-creating the same context a user was viewing at the time they saved it. Snapshot links are a great way to exchange lists of records with other users without much effort. They can be emailed to and used by anyone who has access to the database across the network.

To generate a snapshot link, open a database and establish the desired found set. Then select the File > Save/Send Records As > Snapshot Link menu to open the Save Records As Snapshot Link dialog, as shown in Figure 39-12.

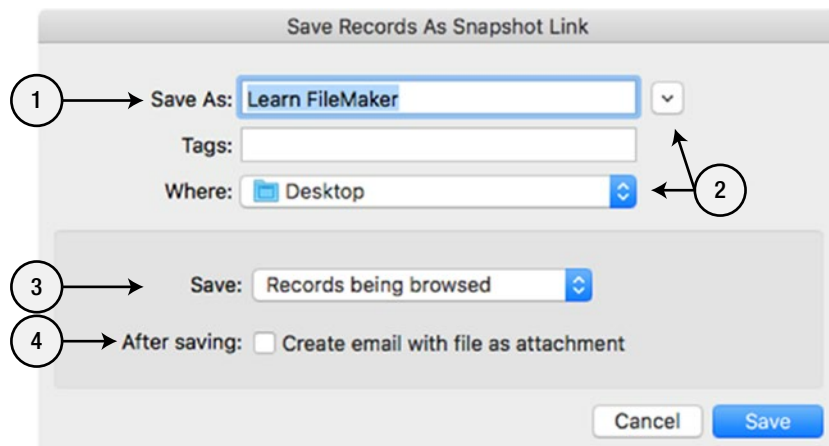


Figure 39-12. The dialog used to specify the location and settings when saving a snapshot link

The following options are available in the dialog:

1. **Save As** — Enter the name for the snapshot link file. The `.fmps1` extension will be automatically appended to the name you select.
2. **Where** — Select the folder location in which to save the file, either from the pop-up menu of standard folders or click to expand the dialog to navigate to another location.
3. **Save** — Select a save option of `Records being browsed` or `Current record`.
4. **After saving** — Enable this check box to automatically create an email with the snapshot link file as an attachment.

Using the FileMaker URL

The *FileMaker url*, an Internet protocol that is registered on your computer when you install FileMaker Pro, allows you to open a database and run scripts using a URL.

■ **Caution:** A user account must have explicit permission to use this feature. This is granted by the `fmurlscript` extended privilege (Chapter 40).

The URL can be used in many ways, including:

- *Manually typed into a web browser* — When typed into the macOS Safari browsers, the user will be presented a dialog confirming permission for the browser to open the database.
- *Included as an href in a web page* — Using the href tag to create a link with the FMP URL will allow a user to click to open the database. The web page can be accessed through a browser or from within FileMaker using a web viewer.
- *Used with the Open URL script step* — Embed the URL within a FileMaker script to automate access and allow dynamic triggering of other scripts without hard-coding script references.
- *Used in external applications and scripts* — Programming languages such as AppleScript can open databases and run scripts using the FMP protocol.

Formatting a Basic FMP URL

At minimum, the URL must include the FMP prefix and the name of the database, following this formula:

```
FMP://<databaseName>.fmp12
```

For example, to open a Contacts database, the address would be formatted like this:

```
FMP://Contacts.fmp12
```

Extensions Are Optional

The file extension can be included but is *not required* since FileMaker will only attempt to open databases with an extension appropriate to its capabilities.

Percent Encoding Required in Most Cases

It is important to remember that that, except when using the Open URL script step, all URLs must be properly percent encoded. For example, any spaces must be replaced with %20, as shown here:

```
FMP://Learn%20FileMaker.fmp12
```

■ **Note** If %20 doesn't work, try using a plus sign instead.

Including Optional Components

The FMP formula can be composed with any of the following optional components:

- Server address
- Account name and password
- Script name, parameter, and variables

Including a Server Address

When the URL includes only a database name, FileMaker will look for an open database on the local computer to activate. If the database named isn't found, it will silently fail. However, the real intention of the FMP protocol is to allow systems to dynamically access databases hosted on a network server. To do so, the address to a FileMaker Server host computer must be specified. This expands the formula to include both an address and a database name as shown here:

```
FMP://<serverAddress>/<databaseName>
```

For example, if the Learn FileMaker database were hosted on a server, the FMP URL might be:

```
FMP://10.1.0.10/Contacts.fmp12
```

The address portion of the URL can be an IPv4 address (as shown above) or it can be an IPv6 address or DNS name as shown in these example:

```
FMP://[2001:0db8:0a0b:12f0:0000:0000:0000:0001]/Contacts.fmp12
FMP://filemaker.local/Contacts.fmp12
```

Including Access Credentials

Since most networked databases will require an account name and password to gain access, unless these are specified in the URL, FileMaker will present a dialog requesting this information before opening the file. This may be fine for some circumstances where humans are accessing the file using the FMP protocol. However, in some cases, it might be an unnecessary annoyance. When used with a fully automated system, it would become catastrophic.

To completely bypass the login dialog, an account name and password can be added as colon delimited values ahead of the address separated by an @ sign, as shown here:

```
FMP://<account>:<password>@<serverAddress>/<databaseName>.fmp12
```

For example, if the account name is “admin” and the password is “58Jt-234,” then the following will open the database without a password prompt:

```
FMP://admin:58Jt-234@10.1.0.10/Contacts.fmp12
```

If the account name specified doesn’t exist or the password is incorrect, FileMaker will still display the login dialog and wait for accurate credentials to be entered before proceeding.

■ **Note** If a database is already open on the computer, FileMaker will ignore the credentials specified in the URL and activate the database.

Including a Script Name

Opening a database isn’t the only function of the URL; it can trigger a script within the database by adding a script name after the database name with a question mark between them:

```
FMP://<serverAddress>/<databaseName>?<scriptName>
```

For example, the following URL will run a script named Find All Active in a Contacts database on the specified host computer:

```
FMP://10.1.0.10/Contacts.fmp12?Find%20All%20Active
```

■ **Note** The script will be executed with the privileges and limitations of the account used to open the database, either through the URL or by a user.

Adding a Script Parameter

If the script accepts parameters, you can include an input value in the URL by adding it after the script name as shown in this formula:

```
FMP://<serverAddress>/<databaseName>?<scriptName>&param=<scriptParameter>
```

For example, the following URL will run a script named Find All in a Contacts database on the specified host computer and include a qualifying parameter of “Active”:

```
FMP://10.1.0.10/Contacts.fmp12?Find%20All&param=Active
```

Adding Script Variables

The URL can even initialize local variables within the script to provide additional values by adding an ampersand, the name, and value of the variables as shown here:

```
FMP://<serverAddress>/<database>?<script>&param=<Parameter>&<$variable>=<value>
```

This example shows how to initialize a variable specifying a state name for the find script:

```
FMP://10.1.0.10/Contacts.fmp12?Find%20All&param=Active&$State=Pennsylvania
```

Using AppleScript to Automate FileMaker

■ **Caution:** A user account must have explicit permission to use this feature. This is granted by the `fmextscriptaccess` extended privilege (Chapter 40).

AppleScript is an Open Scripting Architecture compliant command language that can communicate with and control Macintosh applications. Back in 1994 when the language was first introduced as a part of the Macintosh System 7.5, one of the first scriptable applications was FileMaker Pro. Since then, support for the language has steadily evolved and continues to provide a great method for accessing and manipulating data with scripts from outside of FileMaker.

Defining the Tell FileMaker Statement

AppleScripts are written in the Script Editor application that is in the `/Applications/Utilities/` folder. Commands are sent to an application by using a `Tell` statement that names the application and encloses the object references and command instructions. For FileMaker, the statement formula would be:

```
tell application "FileMaker Pro"
    <commands>
end tell
```

Opening FileMaker's Script Dictionary

As with any application, FileMaker's script dictionary can be opened by either of the following methods:

- Drag and Drop the application onto the Script Editor application icon
- Open the Script Editor and select the `File > Open Dictionary` menu item, which opens a dialog listing every application installed on the computer.

This will open a window displaying the dictionary of objects and commands available for controlling the application, as shown in Figure 39-13.

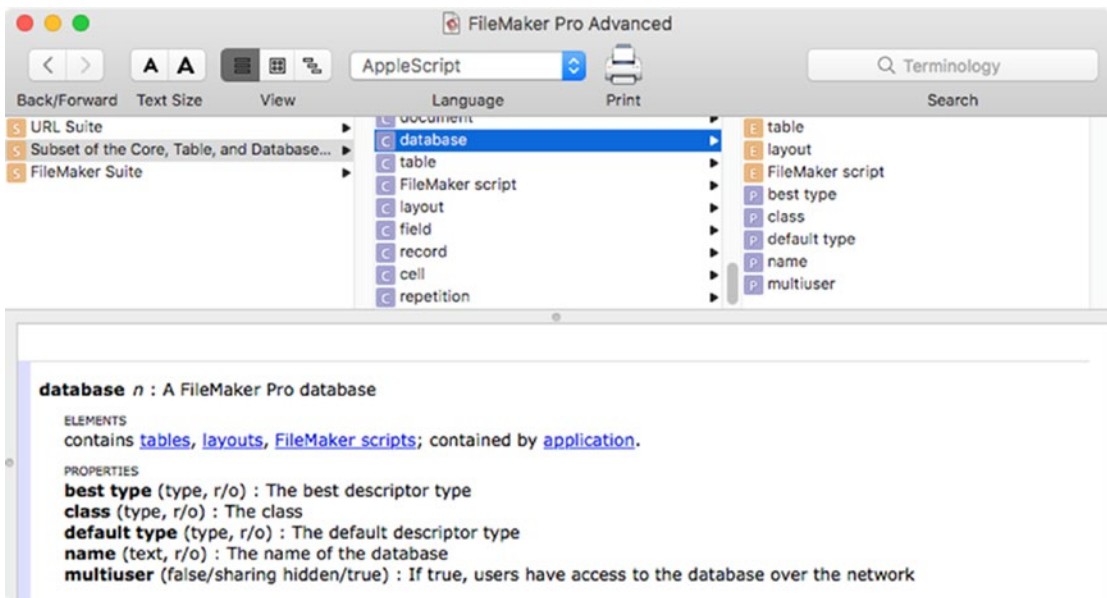


Figure 39-13. The FileMaker Pro Advanced dictionary window contains definitions of all the objects and commands available for controlling the application with AppleScript

■ **Caution** FileMaker’s script commands use three different words to describe a database window: database, document, and window. While somewhat confusing, each of these can sometimes be used interchangeably but may have a slightly different effect.

Scripting Basic Tasks

An entire book could be written containing details and examples of all the various commands that can be sent to FileMaker with AppleScript. In this section, we cover the following basic tasks to provide some examples:

- Opening a database
- Activating an open database
- Changing layouts
- Performing a find
- Counting record

■ **Note** For more general information on the AppleScript language, see my book *AppleScript: Developer Reference* (2010, Wiley) or visit the language guide on Apple’s website at: <https://goo.gl/i70lnx>.

Opening a Database

There are several methods available to open a database using AppleScript, including:

- From a local directory
- From a server host using the FMP protocol

From a Local Directory

Opening a database stored within the ~/User folder on macOS can be achieved using the `open file` command:

```
set pathToDatabase to "Macintosh HD:Users:shannonmiller:Learn FileMaker.fmp12"
set textPassword to "58Jt-234"
set textAccount to "Admin"

tell application "FileMaker Pro"
    open file (pathToDatabase) with passwords textPassword for Accounts textAccount
end tell
```

From a Server Using the FMP Protocol

To open a database hosted on a FileMaker Server, the `getURL` command will accept a FMP URL described earlier in this section. For example:

```
tell application "FileMaker Pro"
    getURL "fmp://Admin:58Jt-234@10.0.1.20/Learn%20FileMaker.fmp12"
end tell
```

■ **Tip** The beginning of this tip is trying to open a URL. Seems like a coding error???

Activating an Open Database

To activate FileMaker and bring an open database to the front of the document stack, this can be done by checking to see if the document exists and then using the `show document` command:

```
set nameDatabase to "Learn FileMaker"

tell application "FileMaker Pro"
    if (document nameDatabase exists) = true then
        show document nameDatabase
    end tell
```

Changing Layouts

To change the current layout in the window of an open database, use the `show layout` command. This example will switch to the Contact layout in an open Learn FileMaker database:

```
set nameDatabase to "Learn FileMaker"
set nameLayout to "Contact"

tell application "FileMaker Pro"
    tell document nameDatabase to show layout nameLayout
end tell
```

Performing a Find

An AppleScript can prompt FileMaker to search for records based on a variety of criteria.

Finding All Records

To perform the equivalent of FileMaker's Find All command with a script, use the `show every record` command within a `tell database` statement, as shown here:

```
set nameDatabase to "Learn FileMaker"
set nameTable to "Contact"

tell application "FileMaker Pro"
    tell database nameDatabase
        tell table nameTable
            show every record
        end tell
    end tell
end tell
```

Finding Records Based on a Field Value

To search based on a specific value within one field, add a `whose` clause to the `show every record` command:

```
set nameDatabase to "Learn FileMaker"
set nameTable to "Contact"
set nameField to "Contact Address State"
set textToFind to "NY"

tell application "FileMaker Pro"
    tell database nameDatabase
        tell table nameTable
            show (every record whose cell nameField of table nameTable contains
                textToFind)
        end tell
    end tell
end tell
```

The `whose` clause can use various operators for matching data in the field. These include:

- `contains`
- `does not contain`
- `is equal to`
- `is not equal to`

Further, the clauses can be grouped with others to form compound search criteria. For example:

```
show ( every record
      whose cell nameField1 of table field1 contains value1 or
      whose cell nameField1 of table field1 is equal to value2 and
      whose cell nameField1 of table field3 is not equal to value3
    )
```

Counting Records

An AppleScript can count the number of records in the found set or the total record count in a table.

Counting Records in the Found Set

To count the number of records in the found set, use a `tell window` statement to count the records:

```
set nameDatabase to "Learn FileMaker"
set nameTable to "Contact"

tell application "FileMaker Pro"
    tell window nameDatabase to return the number of every record
end tell
```

In this case, since we are getting the number of records contained within the context of a *window*, we don't have to specify the table name.

Counting Every Record in a Table

To count the number of records in the found set, use a `tell database` statement to count the records.

```
set nameDatabase to "Learn FileMaker"
set nameTable to "Contact"

tell application "FileMaker Pro"
    tell database nameDatabase to the number of every record of table nameTable
end tell
```

In this case, since we are accessing the count of records in a table, we must specify the table name and send the command to the database instead of the window. The code will access the table directly without regard to context, so the the current window doesn't have to be displaying a layout for that table.

Summary

In this chapter, we discussed the basics of sharing databases on a network and described a few methods of accessing shared databases from outside of FileMaker.

CHAPTER 40



Securing Database Access

FileMaker database files can be secured by defining accounts that limit who can gain access and the actions they can perform.

In this chapter, we review the basics of configuring database security settings, covering:

- Exploring the benefits of security
- Defining default security
- Exploring the manage security dialog
- Creating accounts
- Exploring privilege sets
- Defining extended privileges
- Controlling file access

Exploring the Benefits of Security

There are many good reasons to secure your databases. These include:

- Protecting data from access by *unauthorized* users.
- Restricting access to data and functions depending on each *authorized* user's needs based on their job requirements.
- Limiting access to developer functions to only trusted users who possess the skills necessary to make changes to the data structure, layouts, and script workspace.

Files stored on shared workstations or hosted on a network should be secured to avoid an unauthorized user from freely making changes to the file's data or structure. However, even files limited to single-user local use should be secured as added protection against theft or illicit access when the computer is unattended.

Defining Default Security

Technically, every database must have a *minimum* of two security accounts, even if they are not actively used: a guest account and at least one account configured to allow full developer access to the file.

To satisfy this requirement, every new database created by FileMaker automatically has the following two accounts configured:

- [Guest] — Disabled by default, this account can allow a user to enter the file *without* entering a user name or password. This account cannot be deleted but can be left disabled.
- Admin — Enabled by default, this account has no password entered. However, since the Log in using setting of the File Options dialog (Chapter 6, “Database File Options”) is default configured to automatically enter this account when a file is opened, it creates the illusion that new files have no access settings. In fact, new files are bypassing the login screen by logging in as a full access user named Admin with no password.

■ **Caution** When a file is configured to auto-enter a Full Access account or if Full Access accounts don’t require a password, FileMaker Server will automatically reject and refuse to host the file due to the vulnerability. If a currently hosted file is modified to auto-enter a Full Access account, a warning will appear informing you that the file will not be re-hosted after the next time it is closed by the server.

Exploring the Security Dialog

To add security settings to a database file, select the File > Manage > Security menu. When performed for the first time in a file or when the same situation is subsequently detected, a dialog will warn that the file has an Full Access account without a password, which is a major security risk.



Figure 40-1. A dialog warning about the lack of a password on the default full access account named Admin

Click the Set Password button to divert to the Change Password dialog to assign a password to the Admin account or click Ignore to continue to edit security settings without assigning a password.

■ **Note** This dialog will reappear each time you open or close the security dialog and when you switch modes within the window.

Exploring the Interface Modes

Security settings are configured within the Manage Security dialog. This dialog has two optional modes:

- Basic setup
- Detailed setup

The default for a new file will be the basic option. However, the chosen mode is “sticky” so each subsequent opening of the Manage Security dialog for a file will default to whichever mode was displayed when the dialog was closed.

Exploring the Basic Setup Dialog

The Basic Setup version of the Manage Security dialog, shown in Figure 40-2, allows the management of accounts with name, password, and privilege set.

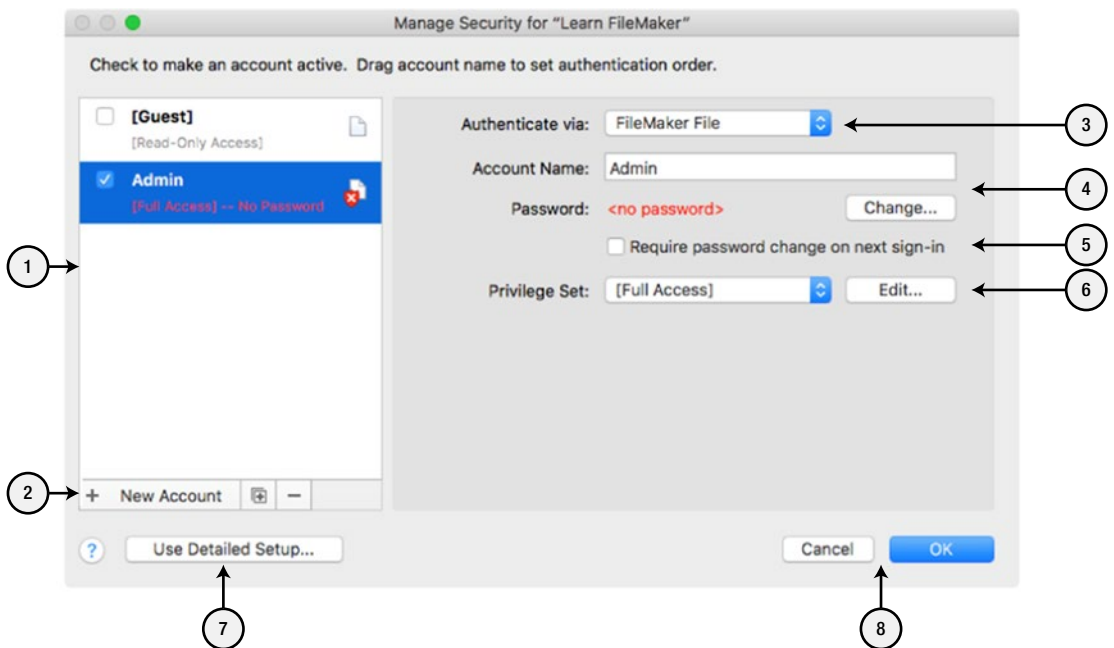


Figure 40-2. The Basic Setup version of the Manage Security dialog

The Basic Setup version of the dialog contains the following controls:

1. **Account List** — This area lists all the accounts currently defined in the file. The check box indicates an active account. Selecting an account displays editable details for it in the adjacent area.
2. **Account Tools** — These three buttons allow the creation, duplication and deletion of accounts in the list above.

3. **Authenticate via** — This pop-up menu offers the following choices for the selected account's authentication method:
 - **FileMaker file** — This option stores all the account information within the current database file.
 - **External Server** — This option stores only a group name and privilege set within the database. The *group name* refers to an account group created on the host computer, external to FileMaker. Any user accounts defined on the host computer that are members of the group will be able to log into the database and with the privilege defined in the assigned set (see “Creating External Accounts” later in this chapter).
 - **Open Authorization (OAuth) Options** — Several additional options allow files hosted on a FileMaker Server to be authenticated through a third-party service over the internet including Amazon, Google, and Microsoft Azure AD.
4. **Local Account Options** — For local accounts:
 - **Account Name** — The name of the account appears in the editable text area and can be modified if desired.
 - **Password** — The existence of a password for the account is indicated here, either as a “no password” warning or a bullet-obscured password display. Click the **Change** button to open the **Change Password** dialog.
5. **Require password change on next login** — If this check box is enabled, the next time the account is used to access the file, a dialog will open requiring the user to authenticate with the current password and to enter a new password.
6. **Privilege Set** — This pop-up menu lists all the privilege sets defined in the file. The selected set controls the functions available to users accessing the database with the account.
7. **Use Detailed Setup** — Click this button to toggle the dialog between basic and detailed mode.
8. **Buttons** — These buttons will close the dialog:
 - **Cancel** — This button will discard all unsaved changes and close the dialog after a warning dialog.
 - **OK** — This button will save changes and close the dialog after requiring that the user enter the name and password of an account that has full access, proving that they aren't about to permanently lock themselves out of the file.

Exploring the Detailed Setup Dialog

The **Detailed Setup** version of the **Manage Security** dialog, shown in Figure 40-3, is the traditional FileMaker security dialog. This dialog allows the same functions available in basic mode with the addition of extended privileges and file access settings.

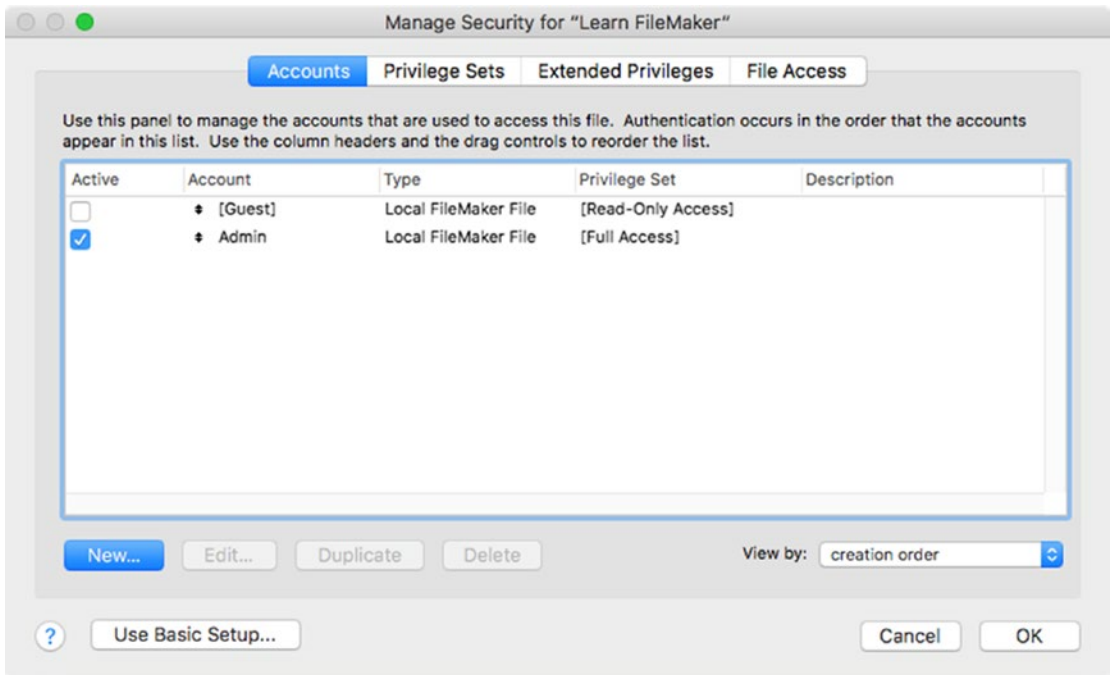


Figure 40-3. The Detailed Setup version of the Manage Security dialog

The dialog is broken down into the following four tabs:

- Accounts
- Privilege Sets
- Extended Privileges
- Accounts

Each tab contains the following features:

- A list of settings based on the tab.
- Buttons to Create, Edit, Duplicate, and Delete items in the list.
- A View by pop-up menu to sort the list by various criteria.
- Column heading buttons to click to sort the list.
- Double-click an item in the list or select an item and click Edit to open an Edit Account dialog.

■ **Note** The remaining examples in this chapter will be described using the Detailed Setup mode.

Creating Accounts

FileMaker accounts can be created to authenticate locally or using an external server.

Creating a New Local Account

To get started creating a new local account for full developer access, open the Manage Security dialog. Click on the Accounts tab and then on the Create button. This will open a new empty account in the Edit Account dialog, shown in Figure 40-4.

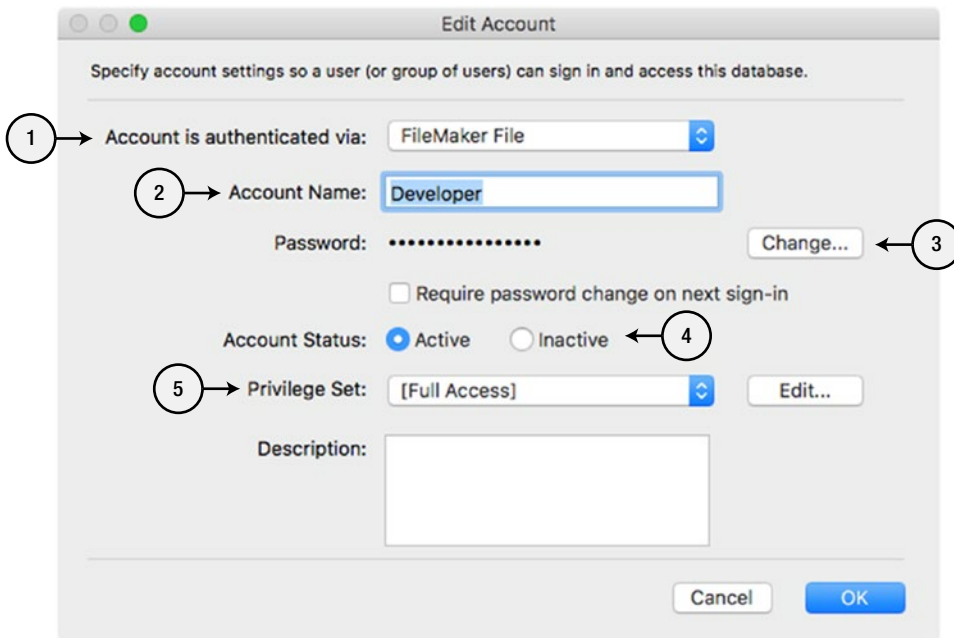


Figure 40-4. The Edit Account dialog

Follow these steps to set up the new account:

1. Account is authenticated via — Select FileMaker File in the pop-up menu.
2. Account Name — Enter “Developer” as the account name.
3. Password — Click the Change button to open the Set Password dialog, shown in Figure 40-5. Enter a password in both the New Password and Confirm New Password text areas. Click the Set Password button to save the password and return to the Edit Account dialog.

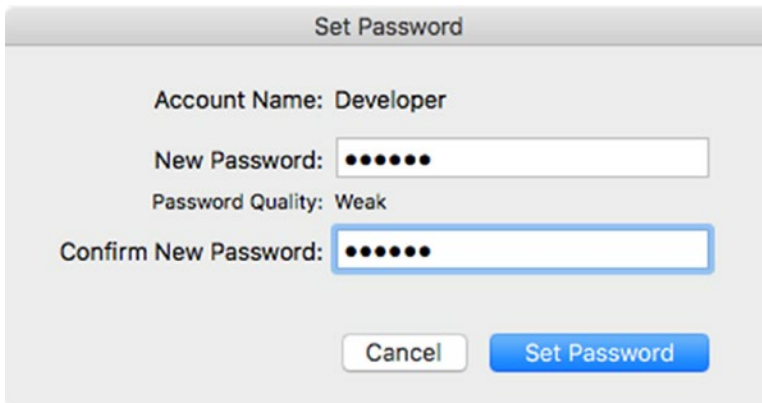


Figure 40-5. The Set Password dialog

4. Account Status — The new account should default to an Active status. If not, change it.
5. Privilege Set — This pop-up menu lists all the privilege sets defined in the file. Since this will be an account for developers, choose [Full Access].

When finished, click the OK button to save the changes and close the Edit Account dialog. You should see the new Developer account listed in the Manage Security dialog. Click the OK button there to save the security settings. FileMaker will ask you to enter a full access user name and password to confirm that you have not forgotten your password.

■ **Caution** Make sure that you write your password down as there is *no* process to recover a forgotten password.

Once finished, close the file and then reopen it. You should see a credentials dialog like the one in Figure 40-6.

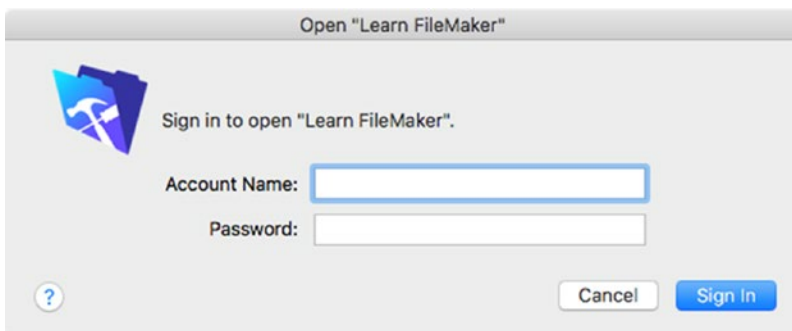


Figure 40-6. The credentials dialog requesting an account name and password when opening a file

Type in the account name and password and click OK to open the file.

Creating a New External Account

When authenticating using an external server, user accounts and user groups are created on the host FileMaker Server computer. In the database, external server accounts are created with the group name and a privilege set. When configured properly, this allows any user to log into the database using their user name and password, gaining access to features based on their group membership.

Creating an External Server Account

To begin, open the Manage Security dialog. Click on the Accounts tab and then on the Create button. This will open a new empty account in the Edit Account dialog. Select External Server from the pop-up menu, enter a Group Name, and select Privilege Set as shown in Figure 40-7. Then click OK to save the account settings.

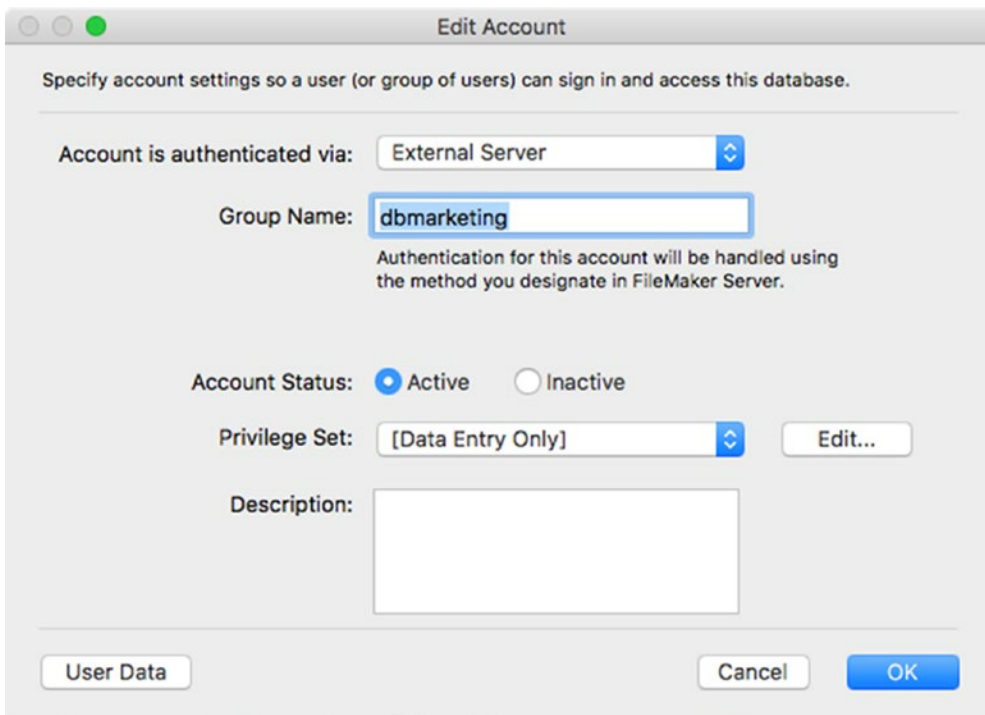


Figure 40-7. The Edit Account dialog when authenticating via an external server

Configuring FileMaker Server for External Authentication

The database must be hosted on a FileMaker Server that is configured to allow authentication from both local and external accounts. To confirm this, follow these steps:

1. Open and sign into the Admin Console (Chapter 39).
2. Click on Database Server in the sidebar.
3. Click on the Security tab.

4. Under the Client Authentication heading, select FileMaker and external server accounts from the pop-up menu.
5. Click the Save button and sign out of the Admin Console.

Creating Groups and Accounts (macOS)

Next you must create groups and user accounts on the FileMaker Server host computer's operating system.

Creating a Group

To create a group in macOS Sierra, follow these steps:

1. Open the System Preferences application.
2. Click on the Users & Groups icon.
3. Click the lock icon to make changes. You will be asked to authenticate with credentials for a user with administration privileges for the computer.
4. Click the + icon to open the New Account panel and then select Group from the pop-up menu at the top. This will change the options in the panel to those shown in Figure 40-8.

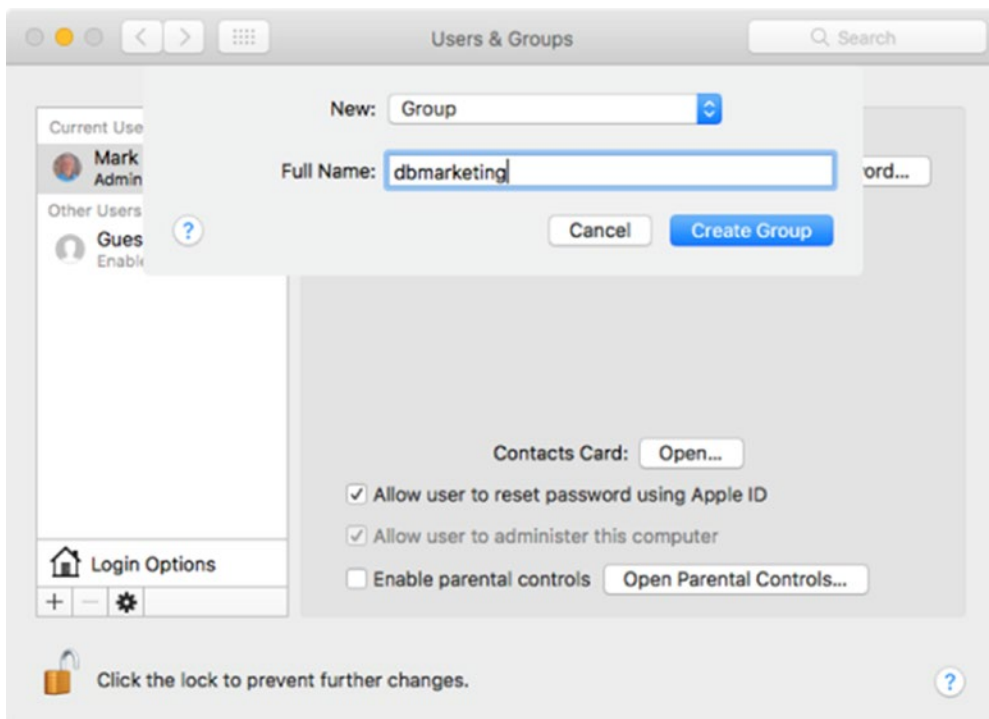


Figure 40-8. The New Account panel of the Users & Groups section of the System Preference application once Group is selected from the pop-up menu

5. Enter the name of the group in the test area and click the Create Group button.

■ **Caution** When entering a group name, it is a good idea to always use lowercase or at least choose a case and stick with it. Changing the case or the name of a group later may require some advanced editing of accounts, especially depending on your version of the macOS operating system.

Creating a User

To create a user account in macOS (Sierra), follow these steps:

1. Open the System Preferences application.
2. Click on the Users & Groups icon.
3. Click the lock icon to make changes. You will be asked to authenticate with credentials for a user with administration privileges for the computer.
4. Click the + icon to open the New Account panel and then select Sharing Only from the pop-up menu at the top. This will change the options in the panel to those shown in Figure 40-9.



Figure 40-9. The New Account panel of the Users & Groups section of the System Preference application once Sharing Only is selected from the pop-up menu

■ **Note** The Sharing Only option will create an account for the user on the computer without creating a user folder and is ideal to grant database access. A regular user account on the computer can be assigned to a group if desired.

5. Enter the Full Name, Account Name, and Password and click the Create User button.

Assigning a User to a Group

Once the group and user account is created, select the group and enable the user membership into the group, as shown in Figure 40-10.

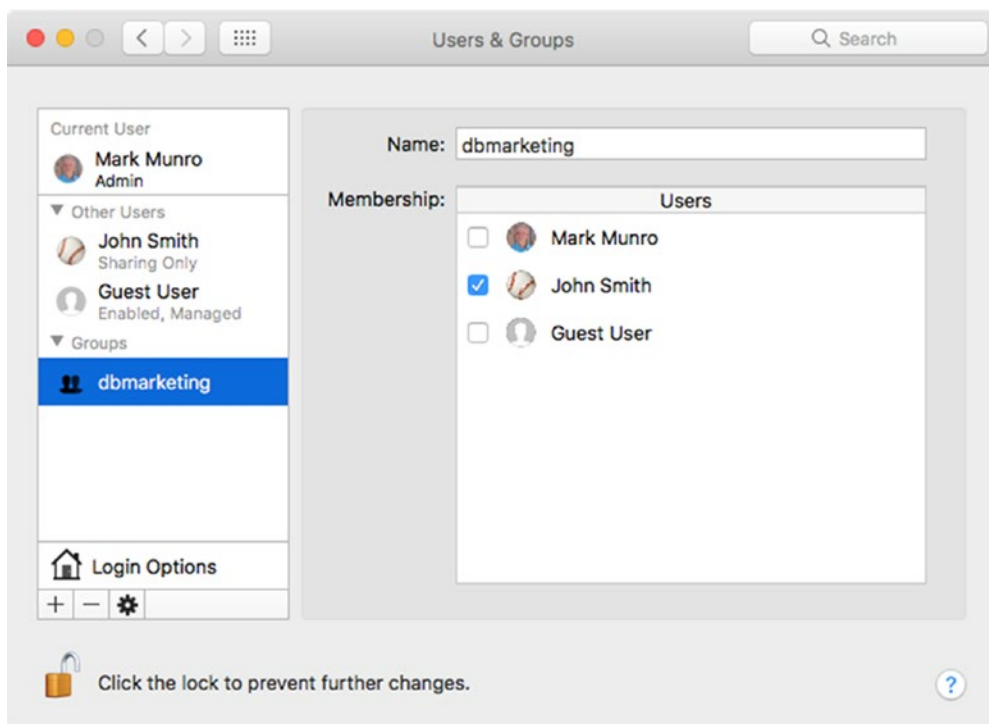


Figure 40-10. Once a user is assigned membership to the group, they will be able to log into the database with their user name and password

Exploring Privilege Sets

A *privilege set* is a collection of permissions for data access, design, and other features, which can be assigned to one or more accounts to define a user's abilities within a database file. Files come with default privilege sets, but you can create as many new sets as are needed to define the various types of access for your projects.

Default Privilege Sets

Every new database will automatically have the following three permanent, non-editable privilege sets:

- [Full Access] — This set contains full access to *every available feature*.
- [Data Entry Only] — This set contains limited access to perform tasks associated with data entry, including:
 - Can create, edit, and delete records in every table.
 - Limited to viewing layouts and value lists.
 - Limited to executing scripts.
 - Can print, export.
 - Will be disconnected from the server when idle.
 - Can change their own account password.

Creating Custom Privilege Sets

To get started creating a new privilege set, open the Manage Security dialog. Click on the Privilege Sets tab and then on the Create button. This will open a new empty account in the Edit Privilege Set dialog, shown in Figure 40-11.

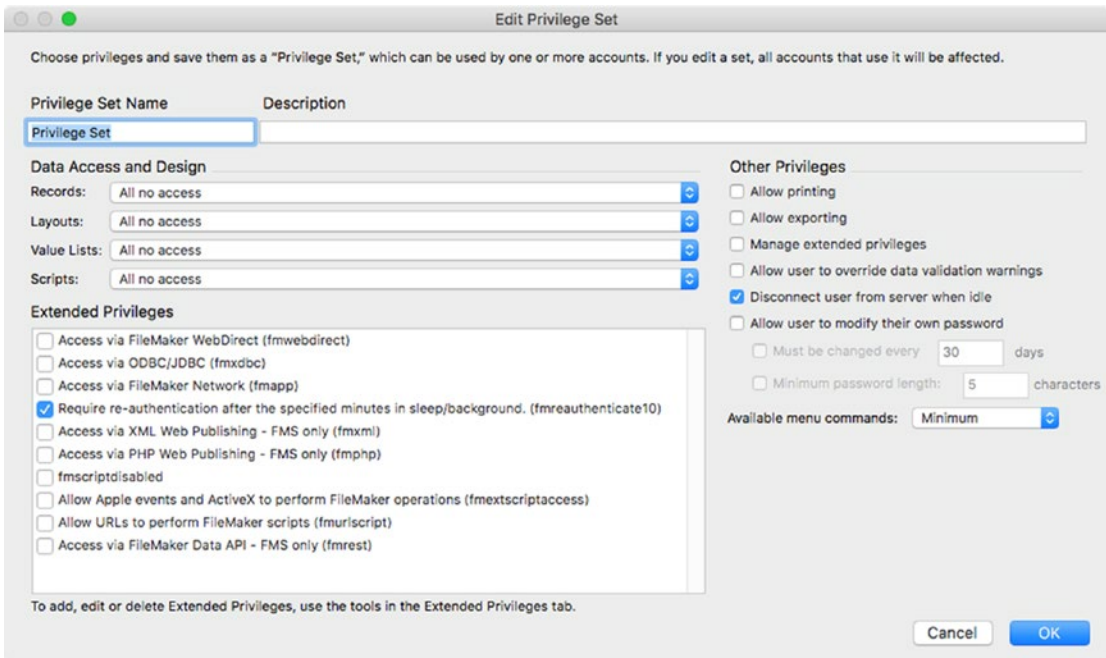


Figure 40-11. The Edit Privilege Set dialog

The default configuration of the new set is essentially no access. You must assign a unique name to the set and explicitly assign those permissions you want to allow.

Defining Available Permissions

The dialog organizes permissions into the following categories:

- Data access and design privileges
- Extended privileges
- Other privileges

Assigning Data Access and Design Privilege Settings

The *data access and design* privileges define what actions accounts assigned the privilege set can perform regarding access to records, layouts, value lists, and scripts.

Each of these has several options that are applicable to all entities within the class. Generally, these are open access, view/execute only, no access. In addition to these, each has an option to assign custom privileges on a granular level.

Accessing Records

The *records access* pop-up menu on the Edit Privilege Set dialog offers the following four options:

- Create, edit, and delete in all tables — Users of the privilege set will have all available record permissions in every table.
- Create and edit in all tables — Users of the privilege set will be limited to creating and editing records in every table, but the Delete Record function will be completely disabled.
- View only in all tables — Users of the privilege set will be limited to viewing records only while the Create, Duplicate, and Delete Record functions will be disabled.
- All no access — Users of the privilege set will have no record permissions and be unable to see records in any table.
- Custom privileges — When none of the other options provide the level of detail required due to the complexity of the database or the type of user, choose this option to assign custom privileges.

Assigning Custom Record Privileges

To assign customized record permissions, choose Custom Privileges from the Records pop-up menu to open the Custom Record Privileges dialog, shown in Figure 40-12.

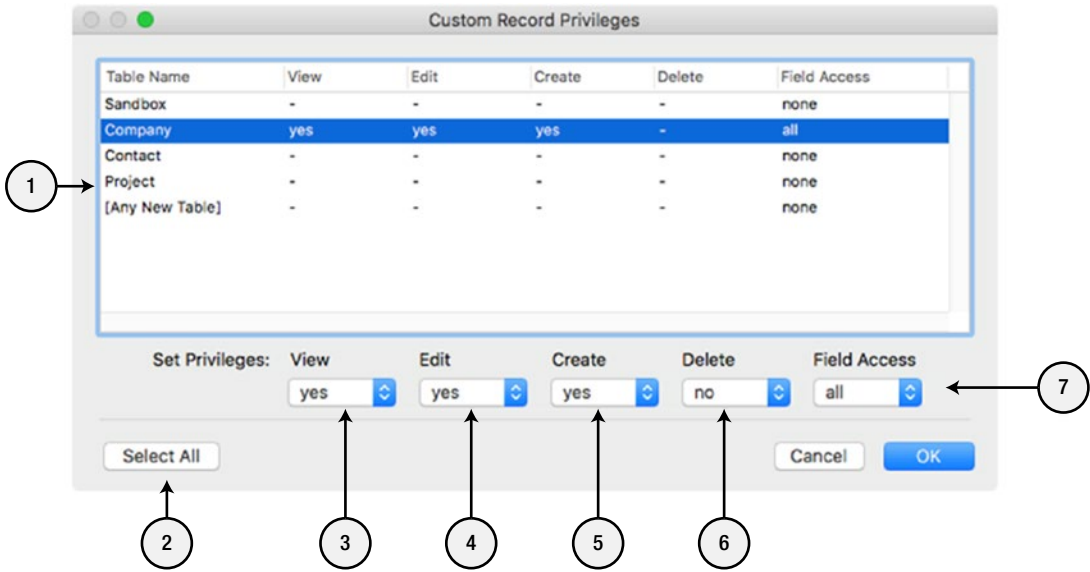


Figure 40-12. The Custom Record Privileges dialog allows fine control of permissions for record access customized per table

Note Remember, the permissions you set in this dialog affect records for *only* the privilege set you are currently editing. Each set can have completely different settings.

The Custom Record Privileges dialog contains the following controls:

1. **Tables** — This area lists every table currently defined in the database with columns for each type of available action. The column headings allow you to sort the table list by the values in that column, toggling between ascending and descending with each click. Select a table to edit its Set Privileges menus or choose [Any New Table] to define the default permission for any tables added to the database in the future.
2. **Select All** — Click this button to select every table in the list at once.
3. **View** — This menu contains the following options for *viewing* records in the selected table(s): yes, no, or limited. This last option opens a Specify Calculation dialog to control when a record is viewable with a custom formula.
4. **Edit** — This menu contains the following options for *editing* records in the selected table(s): yes, no, or limited. This last option opens a Specify Calculation dialog to control when a record is editable with a custom formula.
5. **Create** — This menu contains the following options for *creating* records in the selected table(s): yes or no.

6. Delete — This menu contains the following options for *deleting* records in the selected table(s): yes, no, or limited. This last option opens a Specify Calculation dialog allowing the entry of a custom formula that will be used to determine when a record can be deleted.
7. Field Access — This menu contains the following options for *field access* in the selected table(s): all, none, or limited. This last option opens a Custom Field Privileges dialog, shown in Figure 40-13.

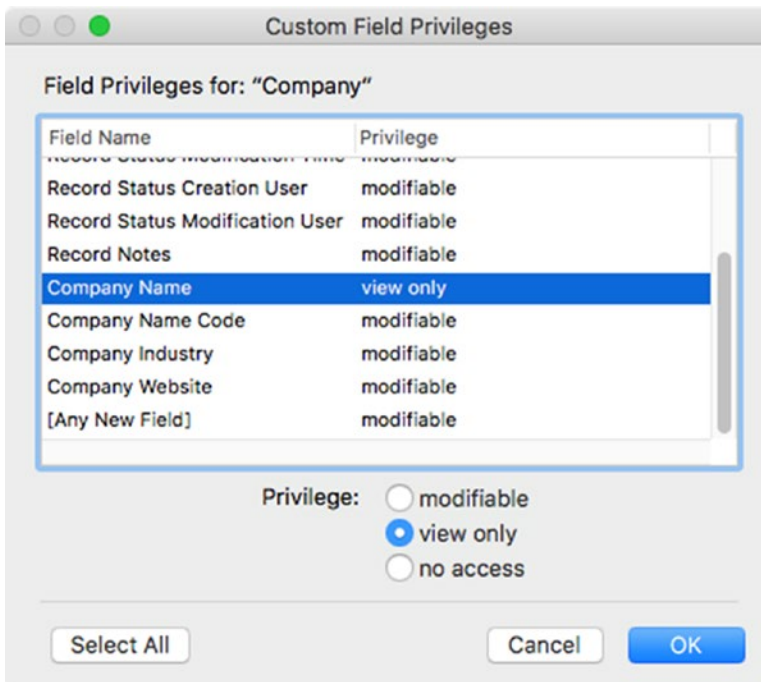


Figure 40-13. The Custom Field Privileges dialog is used to control the privileges of specific fields for a table

Assigning Custom Field Privileges

The Custom Field Privileges dialog displays a list of every field in the selected table. Select one or more fields in the list to edit their Privilege settings, choosing between *modifiable*, *view only*, and *no access*. The permissions assigned to the [Any New Field] option will automatically be assigned to all new fields added to the table in the future.

Accessing Layouts

The *layouts access* pop-up menu on the Edit Privilege Set dialog offers the following four options for layout access and design capabilities:

- All *modifiable* — Users of the privilege set will be able to view and modify any layout within the database.
- All *view only* — Users of the privilege set will be able to view any layout in the database but be prohibited from making design changes.

- All no access — Users of the privilege set will not be able to see any layouts in the database.
- Custom privileges — When none of the other options provides the level of detail required due to the complexity of the database or the type of user, choose this option to assign custom privileges.

Caution If a user can't access a layout, neither can a script run from their session unless that script is configured to run with full access.

Assigning Custom Layout Privileges

To assign customized layout permissions, choose Custom Privileges from the Layouts pop-up menu to open the Custom Record Privileges dialog, shown in Figure 40-14.

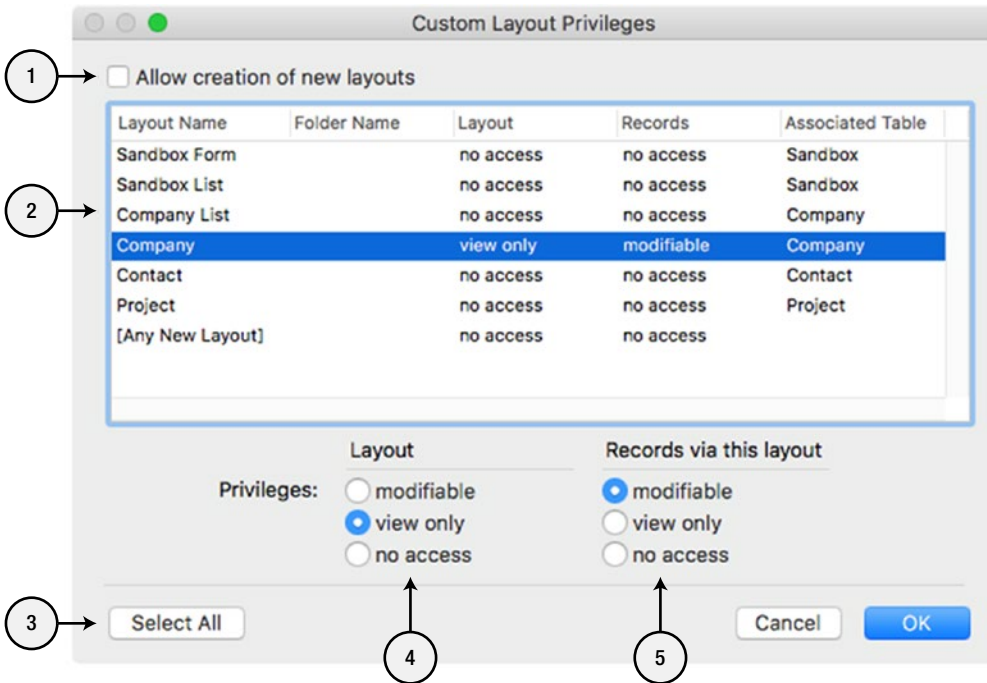


Figure 40-14. The Custom Layout Privileges dialog allows fine control of permissions for individual layouts

The Custom Layout Privileges dialog contains the following controls:

1. **Allow creation of new layouts** — Enable this check box to allow accounts assigned to the privilege set to create new layouts.
2. **Layouts**— This area lists every layout in the database with columns indicating the current level of access allowed. The column headings allow you to sort the list by the values in that column, toggling between ascending and descending with each click. Select a layout to edit its settings or choose [Any New Layout] to define the default permission for any layouts added to the database in the future.
3. **Select All** — Click this button to select every layout in the list at once.
4. **Privileges for Layout** — These radio buttons offer a choice of three access levels for the selected layout(s): *modifiable*, *view only*, *no access*.
5. **Privileges for Records via this layout** — These radio buttons offer a choice of three access levels for record access when viewing the selected layout(s): *modifiable*, *view only*, *no access*. These settings apply to record access for only the selected layout(s) and further refine, but do not override, the record access granted globally to records.

Accessing Value Lists

The *value list access* pop-up menu on the Edit Privilege Set dialog offers the following four options for value list access and editing capabilities

- **All modifiable** — Users of the privilege set will be able to see and edit the settings of every value list in the database.
- **All view only** — Users of the privilege set will be able to see and use every value list assigned to fields but will not be able to edit their settings unless specifically permitted by the use on a field using the Edit property of a field's Control Style.
- **All no access** — Users of the privilege set will not be able to see any value lists.
- **Custom privileges** — When none of the other options provide the level of detail required due to the complexity of the database or the type of user, choose this option to assign custom privileges.

Assigning Custom Value List Privileges

To assign customized value list permissions, choose Custom Privileges from the Value List pop-up menu to open the Custom Value List Privileges dialog, shown in Figure 40-15.

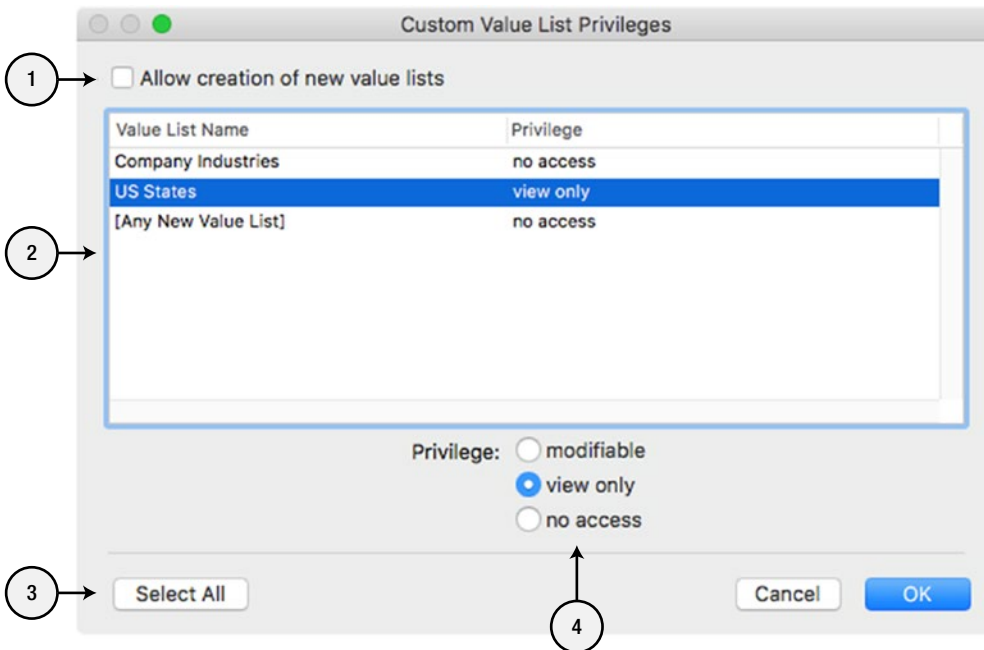


Figure 40-15. The Custom Value List Privileges dialog allows fine control of permissions for individual value lists

The Custom Value List Privileges dialog contains the following controls:

1. Allow creation of new value lists — Enable this check box to allow accounts assigned to the privilege set to create new value lists.
2. Value Lists— This area lists every value list in the database with the current level of access allowed. The column headings allow you to sort the list by the values in that column, toggling between ascending and descending with each click. Select a value list to edit its settings or choose [Any New Value List] to define the default permission for any value lists added to the database in the future.
3. Select All — Click this button to select every layout in the list at once.
4. Privileges — These radio buttons offer a choice of three access levels for the selected value lists(s): modifiable, view only, no access.

Accessing Scripts

The *scripts access* pop-up menu on the Edit Privilege Set dialog offers the following four options for value list access and editing capabilities:

- All modifiable — Users of the privilege set will be able to execute, open, view, create, and edit all scripts within a database.
- All executable — Users of the privilege set will be limited to executing scripts without the ability to create, view, or edit them.

- **All no access** — Users of the privilege set will not be able to execute any scripts or access them in any way.
- **Custom privileges** — When none of the other options provide the level of detail required due to the complexity of the database or the type of user, choose this option to assign custom privileges.

Assigning Custom Script Privileges

To assign customized script permissions, choose **Custom Privileges** from the **Scripts** pop-up menu to open the **Custom Script Privileges** dialog, shown in Figure 40-16.

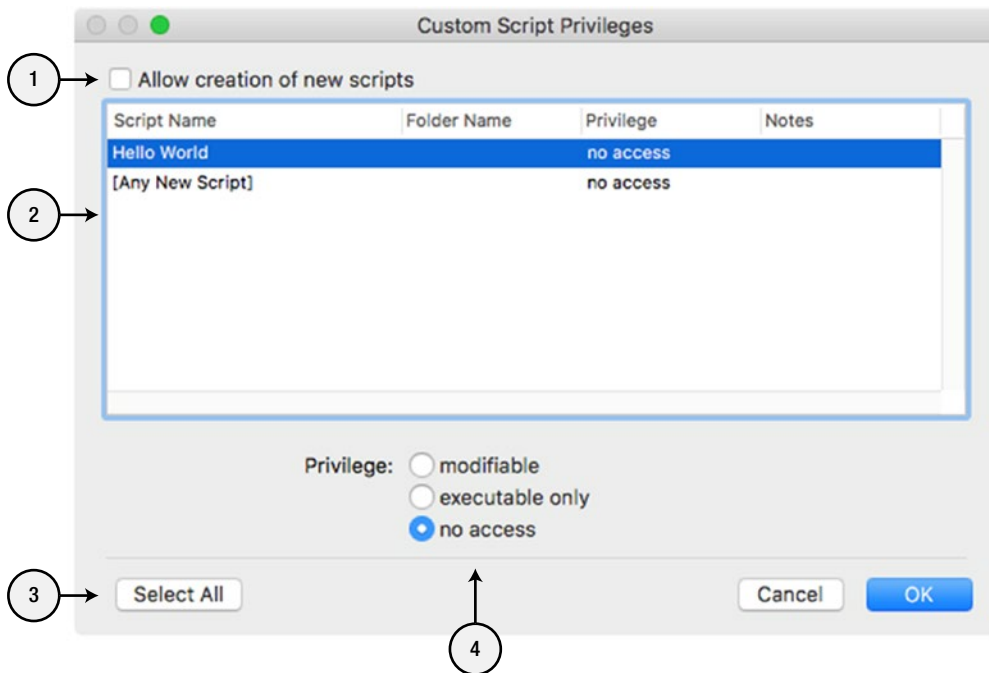


Figure 40-16. The *Custom Script Privileges* dialog allows fine control of permissions for individual scripts.

The **Custom Value List Privileges** dialog contains the following controls:

1. **Allow creation of new scripts** — Enable this check box to allow accounts assigned to the privilege set to create new scripts.
2. **Scripts**— This area lists every script in the database. The column headings allow you to sort the list by the values in that column, toggling between ascending and descending with each click. Select a value list to edit its settings or choose **[Any New Script]** to define the default permission for any scripts added to the database in the future.

3. **Select All** — Click this button to select every script in the list at once.
4. **Privileges** — These radio buttons offer a choice of three access levels for the selected value lists(s): `modifiable`, `executable only`, `no access`.

Assigning Other Privilege Settings

The other options available on the right-hand side of the `Edit Privilege Set` dialog control the following miscellaneous capabilities:

- **Allow Printing** — When enabled, users can save records as a PDF file and print records manually using the `Print` command.
- **Allow exporting** — When enabled, users can perform a variety of data output functions including exporting records, saving records as an Excel file, copying all records in the found set, saving a copy of records, and various `AppleEvents` that access and extract record information.
- **Manage extended privileges** — When enabled, users can enable and disable extended privileges in the various sharing dialogs available in the `File > Sharing` submenu and edit extended privileges in the `Manage Security` dialog.
- **Allow user to override data validation warnings** — When enabled, users can override any field validation warning dialog that is not explicitly defined to not allow it (Chapter 8, “Validation Control”).
- **Disconnect user from server when idle** — When enabled, idle users will be automatically disconnected from the file when it is hosted on a `FileMaker Server`.
- **Allow user to modify their own password** — When enabled, users can change their password by selecting `File > Change Password`. Two adjacent check boxes allow you to specify:
 - Users must change their password periodically after a specified number of days.
 - A minimum length for the user passwords.
- **Available menu commands** — While a user’s access to menu items can be influenced by other permissions granted to the privilege set to which they are assigned, this menu specifies a category of commands they can access. Choose from the following options:
 - **All** — The user will have access to the entire menu bar.
 - **Editing Only** — The user will have access to menu commands that perform basic editing functions.
 - **Minimum** — The user will have access to no menu commands except for basic functions such as open, close and create database files, window functions, perform scripts, preferences, and help.

Assigning Extended Privilege Settings

The *extended privileges* section of the `Edit Privilege Set` dialog allows you to choose from a list of additional privilege options that extend the functionality of a database. To assign an extended privilege to the current privilege set, check the box next to it in the list.

FileMaker defines ten default options and offers the ability to for developers to create custom extended privileges. These are described in the next section.

Exploring Extended Privileges

An *extended privilege* is a keyword-based permission setting that can be assigned to one or more privilege sets to control access to a specific type of functionality within a database.

FileMaker includes several extended privileges that are hard-coded to specific functions, mostly various types of networking access. However, developers can define custom extended privileges and use them as a way of controlling access to features they build.

Extended privileges are listed in and defined from the Extended Privileges tab of the Manage Security dialog, shown in Figure 40-17. They are also accessible as check box options in the Edit Privilege Set dialog.

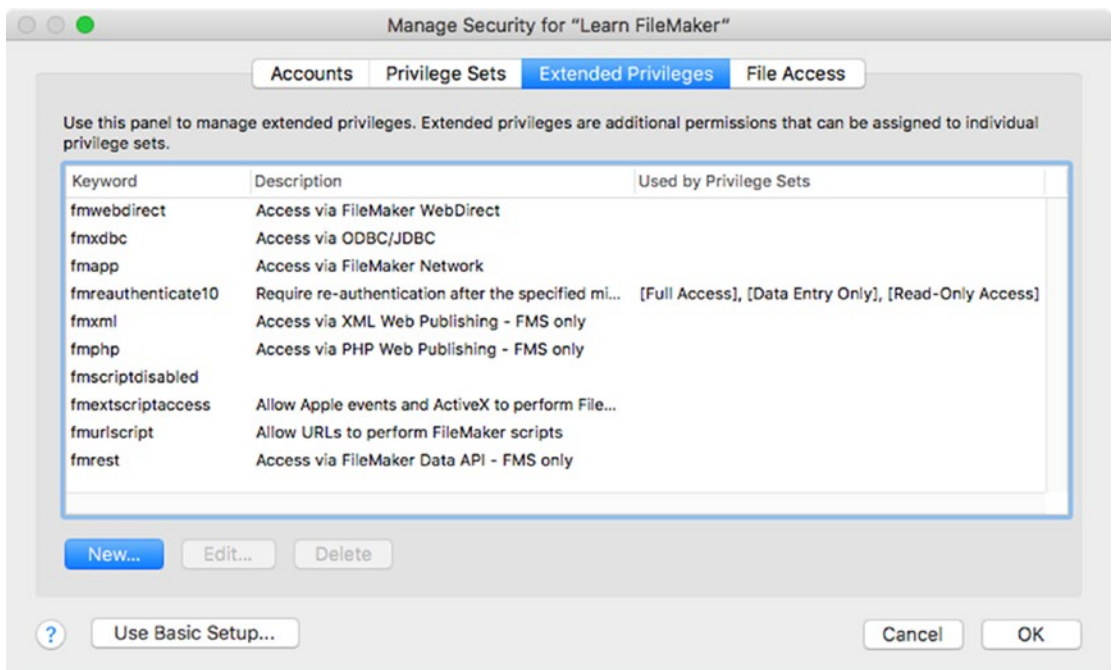


Figure 40-17. The Extended Privileges tab of the Manage Security dialog

Defining Built-In Extended Privileges

The following extended privileges, mostly controlling types of network access to a database, are included in every database by default:

- `fmwebdirect` — Grants the user access to the database from a web browser using FileMaker WebDirect.
- `fmjdbc` — Grants the user access using the database as an ODBC/JDBC data source.

- `fmapp` — Grants the user access to the file shared by a peer or a server using the FileMaker Network protocol from a copy of FileMaker Pro, FileMaker Advanced, or FileMaker Go.
- `fmreauthenticate10` — When accessing a file with FileMaker Pro or FileMaker Go, a user will have to re-authenticate after a specified number of minutes of sleep or of using a different application.
- `fmxml` — Grants the user access to databases hosted by FileMaker Server from a web browser or other application using XML Web Publishing.
- `fmphp` — Grants the user access to databases hosted by FileMaker Server from a web browser or other application using PHP Web Publishing.
- `fmscriptdisabled` — This setting has been deprecated and will be removed soon.
- `fmextscriptaccess` — Grants the user to access the database with AppleScript (macOS) and ActiveX (Windows).
- `fmurlscript` — Grants the user access to run a database script from a URL (Chapter 39, “Using the FileMaker URL”).
- `fmrest` — Grants the user access to a database file from a Representational State Transfer (RESTful) web service via the FileMaker Data API when hosted on a FileMaker Server.

Each of these settings is hard-coded to a built-in FileMaker feature. Some are connected to the FileMaker Network Settings dialog, accessed from `File > Sharing` menus, described in Chapter 39. For example, if a privilege set is allowed `fmapp` access in the `Edit Privilege Set` dialog, then this privilege will be automatically enabled in the FileMaker Network Settings dialog and vice versa.

Using Custom Extended Privileges

A *custom extended privilege* is a keyword-based permission that is defined by a database developer to dynamically assign specific features to privilege sets with minimal effort.

This feature allows permission limits to be imposed at a deeper and more granular level than the custom privilege menus in the `Edit Privilege Set` dialog. For example, the default permission options allow a calculation field to be flagged as `accessible`, `read only`, or `not accessible` while a custom extended privilege can control, hide or change the functionality of a portion of the field’s formula while keeping the field itself accessible. Similarly, a script can remain executable while some steps within it can be conditionally skipped based on a custom extended privilege.

Anywhere in a database that a formula can be entered, one might include a permission-based limitation. For example:

- *Calculation field formulas* — Changing or excluding some portion of a formula’s result depending on permissions.
- *Layout object visibility* — Hiding objects depending on permissions.
- *Script steps* — Altering or excluding groups of script steps conditionally based on permissions.
- *Custom menu visibility* — Installing menus or menu items depending on permissions.

While accounts define *user permissions* and privilege sets define *user group permissions*, custom extended privileges can define *feature group permissions*.

Defining Feature Groups

A *feature group* is a collection of resources that pertain to a custom feature of a database. The resources regarded as a feature group may not be discreetly contained. Instead, they may be found in several different tables and may be interwoven with many other features.

For example, imagine a feature group called `InvoiceApproval` that contains only processes that control approving an invoice. These would be a subset of the overall `Invoice` table, its layouts and scripts. They might be present in both an `Invoice` and/or an `Invoice Line Item` table and could be embedded within scripts that perform other functions unrelated to approving invoices. There may even be a field in a `Contacts` table that controls some aspect of the approval process. These various resources might need to be hidden or disabled for user accounts that otherwise have access to all other contact and invoicing functions.

Once a feature group requiring special permission restrictions is identified, extended privileges is usually the best method of securing it while allowing for future flexibility.

Limiting Permissions for a Feature Group

To understand the full benefit of using custom extended privileges to control conditional access in formulas, let's compare the methods of achieving this level of control both with and without them.

Limiting without Extended Privileges

Limiting access to features can be done by adding a condition into the formulas of every object in a feature group based on the current user's account name and/or privilege set, as illustrated in Figure 40-18.

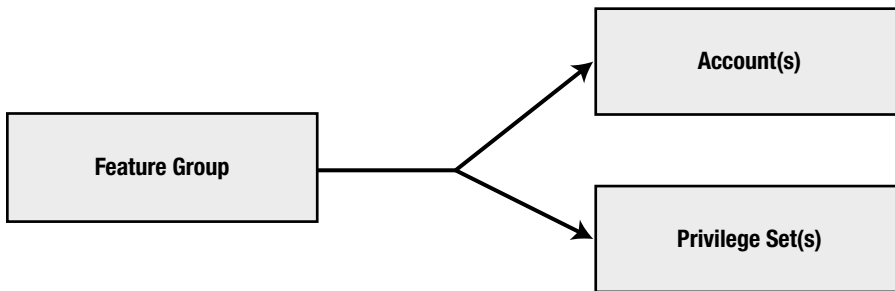


Figure 40-18. Without extended privileges, a formula must directly check account(s) and/or privilege set(s) to confirm permission

Altering the output of a calculation field based on the user's account name can be done using a Case statement as shown in this example:

```
Case ( Get ( AccountName ) = "Tim Smith" ; "Yes" ; "No" )
```

While this approach might be fine in many circumstances, there is a major drawback when future personnel changes occur and Tim Smith is no longer with the company or has moved on to a new role. The problem becomes compounded with each additional object involved in the feature group, increasing the burden and possibility of overlooking something when changing them all.

The situation can be improved slightly by comparing the user's privilege set instead of account name, as shown in this example:

```
Case ( Get ( AccountPrivilegeSetName ) = "Finance" ; "Yes" ; "No" )
```

Now, if members of the privilege set are added or removed, the formula(s) will not require a change. However, even with this method, there is still the potential for problems.

Using an account name or privilege set for permission conditions is not as simple as indicated in these examples. Often, you will often need to include *multiple conditions*. For example, since a Developer account must be a member of the Full Access privilege set, that may need to be included in the formula, otherwise the user would be unable to test the feature. In other situations, you will need to include the privilege sets for users responsible for using the feature, administrators responsible for overseeing them, and developers.

Also, any future restructuring of the privilege sets will result in the same massive editing requirement as using the account name. For example, if the members of the Finance set need to be divided between two or more different subsets to accommodate other privilege requirements or shifts in responsibility, every formula in the database with conditional functionality based on the privilege set will need to be modified. This is where the benefit of using custom extended privileges become clear.

Limiting with Extended Privileges

Limiting access to features using custom extended privileges is easy to implement and provides the most flexibility for long-term changes to your security structure. In this case, the formulas of objects in the feature group will contain conditions hard-coded to an extended privilege created to control permission of the feature, with that assigned to privilege sets that are assigned to user accounts, as shown in Figure 40-19.

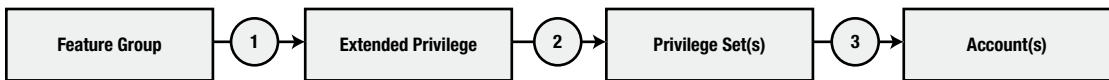


Figure 40-19. With extended privileges, a feature group checks an extended privilege with future changes to permission easily reconfigurable security settings

In this case, the reconfiguration burden is shifted away from the potentially numerous formulas of resources in the feature group and onto the Manage Security dialog where it is much easier to perform:

1. The formula of the objects in the feature group contains hard-coded conditions locked to a custom extended privilege specific to the feature.
2. The feature’s extended privilege can be easily assigned to one or more privilege sets by enabling a check box on either the Edit Extended Privilege or Edit Privilege Set dialogs.
3. User accounts are assigned a privilege set, thereby inheriting the extended privilege to the feature group.

Using this method, all permission changes are easily adjusted within security settings.

Creating a Custom Extended Privilege

The first step to creating a custom extended privilege is to identify a feature group that requires limited permissions. This might be done during the planning stages or after the resources are in place. Once identified, you will need to develop a name for the group. Ideally the name should be made up of two or more words that identify the table or function group of tables involved and the feature itself, for example InvoiceApproval, FinanceReporting, or ClientEmail.

■ **Tip** Choose custom extended privilege names carefully before implementing them to avoid future changes that would require all the formulas involved to be updated, the very task we are trying to avoid in the first place.

Open the Manage Security dialog and click on the Extended Privileges tab. Then click the New button to open a new, empty extended privilege in an Edit Extended Privilege dialog, shown in Figure 40-20.

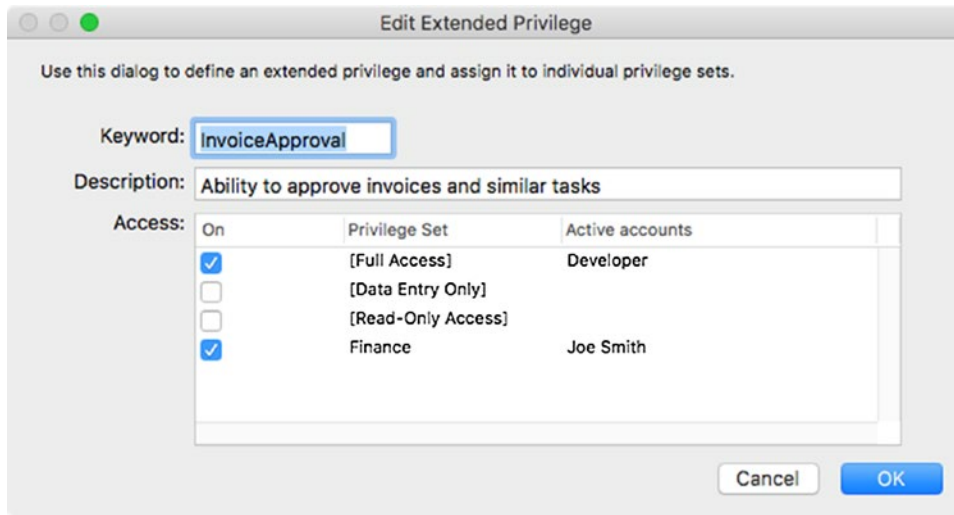


Figure 40-20. The Edit Extended Privilege dialog

The Edit Extended Privileges contains the following controls:

1. **Keyword** — This text area contains the name of the extended privilege.
2. **Description** — This text area can contain an optional description of the feature group it controls.
3. **Privilege Sets** — This area lists all the defined privilege sets in the database. Click the **On** check box on one or more to enable their permission to the feature group.

Once configured, click **OK** to save the new extended privilege.

Now if you open a privilege set, you will see the new extended privilege as an option that can be enabled or disabled as needed. Based on the example above, opening the Finance privilege set will reveal the new InvoiceApproval extended privilege enabled, as shown in Figure 40-21.

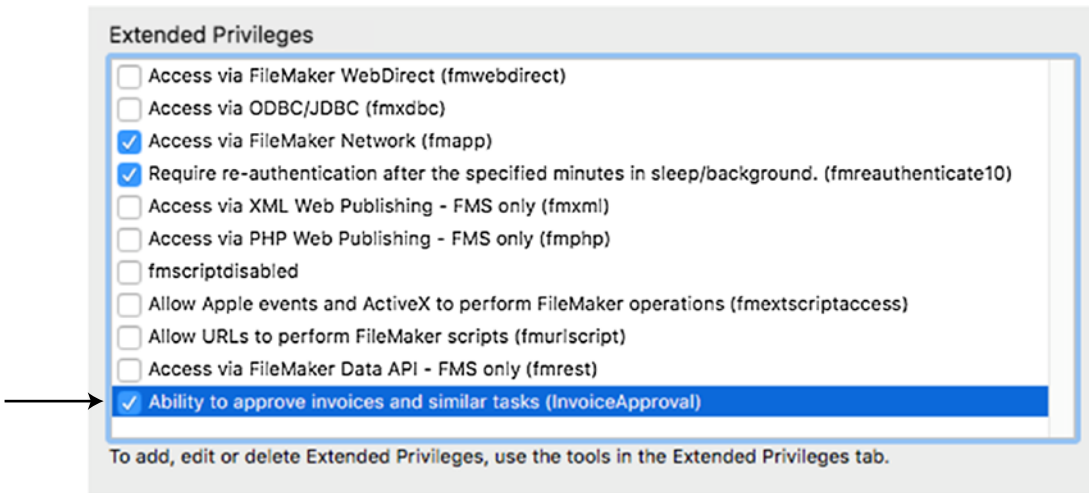


Figure 40-21. The new *InvoiceApproval* extended privilege enabled on the *Finance* privilege set

Now, any formula that requires conditional permission can be hard-coded to the new *InvoiceApproval* extended privilege, as shown here using a Case statement:

```
Case (
    not IsEmpty (
        FilterValues ( Get ( AccountExtendedPrivileges ) ; "Finance" )
    ) ;
    "Yes" ; "No"
)
```

Since the `Get (AccountExtendedPrivileges)` function returns a list of all the extended privileges that the user account’s privilege set has enabled, you must use the `FilterValues` function to detect a single value within.

Controlling File Access

The final tab of the `Manage Security` dialog, shown in [Figure 40-22](#), is named `File Access` and allows limitation to access of the database from other FileMaker databases.

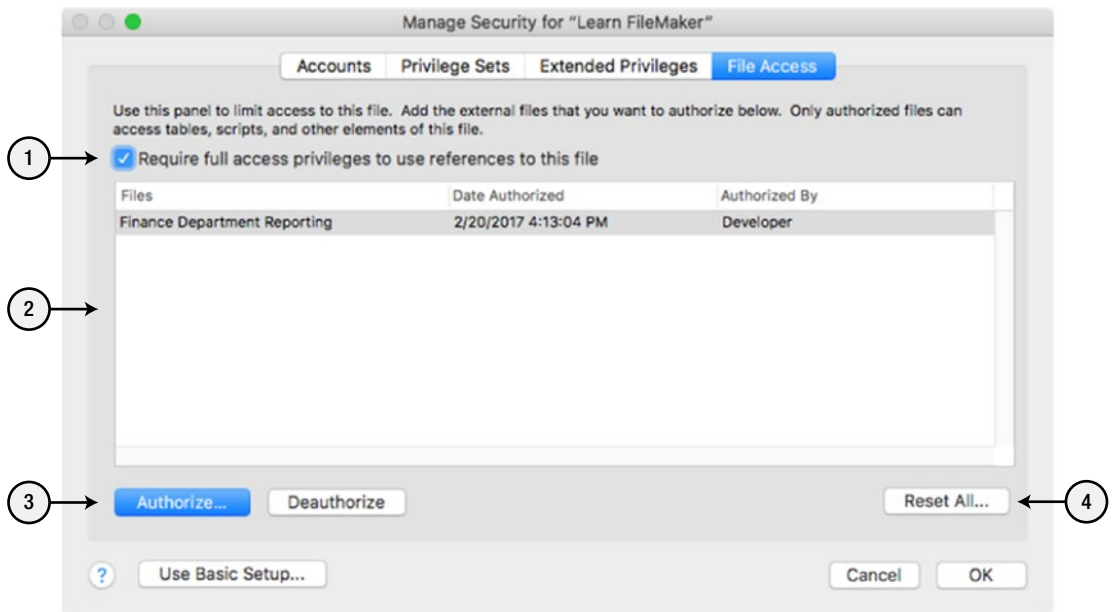


Figure 40-22. The File Access tab of the Manage Security dialog

The following controls are available on the File Access tab of the Manage Security dialog:

1. Require full access privileges to use references to this file — Enable this check box to deny access to any FileMaker databases not explicitly listed below.
2. Files — This area lists all the files that have been granted explicit access to use this database as an external data source.
3. Authorize — Click this button to open a Choose File dialog and select that FileMaker database is authorized to access the current database. You may be required to enter credentials for an account with full access to the selected file.
4. Deauthorize — Click this button to remove the selected database from the list of authorized files.
5. Reset All — Click this button to reset the ID of all the files listed and deauthorize them and require reauthorization. This is used to ensure unique IDs if a duplicate or clone of a file will be granted access along with the original.

Summary

In this chapter, we discussed many of the security options available to maintain secure databases.

CHAPTER 41



Advanced Sharing and Analysis Tools

In addition to script debugging and menu customization features, the FileMaker Pro Advanced Tools menu contains developer utilities to assist in analyzing, sharing, and securing a database.

In this chapter, we review these tools, covering:

- Database design report
- Developer utilities

Database Design Report

A *database design report* is a detailed text-based report of the schema of one or more open database files. At minimum, the report includes a list of every table in the database. However, it can optionally include information about many other resources.

A design report can be saved as a HTML or XML file and can be used to:

- Troubleshoot problems with the database structure, making it easier to locate missing references, broken relationships, and other problems.
- Locate obsolete elements and establish with certainty that they are not in use in some obscure part of the database.
- Gather statistical information about the structure of the database, including the number of various resources and file option settings.
- Provide detailed technical documentation of the database structure at a certain point in time, which can be consulted in the event a resource or the entire database is permanently lost and must be manually re-created.

As your databases increase in complexity, you will find the design report an invaluable tool.

Generating a Design Report

To generate a design report, open a database and select the **Tools > Database Design Report** menu. This will open the **Database Design Report** dialog, shown in Figure 41-1.

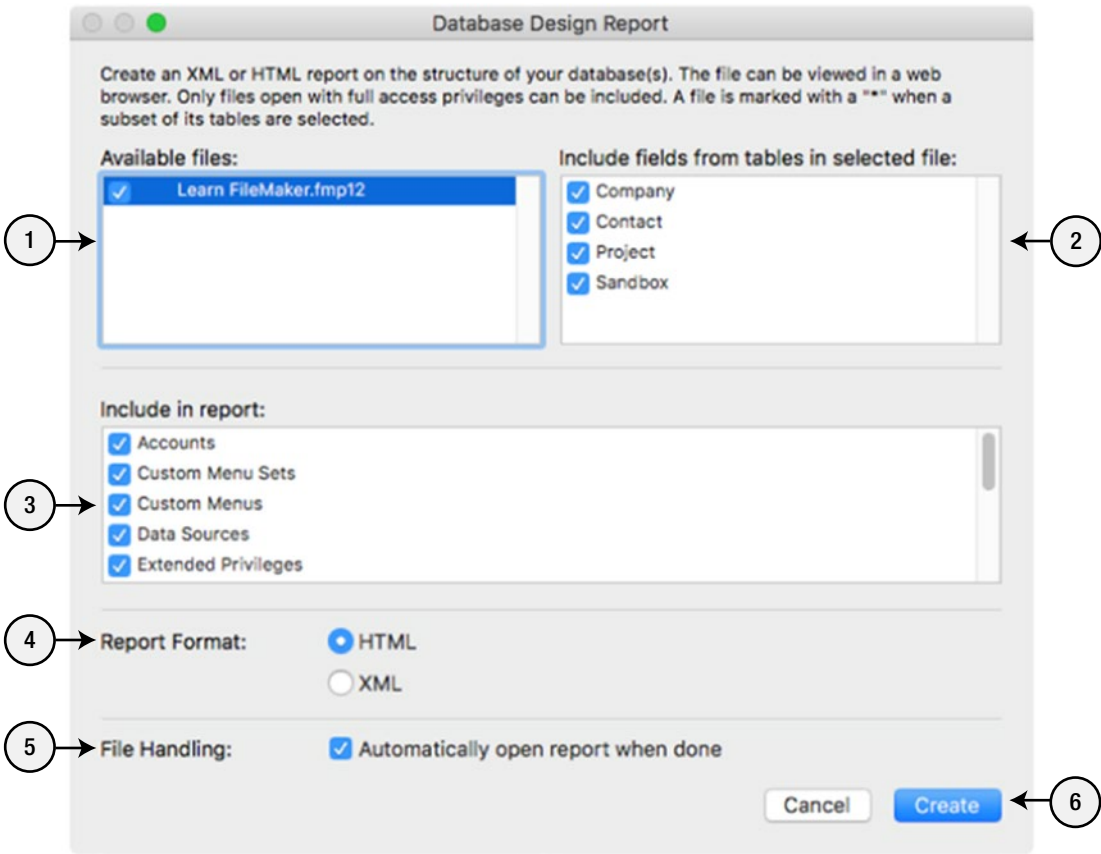


Figure 41-1. The Database Design Report dialog

The following controls are available on the Database Design Report dialog:

1. Available Files — This area lists all the database files currently open. The check box next to each file will be enabled by default, but can be disabled to exclude it from the report.
2. Include fields from tables in select file — This area lists every table in the selected database. The check box next to each table will be enabled by default, but can be disabled to exclude it from the report.
3. Include in report — This area lists the optional resources that can be included in the report. The check box next to each will be enabled by default, but can be disabled to exclude it from the report. The available options are:
 - Accounts
 - Custom Menu Sets
 - Custom Menus
 - Data Sources
 - Extended Privileges

- File Access
 - Functions
 - Layouts
 - Privileges
 - Relationships
 - Scripts
 - Value Lists
4. Report format — Select a format for the report, either HTML or XML.
 5. File handling — Enable the check box to have the report automatically open after it is generated.
 6. Buttons — Click the OK button to generate the report and close the dialog. Click Cancel to close the dialog without generating a report.

When clicking OK, a Save Report dialog will open asking for a name for the report and a folder directory in which it should be saved.

Exploring a Design Report

The design report generated will appear in the folder specified as a set of files specific to which format option you chose: HTML or XML.

Opening a HTML Design Report

When a design report is saved as HTML, the folder will contain several files as shown in Figure 41-2.

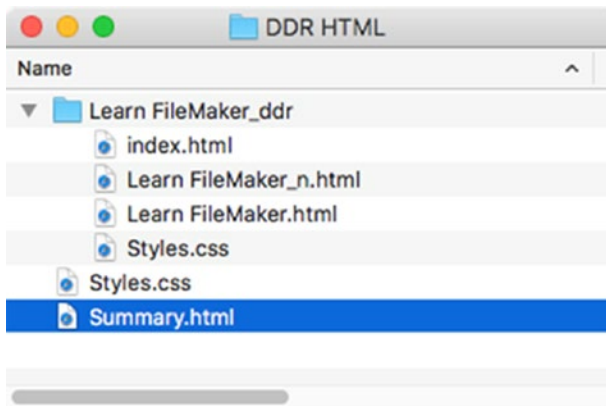


Figure 41-2. The Database Design Report folder when saved as HTML

The HTML report folder contains the following items:

- `Summary.html` — This file contains an overview of the report as a summary of each database included in the report and should be the starting point for exploring it.
- `Styles.html` — This file contains cascading style sheet information for how the report is formatted.
- `Folder(s)` — Each database file that was included in the report will be represented by a folder named `<Database>_ddr`. The files in each folder are:
 - `Index.html` — This file contains an overview of the report for the file represented by the folder and can be opened to directly access it.
 - `<database>_n.html` — This file contains navigation controls that are included in a sidebar frame of the `Index.html` page.
 - `<database>.html` — This file contains the body of the database report that is included in a body frame of the `Index.html` page.
 - `Styles.html` — This file contains cascading style sheet information for how the report is formatted.

To get started, double-click on the `Summary.html` file to open the report summary page, shown in Figure 41-3.

FileMaker Database Design Report

Report Created: 3/10/2017 at 10:33:35 AM

Report Overview

File Name	Tables	Relationships	Layouts	Value Lists	Scripts	Accounts	Privilege Sets	Extended Privileges	Custom Functions	Data Sources	Custom Menu Sets	Custom Menus
Learn FileMaker.fmp12	4	3	6	2	1	4	3	7	0	1	0	0

Figure 41-3. The summary page includes an overview of every database included in the report

The database design report overview page lists each database included in the report with a count of all the resources selected for inclusion on the dialog. Each column represents one type of resource and lists the quantity included in the report. Click these to open the database report and navigate down to that resource. Alternatively, click on the file name to open the report from the top, as shown in Figure 41-4.

Learn FileMaker.fmp12

Overview		File Options	
Tables	4	Default custom menu set	[Standard FileMaker Menus]
Relationships	3	When opening file	
Layouts	6	Minimum allowed version	12.0
Scripts	1	Login using	Account Name; Account= Admin
Value Lists	2	Allow user to save password	Off
Custom Functions	0	Require iOS passcode	Off
Accounts	4	Switch to layout	Off
Privilege Sets	3	Hide all toolbars	Off
Extended Privileges	7	Script triggers	
File Access (in / out)	0 / 0	OnFirstWindowOpen	Off
FileMaker Data Sources	1	OnLastWindowClose	Off
ODBC Data Sources	0	OnWindowOpen	Off
Custom Menu Sets	0	OnWindowClose	Off
Custom Menus	0	OnFileAVPlayerChange	Off
		Thumbnail Settings	
		Generate Thumbnails	On; Temporary

Tables

Table Name	Statistics	Occurrences in Relationship Graph
Sandbox	8 fields defined, 3 records	Sandbox
Company	12 fields defined, 2 records	Project Company, Contact Company, Company
Contact	17 fields defined, 525 records	Contact, Company Contact
Project	12 fields defined, 0 record	Project

Fields

[Back to Overview](#)

[Overview/File Options](#)

[Tables](#)

[Fields](#)

[Base Directories](#)

[Relationships Table](#)

[Occurrences Details](#)

[Layouts](#)

[Value Lists](#)

[Scripts](#)

[Security](#)

[Accounts](#)

[Privilege Sets](#)

[Extended Privileges](#)

[File Access](#)

[Data Sources](#)

Figure 41-4. The HTML DDR for a file

The database report is a single scrollable page that begins by listing a summary overview of all the resource counts and file options, and it is then divided into sections for every resource that was included. In the sidebar on the left is a list of links that allow navigation to a specific section of the report as well as navigating back to the summary overview.

Opening an XML Design Report

When a design report is saved as XML, the folder will contain several files as shown in Figure 41-5.

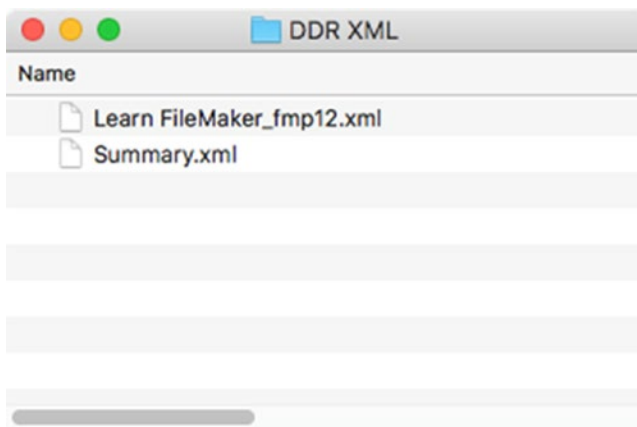


Figure 41-5. The Database Design Report folder when saved as XML

At its root level, the XML report folder contains the following items:

- `Summary.xml` — This file contains a summary of each file included in the report.
- `<database>_fmp12.xml` — Each database file that was included in the report will be represented by a single XML file. These files contain all the report information in XML format.

Introducing Professional DDR Tools

Many companies sell professionally developed products that can import an XML design report and present the material in a more polished interface. These are a great option for developers who do a lot of analysis of databases or just want a better viewing experience for design reports. A few prominent examples are listed here for your convenience.

BaseElements

BaseElements is a great analysis tool for traversing the structure of a database. The solution is available as either standard FileMaker files or a stand-alone runtime application for either macOS or Windows. Both have full layout access, and you can link to BaseElements Data Tables from your database's relationship graph and access the information in scripts. The product includes an open source BaseElements plug-in with many useful functions.

For more information visit: <http://www.baseelements.com>.

FMPerception

Geist Interactive sells a product called *FMPerception*, a utility for searching, analyzing, and maintaining FileMaker databases. The product uses a progressive analysis technique to quickly provide current and accurate information about a database's structure. Some features include:

- Provides a high-level overview of a database and presents a "Database Report Card" that summarizes the complexity of a database and lists any potential problems that were detected.
- Boasts a fast DDR import process that is nearly instantaneous.
- A graphical "call chain visualization" illustrates the flow of script calls from buttons to scripts to subscripts, helping trace the flow of complex functionality.
- Identifies layout objects that might slow down performance.
- Performs a security audit to identify vulnerabilities such as missing passwords or scripts that run with full access privileges.

For more information visit: <https://www.geistinteractive.com>.

InspectorPro

Beezwx DataTools, Inc sells a product called *InspectorPro*. The product is an advanced diagnostics, analysis, and visualization tool that is optimized for FileMaker versions 12-15. The product can help you quickly locate references and resources, boasts over 115 different areas where it normalizes analysis information, and can even automatically detect problems with a database structure.

For more information visit: <https://www.beezwx.net>.

Developer Utilities

Exclusive to FileMaker Pro Advanced is a set of developer utilities that can create a customized copy of a database with structural modifications that allow the following sharing options and enhancing security:

- Create runtime solution application(s)
- Remove admin access from files permanently
- Enable kiosk mode for non-admin accounts
- Databases must have a FileMaker file extension
- Create error log for any processing errors
- Enable database encryption (or Re-encrypt files)
- Remove database encryption

These options, which will be explained later in this chapter, can be selected separately or in unison, although some restrictions apply.

Exploring the Developer Utilities Dialog

These tools can be accessed by selecting the Tools > Developer Utilities menu item. This opens the Developer Utilities dialog, shown in Figure 41-6.

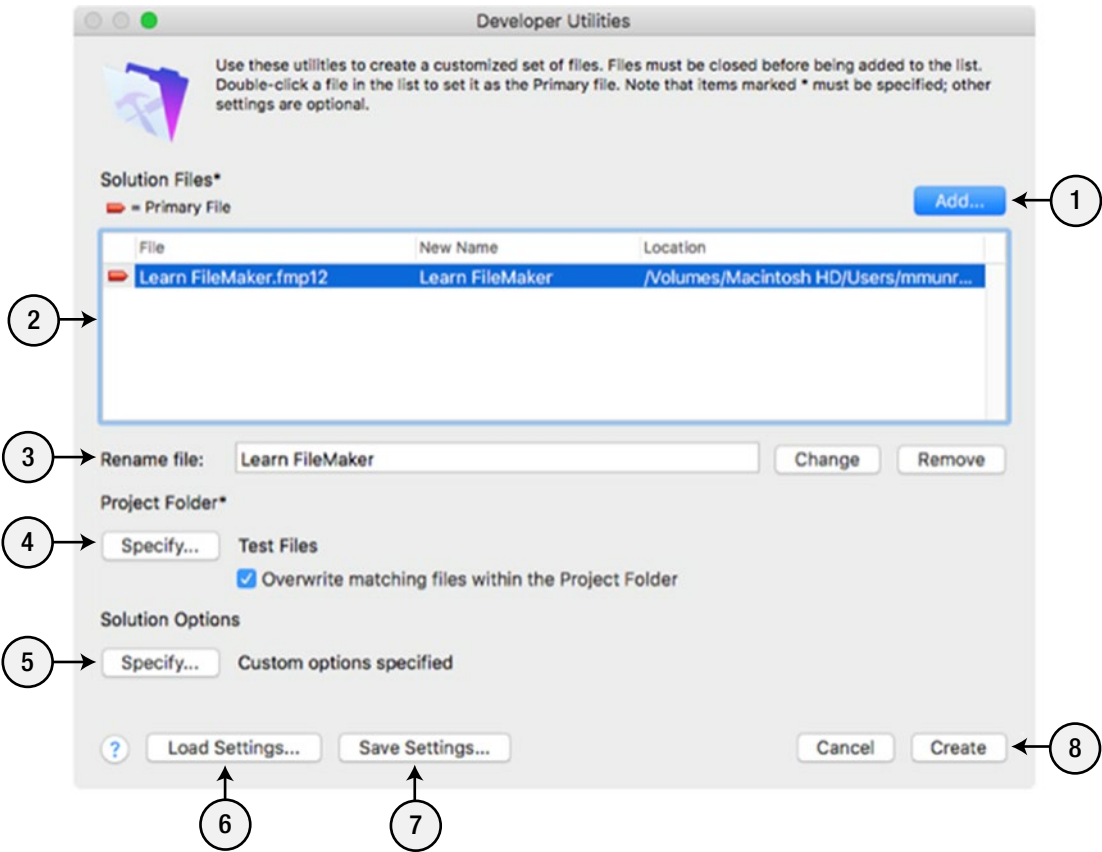


Figure 41-6. *The Developer Utilities dialog*

The Developer Utilities dialog contains the following controls:

1. **Add** — Click this button to select file(s) to add to the list of solution files.
2. **Solution Files** — This area lists all the files selected using the Add button that will be processed and saved into new modified copies using the utility options selected. When processing multiple files, some utilities require a file to be designated as the Primary File, indicated with a red arrow and modifiable by double-clicking on a different file.

3. **Rename File** — This area is used to specify a name for the output that is different from the file selected as primary. This is the recommended method for renaming database files in a multi-file solution since it will rename the specified files *and* update all external file references to them within other files.
4. **Project Folder** — Click this **Specify** button to select a folder in which to save the output. Enable the check box to allow FileMaker to automatically replace existing files in the project folder with the same name as new files it is saving.
5. **Solution Options** — Click this **Specify** button to select the desired options that will determine the modifications to the solution files (described later in this chapter).
6. **Load Settings** — Click this button to replace the current dialog settings with those stored in a selected file.
7. **Save Settings** — Click this button to save files containing all the settings configured in the dialog so that they can be loaded for quick reuse in the future.
8. **Buttons** — Click the **OK** button to begin processing the files. Click **Cancel** to close the dialog without performing any functions.

Specifying Solution Options

Clicking the **Specify** button for **Solution Options** on the **Developer Utilities** dialog to open a **Specify Solution Options** dialog in which you can select and configure the desired utilities that will be used to modify the selected files.

Create Runtime Solution Application(s)

The **Create runtime solution application(s)** option will bind the database files into a stand-alone runtime database application that does not require FileMaker Pro to be used. This allows you to freely distribute your solution to as many users as you want.

■ **Caution** The runtime application can't access databases across the network so this option is limited to single-user database systems.

Select the option in the **Specify Solution Options** dialog, as shown in Figure 41-7.

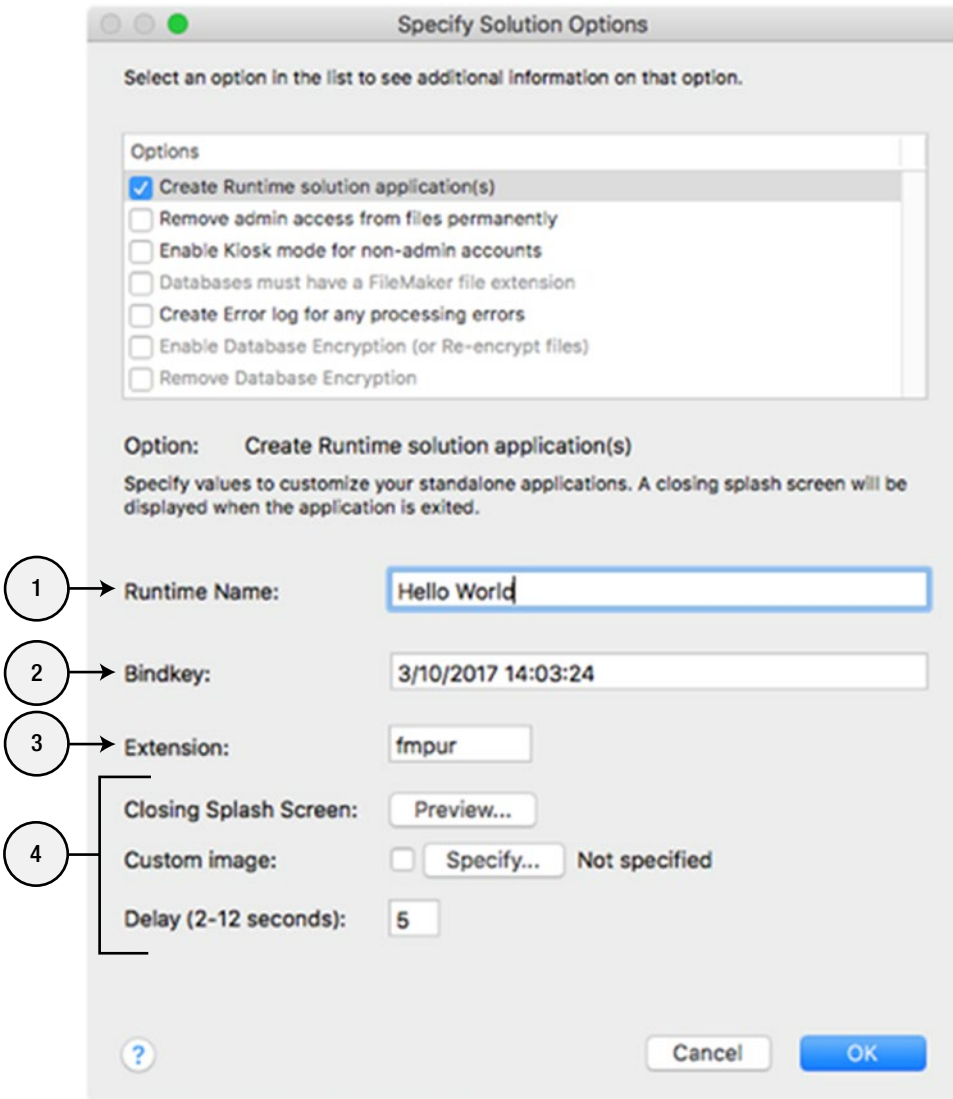


Figure 41-7. The Specify Solution Options dialog for creating a runtime solution

The Specify Solution Options dialog contains the following controls:

1. Runtime Name — Enter a name for the runtime application.
2. Bindkey — This information is used to bind the runtime application to the database files so that they will only open with the appropriate runtime application. This is useful in situations where multiple runtime applications are used on the same computer. The default data and time entry provides a unique key that won't ever conflict.

3. **Extension** — The extension entered will be used on database files in place of the standard `.fmp12` to link the files to the runtime app. The default extension of `fmpur` can be replaced with a custom extension if it is unique and doesn't conflict with other applications installed on the user's computer.
4. **Splash Screen Options** — When the runtime application is closed, a splash screen will appear, which can be customized using the controls:
 - **Preview** — Click this button to see a preview of the splash screen that will appear when the runtime application is closed.
 - **Custom Image** — Click the Specify button to select an image file to use as a custom splash screen.
 - **Delay** — Enter a number within the range of 2-12 to specify the number of seconds the splash screen should remain open.

Click OK to close the Specify Solution Options dialog and return to the Developer Utilities dialog. Then, click OK to run the utility. A folder will be created in the Project Folder containing the runtime app, the file(s) selected, and other resources, as shown in Figure 41-8.

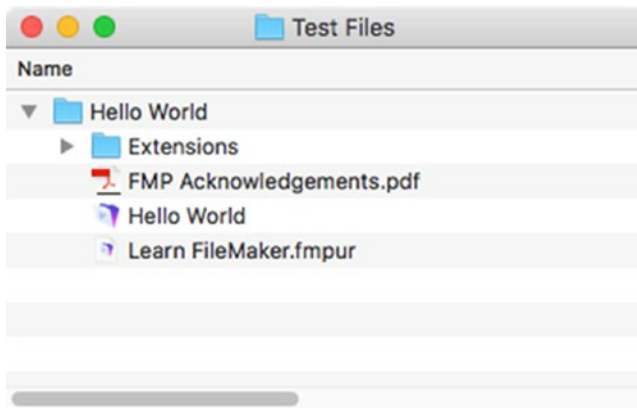


Figure 41-8. An example of a runtime solution folder structure

The folder will contain the runtime application and the database file(s) selected. Double-click the application to launch the solution.

■ **Note** The application will only run on the same operating system upon which it was created.

Assigning Files to the Runtime Application in macOS

When the runtime solution is saved on a macOS computer with the default extension of `fmpur`, there is an additional step that must be taken to correctly map the files to always open with the custom runtime application when double-clicked on directly. Opening the runtime application will work fine and automatically open the primary database file. However, when double-clicking directly on one of the database files, it should open the runtime application and then open the file but instead macOS will present a cryptic error message about a missing primary file “DEBUG,” as shown in Figure 41-9.

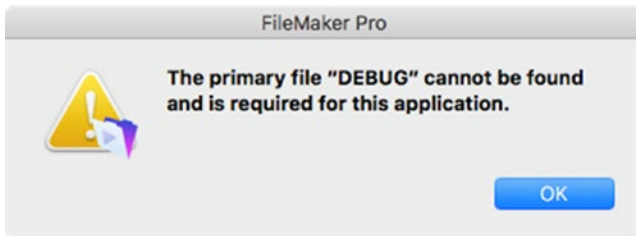


Figure 41-9. The error dialog when double-clicking directly on a database file that is part of a runtime solution

The problem is that the FileMaker Pro Advanced folder contains an application called Runtime located in the following folder:

/FileMaker Pro Advanced/FMI Runtime Resources/Internal/Runtime Libraries/

By default, any file with an extension of `fmpur` will automatically be linked to open with this application and generate the error message. To change this on macOS, select all the database files and then open the contextual menu and open the **Always Open With** submenu. Select the custom runtime app, as shown in Figure 41-10.

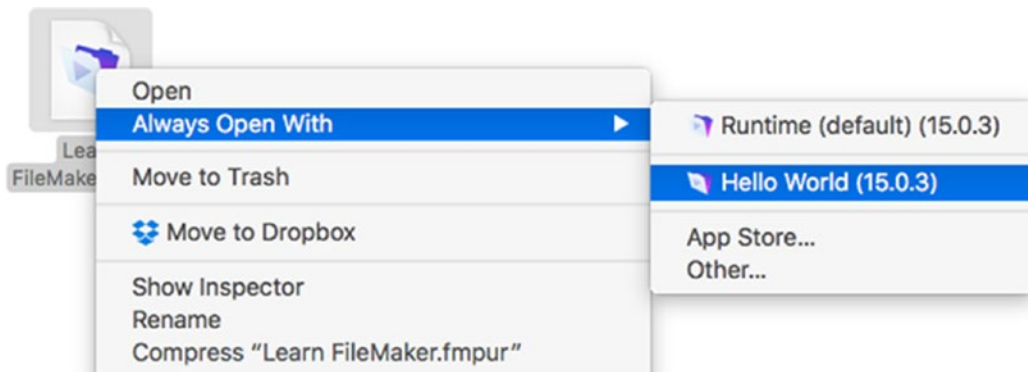


Figure 41-10. Connecting runtime database files to automatically open with the custom runtime application instead of FileMaker’s default

Once this process is complete the files will always open with the custom runtime application. However, this must be performed on *every* computer upon which the solution will be installed.

■ **Tip** Using a custom extension will avoid the necessity of this extra step.

Remove Admin Access from Files Permanently

The `Remove admin access from files permanently` option will completely remove all admin access within the file(s) and remove visible calculation formulae. This prevents *anyone* from opening the file as a developer and making changes to the structure. While this can be a useful method to secure a file, it should be used with caution and while retaining a safe unmodified backup file.

Enable Kiosk Mode for Non-Admin Accounts

The `Enable Kiosk mode for non-admin accounts` option will modify and save the file(s) with full screen mode and no menu bar enforced for all non-admin access. This is a great option for creating kiosks where you don't want to allow users access to anything except the interface controls you make accessible on layouts.

Before using this option consider a few requirements:

- The database must have at least one non-admin user account defined, which must be used to open in kiosk mode.
- Once in kiosk mode, the user's *only* access to navigate within the file or to close the file will be through the controls you provide on layouts. There will be *no* menu bar, and even access to operating system functions such as force quit or toggle applications will be completely disabled.
- When adding a button or method for quitting the database, be sure to include some sort of authentication process to exclude non-authorized users from quitting the solution and accessing the underlying operating system.

■ **Caution** If you don't provide a button on the layout that will quit the application or close the database, the *only* option will be to perform a hard restart of the computer that may lead to file damage.

Databases Must Have a FileMaker File Extension

This feature is deprecated and scheduled to be removed from the interface in the near future.

Create Error Log for Any Processing Errors

The `Create Error log for any processing errors` option will generate a log file in a specified folder containing any errors that occur while processing other utility functions. This option can optionally be selected with any other utility function but is *required* by the two encryption features.

Once selected the only available option for this feature is a `Specify` button that, when clicked, opens a `Specify a File` dialog in which you can point to any folder and assign a name for the log file. The default is a file named `LogFile.txt` that will be saved in the specified `Project Folder`.

Exploring Encryption Features

Encrypting a database file adds a more powerful layer of protection above that of the standard security features (Chapter 40). An encrypted database protects a file from unauthorized access while it is stored on disk. The password assigned during the encryption process *must* be entered to open the file with a copy of FileMaker or to host the file on a FileMaker Server. Once hosted, only the standard security account and password need be entered for a user to gain access. The encryption is a protection against someone gaining access to an unhosted copy of the file and trying to open it offline, which will be impossible without the encryption password.

Caution When encrypting a database, be sure to keep the password safely stored somewhere. There is no possibility of recovery so, if you lose the password, you will be unable to access the file.

The Developer Utilities dialog contains two encryption options:

- Enable database encryption (or Re-encrypt files)
- Remove database encryption

Enable Database Encryption (or Re-encrypt Files)

The Enable Database Encryption (or Re-encrypt files) option will save an encrypted copy of the file(s) encrypted that will require a decryption password to be entered to open the files directly or to host them on FileMaker Server.

To encrypt the selected file(s), select the option in the Specify Solution Options dialog and enter the required information, shown in Figure 41-11.

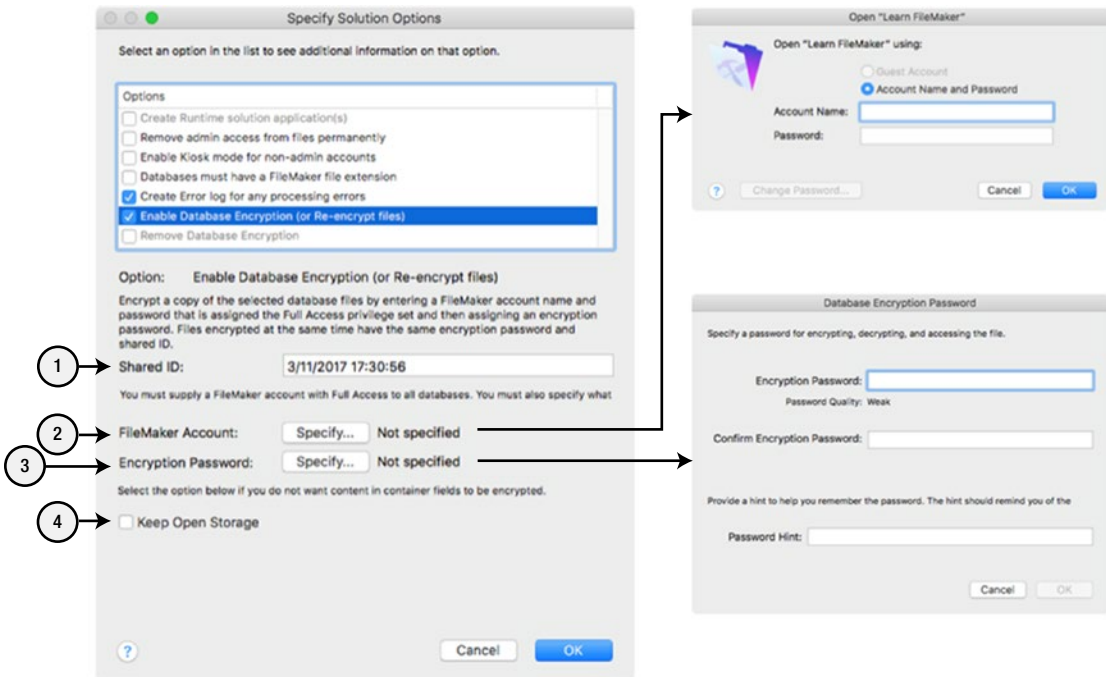


Figure 41-11. The process of encrypting a database

The encryption process involves the following steps:

1. **Shared ID** — This text contains a case-sensitive string of up to 32 characters and is used to link all the database files in a multi-file solution. When one file is opened with the encryption password, that file can then access any other file that has been saved with the *same* Shared ID and encryption password, without prompting the user for the encryption password repeatedly.
2. **FileMaker Account** — You must enter developer credentials before you can encrypt the file(s). Click the adjacent **Specify** button to enter the **Account Name** and **Password** proving that you have [Full Access] to the file. These credentials should be common to *all* the selected **Solution Files**.
3. **Encryption Password** — An encryption password must be entered to secure the file(s), which will be used for all future access and hosting. Click the adjacent **Specify** button to open the **Database Encryption Password** dialog and enter and confirm a password.
4. **Keep Open Storage** — Enable this check box to maintain externally stored container files in an unencrypted state and allow users access freely through the folder directory.

Click **OK** to close the **Specify Solution Options** dialog and return to the **Developer Utilities** dialog. Then, click **OK** to run the utility. The encrypted files will appear in the specified **Project Folder**.

When opening an encrypted file on the desktop, FileMaker will present two dialogs: first the **Open Encrypted Database** prompt, shown in Figure 41-12; and then the standard login prompt for **Account name** and **Password**. When opening an encrypted file for hosting on a FileMaker Server, only the first of these will appear.



Figure 41-12. The *Open Encrypted Database* dialog

Remove Database Encryption

The **Remove Database Encryption** option will save a copy of the file(s) in an unencrypted state, restoring normal operating behavior and access methods.

To remove encryption from the selected file(s), select the option in the **Specify Solution Options** dialog and enter the required information, shown in Figure 41-13.

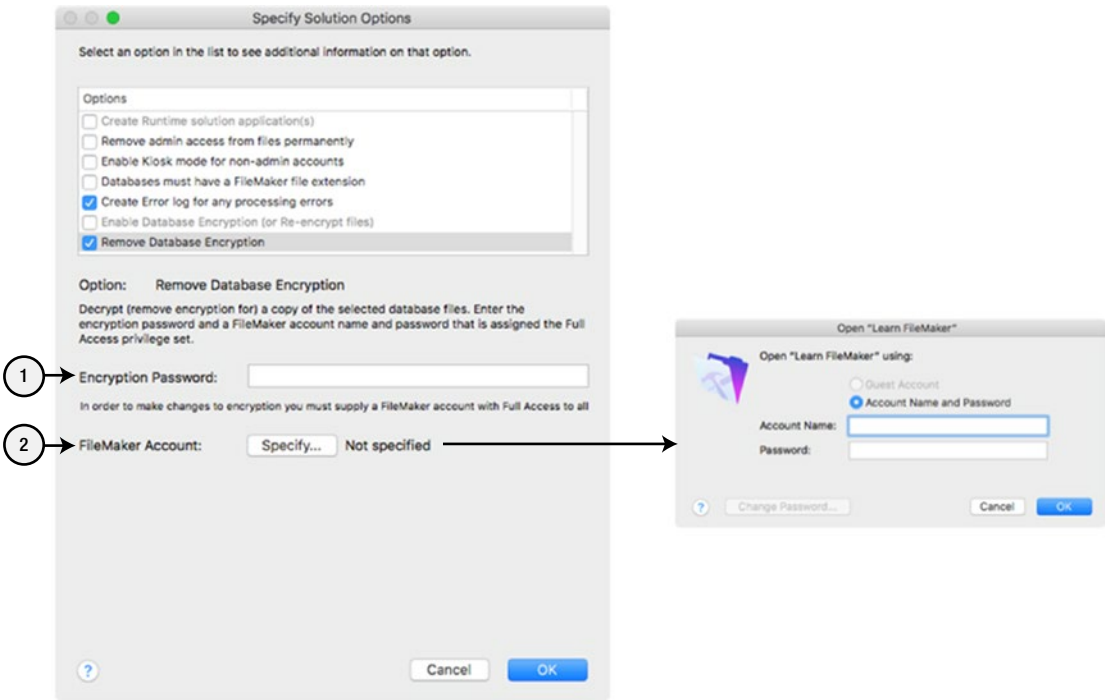


Figure 41-13. The process of removing encryption from a database

The process of removing encryption involves the following steps:

1. **Encryption Password** — You must enter the encryption password for the file(s) to prove that you have access to them.
2. **FileMaker Account** — You must enter developer credentials before you can decrypt the file(s). Click the adjacent **Specify** button to enter the **Account Name** and **Password** proving that you have [Full Access] to the file. These credentials should be common to *all* the selected **Solution Files**.

Click **OK** to close the **Specify Solution Options** dialog and return to the **Developer Utilities** dialog. Then, click **OK** to run the utility. The decrypted files will appear in the specified **Project Folder**.

Now, when opening the file locally, only the standard login prompt for **Account name** and **Password** will be presented.

Summary

In this chapter, we discussed the **Database Design Report** and various **Developer Utilities** that are available through the **FileMaker Advanced Tools** menu.

CHAPTER 42



Getting Connected

Reach out to connect with the author, other readers, and the wider FileMaker community to ask questions, find tips, and explore other resources.

Connect with Mark Munro

Join Mark's professional network on LinkedIn:

<https://www.linkedin.com/in/mark-conway-munro-1521595/>

For professional help with a database or workflow automation project, contact Mark through the *Write Track Media* website:

<http://www.wtmedia.com>

Follow *Write Track Media* on social media:

Facebook	https://www.facebook.com/writetrackmedia/
Twitter	https://twitter.com/WriteTrackMedia
LinkedIn	https://www.linkedin.com/company/1349330

Learn FileMaker Pro 16 Official Sites

Visit the official *Learn FileMaker Pro 16* book website at:

<http://www.learn-filemaker.com>

To ask questions, chat, or share tips with the author, readers, users, and other developers, join the Facebook group:

<https://www.facebook.com/groups/LearnFileMaker/>

To find the book on the Apress website, visit:

<http://www.apress.com/us/book/9781484228623>

FileMaker Inc.

Keyboard shortcuts:

macOS	https://www.filemaker.com/support/product/docs/12/fmp/fmp12_keys_mac.pdf
Windows	https://www.filemaker.com/support/product/docs/12/fmp/fmp12_keys_win.pdf

The FileMaker website is a great place to start looking for help and information.

Home	http://www.filemaker.com
Support	http://www.filemaker.com/support/
Updaters	http://www.filemaker.com/support/downloads/
Community	https://community.filemaker.com/welcome
Solutions	https://solutions.filemaker.com/made-for-filemaker

Help guides and documentation:

Help	https://www.filemaker.com/help/16/fmp/en/
ODBC & JDBC	https://fmhelp.filemaker.com/docs/16/en/fm16_odbc_jdbc_guide.pdf
SQL Reference	https://fmhelp.filemaker.com/docs/16/en/fm16_sql_reference.pdf
Error Codes	http://www.filemaker.com/help/16/fmp/en/index.html#page/FMP_Help%2Ferror-codes.html

Database Design Report (DDR) Analysis Tools

The Database Design Report analysis tools mentioned in Chapter 41:

Base Elements	http://www.baseelements.com
FM Perception	https://www.geistinteractive.com
Inspector Pro	https://www.beezwax.net

Plug-Ins

The manufactures of the plug-ins mentioned in Chapter 38:

Troi Automatisering	http://www.troi.com
Productive Computing	http://www.productivecomputing.com
360 Works	http://www.360works.com

Communities

Brian Dunning was the Technical Editor of *FileMaker Advisor Magazine* from 1997-2016. His website is a great resource for finding all sorts of resources, custom functions, charts, and more.

<https://www.briandunning.com>

FileMaker World is full of information and resources for developers.

<http://www.filemakerworld.com>

Join discussions on FileMaker topics at Reddit:

<https://www.reddit.com/r/filemaker/>

FMForums is a worldwide community with numerous forums with thousands of posts.

<http://fmforums.com>

Listen to a FileMaker podcast by Matt Navarre and Matt Petrowsky:

<http://filemakertalk.com>

ISO FileMaker Magazine has a lot of video tutorials.

<https://www.filemakermagazine.com>

Technology Sites

These are the various technology sites mentioned throughout the book, summarized here for your convenience.

For a complete list of available Unicode numbers visit:

<http://www.ssec.wisc.edu/~tomw/java/unicode.html>

The Internet Engineering Task Force (IETF) Request for Comment (RFC) page:

<https://www.ietf.org/rfc.html>

For more technical information on JSON, visit:

<http://www.json.org>

A JSON validation tool to confirm manually typed objects are formatted properly:

<http://www.jsonlint.com>

For more information about Base64 visit:

<https://www.base64decode.org>

Function Background Information

These are the various function information sites mentioned throughout the book, summarized here for your convenience.

Learn about *standard deviation* here:

https://en.wikipedia.org/wiki/Standard_deviation

Learn about *variance* here:

<https://en.wikipedia.org/wiki/Variance>

Learn about *future value* here:

https://en.wikipedia.org/wiki/Future_value

Learn about *net present value* here:

https://en.wikipedia.org/wiki/Net_present_value

Learn about *present value* here:

https://en.wikipedia.org/wiki/Present_value

Learn about *Coordinated Universal Time (UTC)*, here:

https://en.wikipedia.org/wiki/Coordinated_Universal_Time

For more information on *iBeacons*, visit:

<https://developer.apple.com/ibeacon/>

Learn about *Universally Unique Identifiers (UUID)*, here:

https://en.wikipedia.org/wiki/Universally_unique_identifier

Index

■ A

- Abs function, 328
- Accounts, 856
 - credentials dialog, 857
 - external account
 - authentication, 858
 - server dialog, 858
 - edit dialog, 856
 - external account
 - group membership, 861
 - groups (macOS), 859
 - user account, 860
 - set password dialog, 857
 - steps, 856
- Acos function, 381
- Aggregate function reference, 357
 - average, 358–360
 - count, 360–361
 - list, 362–363
 - Max function, 363–365
 - Min function, 365–366
 - StDev function, 367–368
 - StDevP function, 369–370
 - Sum function, 370–372
 - Variance function, 372–374
 - VarianceP function, 374–376
- Apple, 4
- AppleScript, 845
 - database, 847
 - dictionary, 845
 - field value, 848
 - find records, 848
 - FMP protocol, 847
 - found set, 849
 - layouts, 848
 - local directory, 847
 - open database, 847
 - table, 849
 - tasks, 846
 - tell FileMaker statement, 845
- Application menu, 29

- Application preferences
 - application, 25
 - file cache settings, 26–27
 - general settings, 23–24
 - layout options, 25
 - layout settings, 25
 - memory, 26
 - user interface options, 24
 - user name, 25
- Application programming interface (API), 13, 160
- Asin function, 382
- Atan function, 382
- Average function, 358
 - with local fields, 358–359
 - with related fields, 359–360

■ B

- Back-end data system, 10
- Base64Decode function, 343
- Base64Encode function, 344
- Base64EncodeRFC function, 344
- BaseElements, 884
- Browse mode
 - container field, 42
 - database window, 48
 - record content area, 42
 - status toolbar
 - customization, 53–56
 - default items, 50–52
 - toggle, 50
 - summary field, 42
 - view menu, 35
 - web viewers, 43
- Button controls
 - button, 621
 - contextual menu, 628
 - icon label, 623
 - labeling options, 622
 - script, 625–627
 - setup dialog, 622
 - single step, 624

Button controls (*cont.*)

- button bar, 632
 - bar orientation control, 633
 - conversion, 634
 - segment controls, 634
 - setup options, 633
- popover button
 - interface, 628, 630
 - setup options, 631
 - toolbar control, 629
- steps, 621

C

- Calendar application, 5
- Card window, 788
- Case function, 386
- Char function, 292
- Charts
 - calculated data, 657
 - chart types, 655
 - formats, 655
 - found set, 658
 - pie chart, 657–658
 - setup dialog, 656
 - table view, 660
- Choose function, 386, 387
- Clarix Corp., 4
- Code function, 293
- Combination function, 328
- Communities, 896
- Comparative operators, 229
- Comparison operators
 - equal to, 265
 - greater than or equal to symbol, 266
 - greater than symbol, 266
 - less than or equal to symbol, 267
 - less than sign <, 267
 - not equal to, 266
- Conditional formation, 672
 - condition formula, 674
 - dialog options, 673
 - format settings, 675
 - formula-driven conditions, 674
 - invoice status field, 675
 - menu item, 673
 - value-driven conditions, 674
- Consistency check, 137–138
- Constant-only formula, 279
- Constrain Found Set command, 78
- Constrain Found Set function, 69
- Container fields, 175
- Container function
 - Base64Decode, 343
 - Base64Encode, 344

- Base64EncodeRFC, 344
- CryptAuthCode, 345
- CryptDecrypt, 347
- CryptDecryptBase64, 347
- CryptDigest, 347
- CryptEncrypt, 346
- CryptEncryptBase64, 346
- GetContainerAttribute, 348
- GetHeight, 351
- GetThumbnail, 352
- GetWidth, 352
- HexDecode, 353
- HexEncode, 353
- reference, 343
- TextDecode, 354
- TextEncode, 354
- VerifyContainer, 355
- Containers
 - binary data, 233
 - externalstorage (*see* External storage)
 - extracting files, 242
 - insert files
 - copy and paste, 235
 - menu option, 234
 - methods, 233
 - storage options, 235
 - actual file, 235
 - directories, 236
 - managed directory, 237
 - reference, 235
- Content view
 - form, 49
 - list, 49
 - table, 49
- Contextual menus
 - calculation formulas, 43
 - categories, 41
 - defined, 40
 - fields, 42
 - record content area, 42
 - web viewers, 42
 - window components, 43
- Contextual paths, 152
- Control styles
 - checkbox set, 605
 - concealed edit box, 610
 - drop-down calendar, 607
 - drop-down list, 602
 - edit box, 598
 - pop-up menu, 597, 599
 - radio button set, 606
 - steps, 596
 - unselecting values, 607
- Cos function, 382

- Count function, 360
 - with local fields, 360–361
 - with related fields, 361
- CryptAuthCode function, 345
- CryptDecryptBase64 function, 347
- CryptDecrypt function, 347
- CryptDigest function, 347
- CryptEncryptBase64 function, 346
- CryptEncrypt function, 346
- Current Date function, 64
- Current Time function, 64
- Current User Name function, 64
- Custom functions, 461
 - accessing fields, 471
 - adding parameters, 467
 - date parameter, 467–468
 - day requested parameter, 468–469
 - default current date, 469
 - importance of thorough testing, 470–471
 - defining, 461
 - exploring dialogs, 462
 - Edit Custom Function dialog, 464–465
 - importing, 463
 - managing, 462–463
 - recursive, 472
 - limitations, 473
 - local variables across iterations, 474
 - simple, 472–473
 - test code inside a function, 474–475
 - simple custom function, 465–466

D

Database

- calendar application, 5
- data structure, 13
- defined, 5
- design report, 15
- encryption, 16
- host, 16
- integrated file architecture, 10–11
- Launch Center, 59
- layout, 501
- mail application, 5
- manage submenu, 32
- managing favorite, 60
- multitier architecture, 9
- opening an existing, 58
- presentation tier, 9
- process logic tier, 10
- relational, 7
- script, 8, 14
- starter solution, 46

- stored data tier, 10
- structure, 6
- table, 6
- user interface, 7–8, 14
- Database Design Report (DDR)
 - analysis tools, 896
 - developer utilities, 885
 - dialog, 885
 - encryption, 892
 - error log, 891
 - FileMaker file extension, 891
 - macOS, 889
 - non-admin accounts, 891
 - remove admin access, 891
 - runtime solution application(s), 887
 - dialog menu, 880
 - HTML design report, 881
 - HTML/XML file, 879
 - professional DDR tools
 - BaseElements, 884
 - FMPerception, 884
 - InspectorPro, 885
 - report menu, 879
 - text-based report, 879
 - XML design report, 883
- Database file
 - creating, 119–120
 - icon settings, 132
 - maintenance functions
 - advanced recovery options, 139
 - avoiding design, 140–142
 - avoiding risky behavior, 145
 - consistency check, 137–138
 - copy file, 146
 - development standards, 145
 - file damage, 144–145
 - hardware and software, 144–145
 - locate damage, 146
 - perform maintenance, 145
 - recovering file, 138–139
 - restrain file size, 143–144
 - save copy as, 136
 - training deficiencies, 140–142
 - transferring record data, 147
 - troubleshooting process, 146
 - using FileMaker server, 145
 - open settings, 131
 - options, 130
 - records
 - explicit dialogs, 142
 - forgetting to commit, 141
 - multiple dialogs, 142
 - overwriting an existing, 141
 - require explicit confirmation, 142
 - restrict access, 142

Database file (*cont.*)

- set limits, 142
- user efficiency, 142
- vague dialogs, 141

- resources, 13
- script triggers settings, 135
- spelling settings, 133
- text settings, 133–134

DatabaseNames function, 446

Database window

- browse mode, 48
- content area, 47
- content views, 49
- find mode, 48
- layout mode, 49
- preview mode, 48
- status toolbar, 47

Data entry

- field content modification
 - Cut, Copy, Paste, Undo, and Redo, 63
 - drag and drop, 63
 - Insert functions, 63–64
 - typing, 63
- field focus, 62
- field navigation, tab order, 62
- handling, 134
- record opening, 62

Data structure, 13

Date functions

- built-in function, 335
- date object, 336
- day, 336
- DayName, 336
- DayNameJ, 337
- DayOfWeek, 337
- DayOfYear, 337
- month, 337
- MonthName, 338
- MonthNameJ, 338
- WeekOfYear, 338
- WeekOfYearFiscal, 339
- Year, 339
- YearName, 339

Debugging scripts

- activation, 795
- breakpoints, 798
- definition, 794
- interface
 - data viewer window, 797
 - status toolbar items, 797
 - tools menu, 795
 - window, 796
- troubleshooting techniques, 793
- uses of, 794

Default menu bar

- application menu, 29
- disabled items, 28
- edit menu, 34
- file menu, 30
- format menu, 36
- help menu, 40
- insert menu, 36
- records menu, 37
- scripts menu, 38
- tools menu, 38
- view menu, 35–36
- window menu, 39

Default windows

- Launch Center
 - hosts tab, 23
 - solutions tab, 23
- started screen, 21

Degrees function, 382

Design functions, 445

- DatabaseNames function, 446
- FieldBounds function, 446
- FieldComment function, 447
- FieldIDs function, 447, 448
- FieldNames function, 448
- FieldRepetitions function, 448, 449
- FieldStyle function, 449
- FieldType function, 449, 450
- GetNextSerialValue function, 450
- LayoutIDs function, 450, 451
- LayoutNames function, 451
- LayoutObjectName function, 451, 452
- RelationInfo function, 452, 453
- ScriptIDs function, 453
- ScriptNames function, 453
- TableIDs function, 453, 454
- TableNames function, 454
- ValueListIDs function, 454
- ValueListItems function, 455
- ValueListNames function, 455, 456
- WindowNames function, 456

Div function, 329

Dynamic guides, 665

■ E

Edit layout button, 53

Edit menu, 34

Encryption

- database dialog, 893
- re-encrypt files option, 892–893
- features, 892
- remove option, 893–894

Evaluate function, 387–388

EvaluationError function, 388

- Exact function, 293
- ExecuteSQL function, 389, 477
 - database schema, 496
 - FileMaker_Fields, 497
 - FileMaker_Tables tables, 496–497
 - defining, 478
 - limitations, 480
 - SQL features, 498
 - SQL queries
 - concatenation, 490
 - GROUP BY clause, 493–494
 - JOIN clause, 492–493
 - limiting results of query, 494–495
 - literal text strings, 489, 490
 - SELECT statements, 481–489
 - UNION clause, 494
 - WHERE statement, 491–492
 - working, 478–480
- Exp function, 329
- Extend Found Set command, 77
- Extend Found Set function, 69
- Extend function, 377
- External storage
 - base directories, 237
 - creation, 238
 - delete, 239
 - edit button, 239
 - storage tab, 238
 - Thumbnails tab, 239
 - field
 - changes, 241
 - creation, 240
 - open storage, 241
 - options, 240
 - secure storage, 240

F

- Factorial function, 330
- FieldBounds function, 446
- FieldComment function, 447
- FieldIDs function, 447, 448
- FieldNames function, 448
- Field Picker palette
 - appearance
 - advanced graphic group, 571
 - behavior, 579
 - data formatting, 580
 - graphic group, 570
 - paragraph, 574
 - tabs, 576
 - text, 572
 - theme and style, 568
 - controls, 560
 - data field, 577
 - data formatting
 - container field, 583–584
 - date fields, 581–582
 - decimal format, 581
 - time field, 583
 - inspector, 561
 - arrange & align, 564
 - autosizing, 563
 - grid, 566
 - position, 562
 - sliding & visibility, 565
 - layout mode, 559–560
 - style panel, 568
- FieldRepetitions function, 448, 449
- Fields
 - auto-enter options
 - data entry, 187
 - dialog, 187
 - looked-up value, 188
 - prohibit modification, 188
 - content types, 176
 - date format
 - automatic conversion, 174
 - calculations, 173
 - conversion process, 174
 - short date format, 173
 - delete option, 186
 - display
 - calculation, 176
 - summary, 176
 - entry fields
 - auto-enter options, 187
 - calculation dialog, 196
 - calculations, 175
 - container, 175
 - data type, 172
 - date format, 173
 - furigana option, 196
 - numeric value, 172
 - storage options, 195
 - summarization, 197
 - tabs, 187
 - text, 172
 - timestamp fields, 175
 - time string, 174
 - validation options, 190
- FileMaker (*see* FileMaker object model)
- manage database dialog, 177
 - adding database, 183
 - copy and paste, 184
 - creation, 183
 - duplication, 183
 - selection, 182
- name modification, 185

Fields (*cont.*)

- naming fields, 178
 - visibilities, 178
- record type, 176
- repetitions, 171
- standard fields
 - ideas, 179
 - super group prefix, 180
 - universal status, 180
- storage options
 - container, 195
 - dialog window, 193
 - global storage, 193
 - indexing, 195
 - repeating, 194
- summarization
 - option dialog, 196
 - overview, 197–198
 - types, 197
- type modification, 185
- validation tab
 - control option, 190
 - entry fails, 192
 - FileMaker option, 192
 - option dialog, 190
 - override option, 191
 - requirements, 191
 - warning dialog, 191
- FieldStyle function, 449
- FieldType function, 449, 450
- File cache settings
 - permitted hosts, 27
 - plug-ins, 26–27
- File damage
 - cause, 144
 - understanding, 144
- File extensions, 153
- FileMaker, 99, 289
 - application resources, 13
 - company contact count, 289
 - components, 12
 - contact address label, 290
 - downloading sample data, 100
 - early, 4
 - Export Records process, 110
 - Export Field Contents, 115
 - finishing export, 115
 - Specify Field Order for Export, 111
 - history, 3–5
 - Import Records process, 101
 - automatic recurring import, 102
 - import field-mapping dialog, 104
 - performing import, 104

- Update existing records in found set option, 109
- Update matching records in found set option, 110
- library, 255
- popularity, 3
- product family, 14–15
- relational model, 231
- supported file types, 100
- user experience, 13
- using, 1
- versions, 4–5
- FileMaker Go, 16–17
- FileMaker Inc., 3–5, 896
- FileMaker object model, 199
 - company table, 200–201
 - custom fields, 200
 - project type, 201
 - standard fields, 199
- FileMaker Pro, 11
 - default menu bar, 28
 - feature-rich database, 11
 - features, 15
 - installation process
 - launch application, 21
 - launch installer, 19–20
 - run, 20
 - product family, 14–15
 - run, 21
 - user interface, 1
 - version, 4
- FileMaker Pro 16, 895
- FileMaker Pro Advanced
 - feature-rich database, 11
 - features, 15–16
- FileMaker server
 - access databases, 841
 - snapshot links, 841
 - URL, 842
 - admin console, 829, 831
 - AppleScript (*see* AppleScript)
 - basics, 827
 - client computer (upload), 832
 - cloud server, 840
 - database files, 835
 - databases folder (move), 831
 - deployment, 829
 - admin console, 830
 - features and benefits, 825
 - fmsadmin
 - command, 838
 - Help option, 839
 - location, 837
 - tools, 838
 - hosting databases, 16, 840

- legacy license, 826
- logging, 16
- media streaming, 16
- non-licensed connections, 827
- optimization, 827
- remove option, 836
- resources, 828
- restart, 836
 - safe methods, 837
 - unsafe methods, 836
- selection and sign, 832
- scheduled events, 16
- software installation, 828
- upload, 833
- uploading files, 831
- URL, 842
 - access credentials, 843
 - file extension, 842
 - FMP URL, 842
 - optional components, 843
 - parameter, 844
 - percent encoding, 843
 - script name, 844
 - server address, 843
 - variables, 845
- user connection license, 826
- web direct, 16
- FileMaker WebDirect, 17
- File menu
 - manage submenu, 31-32
 - provides access, 30-31
 - sharing submenu, 33
- File paths
 - building dynamic, 152
 - exclude file extensions, 153
 - functions to generate
 - contextual, 152
 - use variables, 152
 - wildcard, 153
 - formatting, 148-149
 - full, 151
 - network, 151
 - network address, 151
 - relative, 149-150
- FilerValues function, 294
- Filter function, 294
- Filtering portal records
 - formula, 648
 - multiple match fields, 648
 - search field, 650
 - steps, 647
 - user entry, 646
- Financial functions, 379
 - FV function, 379
 - NPV function, 380
 - PMT function, 380
 - PV function, 380, 381
- Find and Replace
 - command, 90
- Find Matching functions, 69
- Find Matching Records
 - function, 69
- Find mode
 - database window, 48
 - record content area, 42
 - view menu, 35
- Floor function, 330
- FMPPerception, 884
- fmsadmin, 837
- Font smoothing style, 24
- Format menu, 36
- Formatting bar, 54-55
- Formatting bar button, 53
- Formatting paths
 - prefixes, 149
 - types, 149
- Form layouts, 522
- Formulas
 - calculation dialog window, 258
 - calculation field
 - manipulation, 286
 - non-repeating fields, 286
 - repetition functions, 287
 - calculation interface, 277
 - comments, 260
 - multi-line comment, 261
 - single-line comment, 261
 - components, 260
 - constant-only formula, 279
 - constants, 261
 - context, 260
 - database, 259
 - dynamic field reference, 262
 - field references, 262
 - functions
 - calling a function, 263
 - nesting function calls, 264
 - optional parameters, 264
 - parameters, 264
 - function selection panel, 285
 - insert components
 - auto-complete, 283
 - field selection panel, 284
 - handling empty fields, 285
 - intentional errors, 280
 - operators
 - comparison, 265
 - definition, 265
 - logical operators, 268
 - mathematical operators, 269

Formulas (*cont.*)

- precedence, 271
- reserved words, 272
- text operators, 270
- space
 - advantage, 287
 - complex statements, 288
 - exceed 000 characters, 30, 289
 - items, 287
- storage options, 281
- variables, 274
 - global database variable, 276
 - Let statement, 275
 - local function/script variable, 275
 - types of, 274
- working process, 259

Form view, 49

Found set, 142

- definition, 84
- Find and Replace command, 90
- Omit Multiple command, 85
- Omit Record command, 85
- Replace Field Contents command, 88
- Show All Records command, 85
- Show Omitted Only command, 85
- sorting records in, 85
- spell checker, 91

Fragmentation, 144

From Index function, 64

From Last Visited Record function, 64

Front-end interface, 9

Full path, 151

Function buttons, 51-52

Function information sites, 897

Furigana function, 295

FV function, 379

G

GetAsBoolean function, 389

GetAsCSS function, 295

GetAsDate function, 295

GetAsNumber function, 296

GetAsSVG function, 297

GetAsText function, 297

GetAsTime function, 298

GetAsTimestamp function, 298

GetAsURLEncoded function, 299

GetAVPlayerAttribute function, 389-390

GetContainerAttribute function

- attributeName option, 348
 - audio, 350
 - barcode, 350
 - examples retrieve, 350
 - general group, 349

- groups of, 348

- image group, 349

- photo, 349

- signature group, 350

- file metadata, 348

GetField function, 390-391

GetFieldName function, 391

Get functions, 401-404

- Get (TriggerGestureInfo) function, 437

- Get (AccountExtendedPrivileges) function, 405

- Get (AccountGroupName) function, 405

- Get (AccountName) function, 405

- Get (AccountPrivilegeSetName) function, 405

- Get (ActiveFieldContents) function, 406

- Get (ActiveFieldName) function, 406

- Get (ActiveFieldTableName) function, 406

- Get (ActiveLayoutObjectName) function, 406

- Get (ActiveModifierKeys) function, 407

- Get (ActivePortalRowNumber) function, 407

- Get (ActiveRepetitionNumber) function, 408

- Get (ActiveSelectionSize) function, 408

- Get (ActiveSelectionStart) function, 408

- Get (AllowAbortState) function, 409

- Get (AllowFormattingBarState) function, 409

- Get (ApplicationArchitecture) function, 410

- Get (ApplicationLanguage) function, 410

- Get (ApplicationVersion) function, 411

- Get (CalculationRepetitionNumber) function, 411

- Get (ConnectionAttributes) function, 411

- Get (ConnectionState) function, 411

- Get (CurrentDate) function, 412

- Get (CurrentExtendedPrivileges) function, 412

- Get (CurrentHostTimestamp) function, 412

- Get (CurrentPrivilegeSetName) function, 413

- Get (CurrentTime) function, 413

- Get (CurrentTimestamp) function, 413

- Get (CurrentTimeUTCMilliseconds) function, 414

- Get (CustomMenuSetName) function, 414

- Get (DesktopPath) function, 414

- Get (Device) function, 415

- Get (DocumentsPath) function, 415

- Get (DocumentsPathListing) function, 416

- Get (EncryptionState) function, 416

- Get (ErrorCaptureState) function, 417

- Get (FileMakerPath) function, 417

- Get (FileName) function, 417

- Get (FilePath) function, 418

- Get (FileSize) function, 417

- Get (FoundCount) function, 418

- Get (HighContrastColor) function, 418-419

- Get (HighContrastState) function, 419

Get (HostApplicationVersion) function, 419
 Get (HostIPAddress) function, 419–420
 Get (HostName) function, 420
 Get (InstalledFMPlugins) function, 420
 Get (LastError) function, 420–421
 Get (LastExternalErrorDetail) function, 421
 Get (LastMessageChoice) function, 421
 Get (LayoutAccess) function, 421–422
 Get (LayoutCount) function, 422
 Get (LayoutName) function, 422
 Get (LayoutNumber) function, 422
 Get (LayoutTableName) function, 423
 Get (LayoutViewState) function, 423
 Get (MenubarState) function, 423
 Get (ModifiedFields) function, 424
 Get (MultiUserState) function, 424
 Get (NetworkProtocol) function, 424
 Get (NetworkType) function, 424–425
 Get (PageNumber) function, 425
 Get (PersistentID) function, 425
 Get (PreferencesPath) function, 425–426
 Get (PrinterName) function, 426
 Get (QuickFindText) function, 426
 Get (RecordAccess) function, 426
 Get (RecordID) function, 427
 Get (RecordModificationCount) function, 427
 Get (RecordNumber) function, 427
 Get (RecordOpenCount) function, 427–428
 Get (RecordOpenState) function, 428
 Get (RegionMonitorEvents) function, 428
 Get (RequestCount) function, 429
 Get (RequestOmitState) function, 429
 Get (ScreenDepth) function, 429
 Get (ScreenHeight) function, 430
 Get (ScreenWidth) function, 430
 Get (ScriptAnimationState) function, 430
 Get (ScriptName) function, 430–431
 Get (ScriptParameter) function, 431
 Get (ScriptResult) function, 431
 Get (SortState) function, 432
 Get (StatusAreaState) function, 432
 Get (SystemDrive) function, 432–433
 Get (SystemIPAddress) function, 433
 Get (SystemLanguage) function, 433
 Get (SystemNICAddress) function, 434
 Get (SystemPlatform) function, 434
 Get (SystemVersion) function, 434
 Get (TemporaryPath) function, 435
 Get (TextRulerVisible) function, 435
 Get (TotalRecordCount) function, 435
 Get (TouchKeyboardState) function, 436
 Get (TriggerCurrentPanel) function, 436
 Get (TriggerExternalEvent) function, 437
 Get (TriggerGestureInfo) function, 437

Get (TriggerKeystroke) function, 437
 Get (TriggerModifierKeys) function, 437
 Get (TriggerTargetPanel) function, 438
 Get (UserCount) function, 438
 Get (UserName) function, 438
 Get (UseSystemFormatsState) function, 439
 Get (UUID) function, 439
 Get (WindowContentHeight) function, 439
 Get (WindowContentWidth) function, 440
 Get (WindowDesktopHeight) function, 440
 Get (WindowDesktopWidth) function, 440
 Get (WindowHeight) function, 440–441
 Get (WindowLeft) function, 441
 Get (WindowMode) function, 441
 Get (WindowName) function, 442
 Get (WindowOrientation) function, 442
 Get (WindowStyle) function, 442
 Get (WindowTop) function, 442
 Get (WindowVisible) function, 443
 Get (WindowWidth) function, 443
 Get (WindowZoomLevel) function, 443
 GetHeight function, 351
 GetLayoutObjectAttribute function, 392, 393
 GetNextSerialValue function, 450
 GetNthRecord function, 394
 GetRepetition function, 378
 GetSummary function, 376–377
 GetThumbnail function, 352
 GetValue function, 299
 GetWidth function, 352
 Graph, 230

H

Help menu, 40
 HexDecode function, 353
 HexEncode function, 353
 Hiragana function, 300
 Host databases, 16
 Hour function, 340
 HTML design report, 881

I

Icon settings, 132
 If function, 394
 Inefficient design, 143
 Inline editing
 direct editing, 743
 formula text, 743
 multi-line formula, 744
 options dialog, 745
 pop-up menu, 745
 toggle button, 744

Insert Audio/Video function, 64
 Insert File function, 64
 Insert menu, 36
 Insert PDF function, 64
 Insert Picture function, 64
 InspectorPro, 885
 Int function, 330
 Integrated file architecture, 8
 benefits, 11
 overview, 10
 IsEmpty function, 395
 IsValid function, 395
 IsValidExpression
 function, 395, 396

■ J

Java Database Connectivity (JDBC), 160
 JavaScript Object Notation
 (JSON), 313
 array index, 315
 data types, 313
 definition, 313
 JSONDeleteElement, 317
 JSONFormatElements, 317
 JSONGetElement, 318
 JSONListKeys, 318
 JSONListValues, 319
 JSONSetElement, 319
 key elements, 315
 object, 314
 path, 315
 readable multi-line format, 314
 text functions, 317
 JSONDeleteElement function, 317
 JSONFormatElements function, 317
 JSONGetElement function, 318
 JSONListKeys function, 318
 JSONListValues function, 319
 JSONSetElement function, 319

■ K

KanaHankaku function, 300
 KanaZenkaku function, 300
 KanjiNumeral function, 300
 Katakana function, 300
 keyOrIndexOrPath method, 315-316
 Kiosk mode, 891

■ L

Last function, 378
 Launch Center, 59
 LayoutIDs function, 450, 451

Layout mode, 501
 identifying, 502-503
 layout design, 501-502
 menu changes, 509
 Arrange menu, 517-518
 Edit menu, 509-510
 Format menu, 514-515
 Insert menu, 513-514
 Layout menu, 516-517
 View menu, 510-512
 status toolbar, 504
 customizing, 508-509
 database menu, 507
 default, 504
 Exit Layout button, 508
 field picker toggle button, 507
 formatting bar button, 508
 inspector toggle button, 508
 Layout menu, 507
 Layout Objects toggle button, 508
 layout settings button, 507
 layout tools, 505-506
 menu changes, 509
 navigation controls, 505
 New Layout / Report button, 505
 screen stencil menu, 507
 theme selector, 507
 LayoutNames function, 451
 LayoutObjectNames function, 451, 452
 Layout objects
 button (*see* Button controls)
 charts, 655
 configuration, 590
 drop-down and pop-up menus
 drop-down list, 611-612
 dual-field, 613
 pop-up menu, 612
 value list, 610
 dynamic placeholder symbols, 620
 entry restrictions
 auto-enter calculation, 614
 auto-enter tab, 616
 methods, 614
 multiple item selection, 615
 rules, 613
 script trigger, 615-616
 single value, 616
 value list, 614
 values and bypass, 614
 field objects
 after adding object, 592
 before adding object, 593
 blank field object, 593
 configuration options, 591
 control style, 596

- duplication, 593
 - labels, 596
 - reference, 595
 - terminology, 591
 - insert, 590
 - merge fields, 620
 - merge variables, 621
 - panelcontrols (*see* Panel controls)
 - placeholders (*see* Placeholders)
 - portals
 - contact record, 639
 - direct record creation, 642
 - direct record deletion, 644
 - filtering portal records, 646
 - illustration, 640
 - parent record, 644
 - records creation, 642
 - remote table creation, 645
 - rows, 642
 - setup dialog, 640
 - static text, 620
 - types, 589
 - web viewers, 650
- Layout Objects palette, 585
- contextual menu, 586–587
 - controls, 585
 - convenient options and shortcuts, 586
 - edit object name text, 587
- Layouts
- label layout
 - contents, 548
 - dialog creation, 547
 - envelope layouts, 549
 - delete, 556
 - duplicate, 549
 - envelope view, 533
 - flow chart, 523
 - form layout, 532
 - label layout
 - contents, 548
 - envelope layouts, 549
 - label view, 533
 - list view, 532
 - manage layouts dialog, 556
 - methods, 530
 - new layout/report
 - dialog, 531
 - optimization, 558
 - part management
 - part definition dialog, 528
 - delete, 529
 - edit, 528
 - part setup dialog, 529
 - toolbar button, 527
 - printing tab
 - fixed page margins, 555
 - multiple columns, 555
 - setup dialog, 554
 - regions and controls, 524
 - boundary and off-screen developer area, 526
 - components, 525
 - horizontally resize, 526
 - part label definition, 525
 - vertically resize, 526
 - visible part, 525
 - report view, 533
 - representation, 519
 - script triggers, 555
 - setup dialog
 - general tab, 550
 - printing tab, 554
 - tabs, 550
 - views, 552
 - standard (non-summary) parts, 526
 - summary parts, 527
 - table
 - add fields dialog, 535
 - custom text dialog, 545
 - field definitions, 537
 - grand totals, 543
 - header and footer information, 544
 - invoice report demonstration, 537
 - modification, 533–534
 - report layouts, 535–537, 546
 - script, 545
 - specify fields dialog, 539
 - sort dialog, 541
 - subtotals and grand totals, 538
 - subtotals dialog, 542
 - view, 532
 - type categories, 519
 - creation options, 521
 - primary functional intent, 521
 - types, 520
 - window content views, 520
- Left function, 300
- LeftValues function, 301
- LeftWords function, 301
- Length function, 302
- Let function, 396–397
- Lg function, 331
- List function, 362
 - with local fields, 362
 - with related fields, 363
- List layouts, 522
- List view, 49
- Ln function, 331
- Location function, 457–458

LocationValues function, 458, 459
 Log function, 331
 Logical functions, 385
 Case function, 386
 Choose function, 386, 387
 Evaluate function, 387–388
 EvaluationError function, 388
 ExecuteSQL function, 389
 GetAsBoolean function, 389
 GetAVPlayerAttribute function, 389–390
 GetField function, 390–391
 GetFieldName function, 391
 GetLayoutObjectAttribute function, 392, 393
 GetNthRecord function, 394
 If function, 394
 IsEmpty function, 395
 IsValid function, 395
 IsValidExpression function, 395, 397
 Let function, 396–397
 Lookup function, 397–398
 LookupNext function, 398
 Self-function, 399
 Logical operators, 268
 AND, 268
 NOT, 268
 OR, 268
 XOR, 268
 Lookup function, 397–398
 LookupNext function, 398
 Lower function, 302

M

Macintosh computer, 4
 macOS, 889
 user name, 25
 Mail application, 5
 Manage submenu, 31–32
 Mathematical operators
 addition, 269
 division, 269
 multiplication, 269
 power of symbol, 270
 precedence, 270
 subtraction, 269
 Max function, 363
 with local fields, 364
 with related fields, 364–365
 Media streaming, 16
 Menu customization, 693
 actions menu
 add, 711
 creation, 710
 reports submenu, 712
 submenu item, 711

adding menu, 709
 capabilities, 693
 commands and menus, 713
 components, 694
 exceptions, 696
 hierarchical structure, 695
 item menu
 commands, 706
 edit dialog, 704
 menu types, 705
 modification, 707
 submenus, 707
 key command, 695
 manage dialog, 697
 menu bar, 694
 menu set, 712
 override standard commands, 708
 remove item, 709
 rename menu items, 708
 separator, 695
 set creation
 standard FileMaker menu, 699
 steps, 698
 set options, 695
 standard FileMaker menus, 699
 current menu, 703
 custom menu dialog creation, 701
 custom menu set, 703
 select menu dialog, 700, 702
 status, 716
 submenu, 695
 terminology, 694
 types, 695
 Middle function, 302
 MiddleValues function, 303
 Middleware, 10
 MiddleWords function, 303
 Min function, 365
 with local fields, 365, 366
 with related fields, 366, 367
 Minute function, 340
 Mobile functions, 457
 Location function, 457–458
 LocationValues function, 458, 459
 RangeBeacons function, 459, 460
 Mod function, 332
 Modify Last Find command, 77
 Multitier database architecture, 9

N

Naming tables, 157
 Nashoba Nutshell, 3
 Nashoba Systems, 3
 Navigation layout, 523

Network path, 151
 Network sharing, 819
 data loss, 821
 FileMaker (*see* FileMaker server)
 host application, 822
 limitations, 823
 peer and server, 822
 peer-to-peer database sharing, 823
 NPV function, 380
 Number functions
 Abs, 328
 built-in function, 327
 combination, 328
 Div, 329
 Exp, 329
 Factorial, 330
 Floor, 330
 Int, 330
 Lg, 331
 Ln, 331
 Log, 331
 Mod, 332
 Random, 332
 Round, 333
 SetPrecision, 333
 Sign, 333
 Sqrt, 334
 Truncate, 334
 Numeric constants, 261
 NumToJText function, 303
 Nutshell, 3

O

Objects manipulation, 661
 arrange and alignment
 align, 668
 distribute/space, 669
 group, 670
 layout mode, 667
 lock, 671
 rotate, 670
 resize to, 668
 stacking order, 671
 conditionalformat (*see* Conditional formation)
 hide objects, 672
 move
 dynamic guides, 665
 grid layout, 664
 guides, 664
 methods, 662
 ruler, 663
 naming objects, 679
 resize, 662

 selection, 661
 tab order, 677
 edit dialog, 677
 functions, 678
 Omit Multiple command, 85
 Omit Record command, 85
 Open Database Connectivity (ODBC)
 API, 160
 FileMaker database
 data entry limitations, 166
 data types, 166
 delete, 166
 external data source, 163–165
 indexing, 166
 information, 167
 modification, 166
 record locking, 166
 relationship graph, 165
 schema lock, 166
 supplemental fields, 166
 updates, 166
 host computer
 driver configuration, 162
 FileMaker, 160, 161
 Manager application, 161
 JDBC client application, 160
 shadow table, 166
 three-step process, 161
 Open settings, access to minimum version, 131
 Overwriting data, 141

P

Panel controls
 object types, 635
 slide control
 layout object, 638
 setup dialog, 638–639
 tab control
 hierarchical tab structure, 635
 new tab control, 636
 setup dialog, 636
 PatternCount function, 303
 Peer-to-peer database sharing, 823
 Permitted hosts, 27
 Pi function, 383
 Placeholders
 display calculation fields, 618
 formatting text, 618
 inspector, 619
 prompt messages, 616–617
 repetitions, 618
 text, 617
 vertical and horizontal arrangement, 618
 window mode, 617

- Plug-in, 896
 - concepts, 815
 - functions, 817
 - installation, 817
 - purchases
 - finding, 816
 - locations, 816
 - productive computing, 815
 - trio automatisering, 816
 - 360 Works, 815
- PMT function, 380
- Pop-up menu, 40
- Position function, 304
- Presentation tier, 9
- Preview button, 52
- Preview mode, 48, 94
- Printing
 - page setup, 93
 - Preview mode, 94
 - Print dialog, 96
- Privilege sets
 - built-in FileMaker feature, 872
 - categories, 863
 - custom extended
 - privileges, 872
 - data access and design
 - definition, 863
 - field dialog, 865
 - layouts, 865
 - records access, 863
 - scripts, 868
 - custom record dialog, 863
 - value lists, 867
 - default option, 862
 - edit dialog, 862, 875
 - extended section, 870–871
 - feature groups, 873
 - InvoiceApproval, 873
 - limiting access, 873
 - miscellaneous capabilities, 870
- Process Logic Tier, 10
- Proper function, 305
- PV function, 380, 381

Q

- Quick find record search, 67
- Quote function, 305

R

- Radians function, 383
- Random function, 332
- RangeBeacons function, 459, 460
- Records, 61

- closing, by committing changes, 64
- closing, by reverting changes, 64
- Export Records process, 110
 - Export Field Contents, 115
 - finishing export, 115
 - Specify Field Order for Export, 111
- field content formatting, 64
- field content modification, 63
- field focus, 62
- field navigation, tab order, 62
- foundset (*see* Found set)
- Import Records process, 101
 - automatic recurring import, 102
 - import field-mapping dialog, 104
 - performing import, 104
 - Update existing records in found set option, 109
 - Update matching records in found set option, 110
- menu, 37
- opening, 62
- Records management
 - Find mode, 70
 - complex finds, 78
 - Constrain Found Set command, 78
 - criteria, 73
 - Extend Found Set command, 77
 - Find requests, 75
 - identification, 70
 - Modify Last Find command, 77
 - saved finds, 79
 - search process, 77
 - status toolbar, 70
 - new record creation, 65
 - record duplication, 66
 - record search, 67
 - Find Matching functions, 69
 - quick find, 67
 - record set deletion, 66
 - single record deletion, 65
- Recursive custom functions, 472
 - limitations, 473
 - local variables across iterations, 474
 - simple, 472–473
 - test code inside a function, 474–475
- Relational databases, 7
- RelationInfo function, 452, 453
- Relationships, 203
 - manage database dialog, 215
 - button icon, 225
 - connection, 224
 - connection types, 205
 - many-to-many, 207
 - one-to-many, 206
 - one-to-one, 205

- data sources, 204, 212
 - edit dialog, 214
 - manage external window, 212
 - pop-up menu, 213
- definition, 203
- FileMaker relational model, 231
- foreign key, 204
- graph, 204, 230
- hops, 204
- hub-and-spoke monolith, 209–210
- integration, 203
- layouts, 205
- local key/primary key, 204
- manipulation
 - box moves, 227
 - comparative operators, 229
 - delete, 230
 - dynamic creation, 229
 - edit relationship dialog, 227
 - end-of-line symbols, 226
 - options, 229
 - small selector box, 226
 - sort, 230
- many-to-many
 - join table, 207
 - methods, 207
 - multi-key/complex key, 208
- match field, 203
- organizational methods, 212
- script steps, 205
- selector-connector concept, 211
- table (*see* Table occurrences)
- table occurrence, 204
- table occurrence group method, 210–211
- uses of, 204
- Relative path, 149–150
- Repeating functions, 377
 - Extend function, 377–378
 - Last function, 378
- Replace function, 306
- Replace Field Contents command, 88
- Request for Comment (RFC), 344
- RGB function, 322
- Right function, 306
- RightsWords function, 307
- RightValues function, 306
- RomanHankaku function, 307
- RomanZenkaku function, 307
- Runtime solution application(s), 887

S

- Sandbox
 - calculation field, 123
 - change default table name, 120–121

- data entry fields, 121–122
- functions, 120
- layouts
 - add fields, 126
 - create list, 127–130
 - renaming, 125–126
 - summary field, 123–124
- ScriptIDs function, 453
- Scripting, 749
 - benefits of, 721
 - calling, 723
 - context changes, 727
 - debugger (*see* Debugging scripts)
 - errors, 790
 - field interactions
 - Go to Field, 757
 - set field by name, 758
 - set field option, 757
 - set next serial value, 759
 - set selection, 759
 - steps, 756
 - flow, 722
 - found sets, 771
 - get directory, 761
 - If statements, 767
 - compound, 769
 - control steps, 767
 - nested, 769
 - simple, 768
 - illustration, 724
 - interaction, 724
 - loop statement, 772
 - confirmation process, 776
 - dialog field value, 775
 - found set, 772
 - lists of data, 774
 - record information, 772
 - summarized list, 773
 - macro, 721
 - methods, 722
 - navigation
 - go to layout, 751
 - Go to Record/Request/Page, 753
 - Go to Related Record, 754
 - parameters
 - definition, 724
 - parsing, 725
 - result value, 726
 - portal functions, 777
 - direct creation, 778
 - row's record, 777
 - records, 762
 - export records, 763
 - import records, 762
 - steps, 762

Scripting (*cont.*)

- searches, 770
- server, 726–727
- show custom dialog
 - button click, 767
 - general tab, 764
 - field input, 766
 - steps, 764
- triggers (*see* Triggers)
- user abort, 749
- variables, 750
- windows
 - card window, 788–789
 - conversion, 789
 - creation, 780
 - delete confirmation dialog, 781
 - delete record, 783
 - layout configuration, 785
 - layout dialog, 782
 - manipulation and management, 779
 - table creation, 782
 - testing, 788
- workspace
 - edit menu, 734
 - method, 728
 - script-editing pane, 731
 - script pane, 729
 - scripts menu, 736
 - step pane, 732
 - toolbar, 729
 - view menu, 735
- writingscript (*see* Writing script)
- ScriptNames function, 453
- Scripts menu, 38
- Script triggers settings, 135
- Search operator, 73
- Search text field, 52
- Seconds function, 341
- Security of database access, 851
 - benefits, 851
 - definition, 851
 - dialog warning, 852
 - file access tab, 877
 - FileMakeraccounts (*see* Accounts)
 - interface modes
 - manage security dialog, 853
 - detailed setup version, 854
 - privilege (*see* Privilege sets)
- Self-function, 399
- SerialIncrement function, 307
- SetPrecision function, 333
- Shadow table, 166
- Sharing. *See* Network sharing
- Sharing submenu, 33
- Show All Records command, 85

- Show Omitted Only command, 85
- Sign function, 333
- Sin function, 383
- SortValues function, 308
- Spell checker, 91
- Spelling settings, 133
- SQL queries, 481
 - concatenation, 490
 - GROUP BY clause, 493
 - HAVING clause, 494
 - limiting results of query, 494
 - FETCH FIRST clause, 495
 - OFFSET and FETCH FIRST clauses, combining, 495
 - OFFSET clause, 495
 - literal text strings, 489–490
 - SELECT statements, 481–482
 - benefits of aliases, 489
 - formatting requirements, 482–483
 - keeping references dynamic, 486–487
 - multiple fields, 485–486
 - reformatting, 487–488
 - SELECT DISTINCT to return unique values, 487
 - selecting entire table, 483–484
 - single field, 484
 - UNION clause, 494
 - WHERE statement, 491–492
 - JOIN clause, 492–493
 - multiple expressions, 492
 - single expression, 491
- Sqrt function, 334
- Stacking order, 671
- Starter solution
 - advanced, 46
 - create database from, 46
 - simple, 45
- Status toolbar
 - add items, 54
 - button style, 53
 - customization panel, 53
 - edit layout button, 53
 - formatting bar, 54–55
 - formatting bar button, 53
 - function buttons, 51
 - layout menu, 52
 - navigation controls, 51
 - preview button, 52
 - rearrange items, 54
 - remove items, 54
 - restore default set, 54
 - search text field, 52
 - toggle, 50
 - view button, 52
 - window ruler, 55–56

- StDev function, 367
 - with local fields, 367, 368
 - with related fields, 368
- StDevP function, 369
 - with local fields, 369
 - with related fields, 370
- Stored Data Tier, 10
- Structured Query Language (SQL), 477
- Styles
 - button object, 684
 - default option, 681
 - definition, 681
 - hierarchy definition, 682
 - modification, 690
 - object, 689
 - object states, 683
 - part types, 683
 - properties, 682
 - types, 682
 - update step, 685
- Substitute function, 289, 309
- Sum function, 370
 - with local fields, 371
 - with related fields, 371, 372
- Summary functions, 376
 - GetSummary function, 376–377

■ T

- TableIDs function, 453, 454
- TableNames function, 454
- Table occurrences
 - arrange, 218
 - color palette tool, 221
 - edit selection
 - page setup tool, 224
 - button icon, 221
 - creation, 222–223
 - deletion, 223
 - duplication, 223
 - graph, 224
 - specify table dialog, 222
 - formation, 220
 - interaction, 217
 - manipulations, 217
 - resize tool icon, 219
 - selection, 216
 - steps, 216
 - viewing options, 220
- Tables
 - adding tables
 - methods, 160
 - ODBC datasource (*see* Open Database Connectivity (ODBC))
 - two-step process, 160
 - data file-database conversion, 169
 - data modeling, 155
 - definition, 155
 - delete, 169
 - duplicate button, 167
 - hypothetical data model, 156
 - import button, 168
 - manage database dialog, 156
 - modification, 169
 - names, 157
 - properties, 157
 - selection, 159
 - warning dialog, 170
- Table view, 49
- Tan function, 384
- Technical design, 157
- Technology sites, 897
- TextColor function, 322
- TextColorRemove function, 323
- TextDecode function, 354
- TextEncode function, 354
- TextFont function, 323
- TextFontRemove function, 324
- TextFormatRemove function, 324
- Text formatting functions
 - built-in function, 321
 - RGB, 322
 - TextColor, 322
 - TextColorRemove, 323
 - TextFont, 323
 - TextFontRemove, 324
 - TextFormatRemove, 324
 - TextSize, 325
 - TextSizeRemove, 325
 - TextStyleAdd, 326
 - TextStyleRemove, 326
- Text function
 - built-in, 291–292
 - Char, 292
 - Code, 293
 - Exact, 293
 - Filter, 294
 - FilterValues, 294
 - Furigana, 295
 - GetAsCSS, 295
 - GetAsDate, 295
 - GetAsNumber, 296
 - GetAsSVG, 297
 - GetAsText, 297
 - GetAsTime, 298
 - GetAsTimestamp, 298
 - GetAsURLEncoded, 299
 - GetValue, 299
 - Hiragana, 300
 - KanaHankaku, 300

Text function (*cont.*)

- KanaZenkaku, 300
- KanjiNumeral, 300
- Katakana, 300
- Left, 300
- LeftValues, 301
- LeftWords, 301
- Length, 302
- Lower, 302
- Middle, 302
- MiddleValues, 303
- MiddleWords, 303
- NumToJText, 303
- PatternCount, 303
- Position, 304
- Proper, 305
- Quote, 305
- Replace, 306
- Right, 306
- RightValues, 306
- RightWords, 307
- RomanHankaku, 307
- RomanZenkaku, 307
- SerialIncrement, 307
- SortValues, 308–309
- Substitute, 309
- Trim, 310
- TrimAll, 310
- UniqueValues, 311
- Upper, 311
- ValueCount, 311
- WordCount, 312
- Text handling functions, 134
- Text operators
 - backslash, 271
 - concatenate text, 270
 - paragraph carriage return symbol, 270
 - quotations mark, 270
 - reserved name, 271
- Text settings, 133–134
 - data entry handling, 134
 - text handling functions, 134
- TextSize function, 325
- TextSizeRemove function, 325
- TextStyleAdd function, 326
- TextStyleRemove function, 326
- Themes
 - change dialog window, 687
 - default option, 681
 - definition, 685
 - layout, 687
 - manage dialog, 688
 - rules, 686
 - update, 685–686

Time functions, 341

- built-in, 340
- Hour, 340
- Minute, 340
- Seconds, 341
- Time, 341
- Timestamp fields, 175
- Timestamp functions, 341
- Tools menu, 38
- Triggers
 - caution, 813
 - database file, 812
 - event, 814
 - event precedence, 812
 - file option, 802
 - OnFileAVPlayerChange, 804
 - OnFirstWindowOpen, 803
 - OnLastWindowClose, 803
 - OnWindowClose, 804
 - OnWindowOpen, 803
 - finding objects, 812
 - layout object
 - OnObjectAVPlayerChange, 811
 - OnObjectEnter, 810
 - OnObjectExit, 811
 - OnObjectKeystroke, 810
 - OnObjectModify, 810
 - OnObjectSave, 811
 - OnObjectValidate, 810
 - OnPanelSwitch, 811
 - selection, 809
 - layouts, 804
 - OnExternalCommandReceived, 808
 - OnGestureTap, 808
 - OnLayoutEnter, 807
 - OnLayoutExit, 807
 - OnLayoutKeystroke, 807
 - OnLayoutSizeChange, 807
 - OnModeEnter, 808
 - OnModeExit, 808
 - OnRecordCommit, 806
 - OnRecordLoad, 806
 - OnRecordRevert, 806
 - OnViewChange, 808
 - parameters, 813
- Trigonometric functions, 381
 - Acos function, 381
 - Asin function, 382
 - Atan function, 382
 - Cos function, 382
 - Degrees function, 382
 - Pi function, 383
 - Radians function, 383
 - Sin function, 383
 - Tan function, 384

Trim function, 310
 TrimAll function, 310
 Troubleshooting process, 146
 Truncate function, 334

■ **U**

UniqueValues function, 311
 Universal resource identifier (URI)
 scheme, 653
 Upper function, 311
 User experience, 13
 User interface, 14, 499
 current layout, 8
 current record, 8
 found set, 8
 layout, 7
 options, 24
 User name, 25

■ **V**

ValueCount function, 311
 ValueListIDs function, 454
 ValueListItems function, 455
 ValueListNames function, 455, 456
 Value lists
 another file, 246
 conditional lists
 active companies, 252
 calculation field, 253
 dedicated relationship, 251
 drawback, 253
 types of, 250
 configuration, 250
 content creation, 246
 context insensitive, 243
 control styles, 243
 custom values, 246
 field, 247
 interface
 dialog screen, 244
 editing value, 245
 records, 249
 Variance function, 372
 with local fields, 372
 with related fields, 373, 374
 VarianceP function, 374
 with local fields, 374
 with related fields, 375, 376
 VerifyContainer function, 355

View menu, 35–36
 Virtual model, 155

■ **W**

Web Direct, 16, 17
 Web viewers, 42
 container field image, 655
 controls, 651
 Hello World creation, 653
 interaction, 652
 page view, 650
 setup dialog, 650
 text fields, 654
 web page, 653
 Wildcards, 153
 Window
 areas, 47
 components, 43
 menu, 39
 minimizing, 58
 modes, 48–49
 multiple
 close, 57
 create, 56
 create new, 56
 hiding and showing, 57
 select from menu, 57
 ruler, 55–56
 user name, 25
 WindowNames function, 456
 Writing script
 auto-complete, 740
 comments, 747
 configurable options, 742
 copy/paste, 742
 creation, 738
 empty dialog screen, 738
 inline options, 743
 insert, 739
 multi-line comment, 748
 open by clicking, 746
 step basics, 739
 steps pane, 741
 steps without
 options, 742
 WordCount function, 312

■ **X, Y, Z**

XML design report, 883