FileMaker® 12

IN DEPTH

Jesse Feiler

FileMaker® 12

IN DEPTH

Jesse Feiler

que®

800 East 96th Street
Indianapolis, Indiana 46240 USA

FILEMAKER® 12 IN DEPTH

ISBN-13: 978-0-7897-4846-1
ISBN-10: 0-7897-4846-0

Library of Congress Cataloging-in-Publication data is on file.

Printed in the United States of America

First Printing: June 2012

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

FileMaker is a registered trademark of FileMaker, Inc.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this.

Bulk Sales

Que Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales

1-800-382-3419

corpsales@pearsontechgroup.com

For sales outside the United States, please contact

International Sales

international@pearsoned.com

Editor-in-Chief
Greg Wiegand

Executive Editor
Loretta Yates

Development Editor
Sondra Scott

Managing Editor
Sandra Schroeder

Senior Project Editor
Tonya Simpson

Copy Editor
Bart Reed

Indexer
Cheryl Lenser

Proofreader
Leslie Joseph

Technical Editor
Beverly Voth

Publishing Coordinator
Cindy Teeters

Book Designer
Anne Jones

Compositor
Bronkella Publishing

CONTENTS AT A GLANCE

CONTENTS

ABOUT THE AUTHOR

Jesse Feiler has worked with FileMaker since its beginnings. He has written a number of books about FileMaker as well as OS X, iOS, iWork, Core Data, Objective-C, and other new technologies. His books have been translated into Japanese, Chinese, Polish, German, Spanish, French, Arabic, Hungarian, and other languages. As software director of North Country Consulting, he has designed and implemented a variety of FileMaker solutions for small businesses and nonprofits in fields such as production, marketing, the arts, printing and publishing, food service, and construction. His meeting management software for iOS devices, MinutesMachine, is published by Champlain Arts Corp (champlainarts.com).

His website is www.northcountryconsulting.com. You can find updates and file downloads there.

ACKNOWLEDGMENTS

This book could not exist were it not for the hard work and support of many colleagues and friends. In addition, heartfelt thanks are due to North Country Consulting clients who have brought a wide range of issues to the table and who, with humor and imagination, have been great partners in our common development of interesting FileMaker solutions.

At FileMaker, Kevin Mallon and Delfina Daves have once again provided continuing support and help. Through the FileMaker Business Alliance and TechNet, many resources are available to FileMaker users and developers, and we thank FileMaker for so aggressively providing the information to help us all use this exciting product successfully.

At Que, Loretta Yates has been a pleasure to work with. Project editor Tonya Simpson, development editor Sondra Scott, and copy editor Bart Reed all worked quickly and accurately to help guide the book through the production process. Beverly Voth has provided insightful and helpful comments from the technical side. Her help is greatly appreciated. And, as always, Carole Jelen at Waterside Productions has helped shepherd this project through to completion.

No acknowledgment would be complete without mentioning all the work our friends at FileMaker, Inc., do to make everything in our careers possible. FileMaker is a fantastic suite of products, and we're terrifically excited by the continued promise the FileMaker platform shows.

WE WANT TO HEAR FROM YOU!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an editor-in-chief for Que Publishing, I welcome your comments. You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@quepublishing.com

Mail: Greg Wiegand
Editor-in-Chief
Que Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

READER SERVICES

Visit our website and register this book at quepublishing.com/register for convenient access to any updates, downloads, or errata that might be available for this book.

INTRODUCTION

WELCOME TO FILEMAKER 12

Best of Three Worlds

Welcome to the world of FileMaker. From the start, one of the most important things to understand is that FileMaker is far more than just a database application.

FileMaker is nearly unique in the world of software. It is a powerful database system that can manage and store a wide range of information, it's an application for end users (like Microsoft Excel or Intuit's Quicken), and it's also a robust rapid application software development platform.

When you hear people speak about FileMaker, keep in mind they might be viewing it from any one of these different perspectives. An IT professional likely sees FileMaker as a database engine that fits into a larger security and network infrastructure. An end user is probably thinking about a specific solution built with FileMaker Pro and how it helps make her work more efficient. A software developer might see FileMaker as one of many tools he employs in building a wide range of applications.

This book was written with an eye toward the FileMaker developer community. If you're mostly interested in learning how to use the essential features of the FileMaker application, though, this book might not be for you. Although some introductory chapters are included to be as comprehensive as possible, the focus is on an audience that is largely familiar with the essential operations of FileMaker already and is interested mostly in topics for the beginning to advanced developer.

A key component of the FileMaker world today is FileMaker Go for iPad and iPhone (separate apps). It enables users to access databases built with FileMaker Pro and hosted by FileMaker Pro (for up to 10 users) or by FileMaker Server (for larger configurations). With FileMaker 12, the prices of these apps changed: they are both free. This means that with your

purchase of FileMaker Pro, an idea for a mobile solutions (and, we hope, this book), you're ready to deploy a world-class mobile solution.

In today's world, FileMaker developers are creating solutions for a range of technologies. The databases you create with FileMaker Pro can be accessed through FileMaker Pro and FileMaker Pro Advanced. They can be deployed with FileMaker Server and FileMaker Server Advanced. In addition, they can be accessed from iPhone, iPad, and iPod touch using FileMaker Go. And, in case that isn't enough for you, you can use Instant Web Publishing and Custom Web Publishing to deploy your FileMaker solution on any browser.

For more on FileMaker Go, ***see*** Data-Driven iOS Apps for iPad and iPhone with FileMaker Pro, Bento by FileMaker, and FileMaker Go, *by Jesse Feiler.*

How This Book Is Organized

FileMaker 12 In Depth is divided into five parts, organized into something like a tree. Part I, "Getting Started with FileMaker," and Part II, "Developing Solutions with FileMaker," constitute the "trunk" of the tree; they cover fundamental material that we recommend everyone read.

Subsequent parts branch out from this base. Part III, "Developer Techniques," focuses on using FileMaker's features to develop complete, robust database applications. Part IV, "Data Integration and Publishing," covers getting data into and out of FileMaker. And Part V, "Deploying a FileMaker Solution," covers options for making a FileMaker solution accessible to others.

The following sections describe the five parts of *FileMaker Pro In Depth* and the topics they cover.

Part I: Getting Started with FileMaker 12

The chapters in Part I introduce you to FileMaker and its uses and features, and they get you started with the basics of defining databases.

- Chapter 1, "FileMaker Overview," situates FileMaker Pro within the wider world of database and productivity software. This chapter is appropriate both for those who are new to FileMaker Pro and for those who have used previous versions and want a quick tour of the major innovations.
- Chapter 2, "Using FileMaker Pro," is intended as an introduction to the software from the perspective of a database user rather than a database developer. You'll see the major components and functions of the FileMaker interface, such as the Status toolbar, layouts, FileMaker's modes, and the basics of record creation, editing, and deletion.
- Chapter 3, "Defining and Working with Fields and Tables," provides a thorough overview of all of FileMaker's field types and field options, including lookups, validation, storage types, and indexing. This chapter is intended to help lay the groundwork for talking about database development and to serve as a thorough reference on FileMaker field types and options.
- Chapter 4, "Working with Layouts," covers all of FileMaker's layout-building options in detail. We cover all aspects of layout building and offer guidelines for quicker and more efficient layout work.

Part II: Developing Solutions with FileMaker

Part II is intended to introduce you to the fundamental techniques of database application development using FileMaker Pro and FileMaker Pro Advanced. Chapters 5 through 7 cover the theory and practice of designing and building database systems with multiple data tables. Chapters 8 through 10 introduce you to foundational concepts in application and reporting logic.

- Chapter 5, "Relational Database Design," introduces you to relational database design concepts. We proceed by working on paper, without specific reference to FileMaker, and introduce you to the fundamental vocabulary and techniques of relational database design (keys and relationships).
- Chapter 6, "Working with Multiple Tables," begins the task of translating the generic database design concepts of Chapter 5 into specific FileMaker techniques. We show how to translate a paper diagram into an actual FileMaker table structure. We show how to model different relationship types in FileMaker using multiple data tables and how to create fields that function effectively as relational keys.
- Chapter 7, "Working with Relationships," builds on the concepts of Chapter 6. Rather than focusing on FileMaker's relationships from the standpoint of database design, we focus on their practical implementation in FileMaker programming. We look in detail at the relational capabilities of FileMaker and discuss nonequality join conditions, file references, and some strategies for organizing a multitable system.
- Chapter 8, "Getting Started with Calculations," introduces FileMaker's calculation engine. The chapter delves into the major types of FileMaker calculations. We cover a number of the most important functions and discuss general strategies and techniques for writing calculations.
- Chapter 9, "Getting Started with Scripting," introduces FileMaker's scripting engine. Like the preceding chapter, this one covers the fundamentals of an important skill for FileMaker developers. We cover some common scripting techniques and show how to use event-driven scripts to add interactivity to a user interface.
- Chapter 10, "Getting Started with Reporting and Charting," illustrates the fundamental techniques of FileMaker Pro reporting (such as list views and subsummary reports), some more advanced subsummary techniques, and some design techniques for improving the look and usability of your reporting layouts. This chapter also explores the charting features of FileMaker Pro.

Part III: Developer Techniques

The chapters in Part III delve deeper into individual topics in advanced FileMaker application development. We build on earlier chapters by exploring more complex uses of portals, calculations, and scripts. We also offer chapters that help you ready your FileMaker solutions for multiuser deployment, and we examine the still-important issue of conversion from previous versions.

- Chapter 11, "Developing for Multiuser Deployment," explores the issues and challenges of designing FileMaker systems that will be used by several people at once. We discuss how FileMaker handles concurrent access to data and discuss the concept of user sessions.
- Chapter 12, "Implementing Security," is a thorough overview of the FileMaker Pro security model. We cover the role-based accounts feature, extended privileges, and many of the complexities of server-based external authentication against Windows or OS X user directories, for example.
- Chapter 13, "Using the Web Viewer," explores one of the interesting recent features of FileMaker Pro. You can incorporate live web pages into your FileMaker layouts, and you can use data from the FileMaker database to construct the URLs that are displayed.
- Chapter 14, "Advanced Interface Techniques," provides detailed explanations of a number of more complex, applied techniques for working with layouts and data presentation in a FileMaker application. You will see how to use FileMaker themes to make your solutions look great and work elegantly on the various devices that may be used to access them. This chapter also focuses on techniques for implementing FileMaker Go interfaces for iOS mobile devices.
- Chapter 15, "Advanced Calculation Techniques," looks closely at some of the more advanced or specialized types of FileMaker calculations, as well as the functions for text formatting and list manipulation.
- Chapter 16, "Advanced Scripting Techniques," like the preceding chapter, is full of information specific to features of FileMaker scripting. Here, we cover programming with script parameters, the significant feature of script variables, programming in a multiwindow system, and the complexities of scripted navigation among multiple tables and recordsets.
- Chapter 17, "Working with FileMaker Triggers," examines one of the most important features of FileMaker. Triggers let you set up automatic behaviors that occur whenever certain events happen. They let you exercise more control over the user interface with less programming in many cases. They can also improve the user experience by automatically performing scripts based on user actions so that you can eliminate buttons that require additional user actions and that use up precious space on layouts.
- Chapter 18, "Advanced FileMaker Solution Architecture," is the last of the chapters in the Advanced series. It presents a variety of features and solutions that integrate and expand some of the techniques in the previous chapters. You will find information on window management, multiwindow interfaces, and selection portals, among other topics.
- Chapter 19, "Debugging and Troubleshooting," is a broad look at how to find, diagnose, and cure trouble in FileMaker systems—but also how to prevent it. We look at some software engineering principles that can help make systems more robust, and can reduce the incidence and severity of errors. The chapter also includes detailed discussions of how to troubleshoot difficulties in various areas—from multiuser record lock issues to performance difficulties over large networks.
- Chapter 20, "Converting Systems from Previous Versions of FileMaker Pro," explores the complex issues involved in moving to FileMaker 12 from previous. We then discuss the mechanics of conversion in detail, and discuss some of the more significant pitfalls to be aware of.

Part IV: Data Integration and Publishing

Part IV covers technologies and capabilities that allow FileMaker to share data, either by exchanging data with other applications or by exporting and publishing data, for example, via ODBC, JDBC, and the Web.

- Chapter 21, "Connecting to External SQL Data Sources," explores FileMaker's ODBC/JDBC interface as well as the exciting features that let you add SQL tables to your Relationships Graph. This means that you can now use SQL tables very much as if they were native FileMaker tables. You can use them in layouts along with FileMaker tables, you can use them in reports, and you can even expand them by adding your own variables to the FileMaker database that are merged with the external SQL data as you use it.
- Chapter 22, "Importing Data into FileMaker Pro," looks at almost all the means by which you can import data into FileMaker. It covers how to import data from flat files, how to batch imports of images and text, and how to import images from a digital camera. (XML importing is covered in Chapter 24.) It also shows you how to import data from Bento on OS X.
- Chapter 23, "Exporting Data from FileMaker," is in some respects the inverse of Chapter 22. It covers almost all the ways by which you can extract or publish data from FileMaker.
- Chapter 24, "Instant Web Publishing," looks at the features of the FileMaker Instant Web Publishing model. Anyone interested in making FileMaker data available over the Web should begin with this chapter.
- Chapter 25, "Custom Web Publishing with PHP and XML," shows you how to use FileMaker's newest web publishing tools to build a PHP-based site. In addition, you will see how to export data using XML.

Part V: Deploying a FileMaker Solution

Part V delves into the choices you have for how to deploy a FileMaker database, including deployment via FileMaker Server and via kiosk or runtime mode using FileMaker Developer.

- Chapter 26, "Deploying and Extending FileMaker," provides an overview of the ways you can deploy a FileMaker database to one or more users, reviews plug-ins, and explores the means of distributing standalone databases. Read this chapter for a quick orientation toward your different deployment choices.
- Chapter 27, "FileMaker Server and Server Advanced," explores in depth setting up and working with FileMaker Server and FileMaker Server Advanced. The chapter covers setting up, configuring, and tuning FileMaker Server, as well as managing server-side plug-ins and authentication. The new Server Admin Console is described in detail here.

Special Features

This book includes the following special features:

- **Troubleshooting**—Many chapters in the book have a section dedicated to troubleshooting specific problems related to the chapter's topic. Cross-references to the solutions to these problems are placed in the context of the relevant text in the chapter as "Troubleshooting Notes" to make them easy to locate.
- **FileMaker Extra**—Many chapters end with a section containing extra information that will help you make the most of FileMaker Pro. In some cases, we offer expanded, fully worked examples of tricky database design problems. In others, we offer shortcuts and maintenance techniques gleaned from our collective experience with developing production FileMaker systems (creating custom function libraries or getting the most out of the team development). And in still others, we delve all the way to the bottom of tricky but vital FileMaker features such as the process of importing records.
- **Notes**—Notes provide additional commentary or explanation that doesn't fit neatly into the surrounding text. You will find detailed explanations of how something works, alternative ways of performing a task, and other tidbits to get you on your way.
- **Tips**—This feature identifies some tips and tricks we've learned over the years.
- **Cautions**—Here, we let you know when there are potential pitfalls to avoid.
- **Cross-references**—Many topics are connected to other topics in various ways. Cross-references help you link related information together, no matter where that information appears in the book. When another section is related to the one you are reading, a cross-reference directs you to a specific page in the book where you can find the related information.
- **FileMaker scripts**—Numerous examples of scripting are provided in the book. Because you can create long lines of code, they are sometimes split in order to be printed on the page. The ➥ character indicates the continuation of the previous line of code.

Downloadable Files

Most of the examples in this book are based on the FileMaker Starter Solutions that are installed automatically for you when you install FileMaker. Thus, you already have most of the files. In some cases, additional files or additional code has been added to the Starter Solutions as described in this book. These files can be downloaded from the author's website at northcountryconsulting.com. You can also download them from the publisher's website at www.informit.com/title/9780789748461.

FileMaker References

Other files and further information are available at filemaker.com. You can join TechNet at www.filemaker.com/technet/. It provides downloads, tech briefs, and members-only webinars on FileMaker topics. TechNet itself is free; you can add a FileMaker Developer subscription for

$99/year. Although the features may change over time, at this writing that subscription includes a FileMaker Server Advanced development license, a download of the official FileMaker Training Series, and, as circumstances permit, early views of unreleased FileMaker software.

If you are converting from a previous version of FileMaker, you can find conversion references at www.filemaker.com/r/conver.

Who Should Use This Book

Like FileMaker itself, this book has several audiences. If you work with structured data a lot (Excel spreadsheets, for example) but are new to databases, this book will provide you with a solid foundation in the world of databases, in the basics of database theory, and in the practical skills you need to become a productive database user or developer. The book's more introductory chapters tell you what you need to know to get started building basic databases for your own use. Later chapters introduce you to the world of multiuser database design and to some of FileMaker's more advanced application design features.

If you've worked with other database systems—either server-side relational database engines based on SQL or desktop development environments such as Access—this book will help you see how FileMaker Pro fits into the universe of database software. Refer to the "How This Book Is Organized" section earlier in this Introduction to get a sense of which chapters will get you started quickly with FileMaker.

And in case you're an old hand with FileMaker, we've provided a good bit of in-depth discussion of advanced techniques.

This page intentionally left blank

1

FILEMAKER OVERVIEW

FileMaker and Its Marketplace

However you approach FileMaker, some core strengths of the platform are important for all types of users:

- **Flexibility—**Working with FileMaker is inherently open ended. Users can easily create ad hoc data queries, quickly manage data entry, add functionality to a live system, or deploy to the Web in minutes.
- **Ease of use—**The folks at FileMaker, Inc., have labored hard to make FileMaker as approachable as humanly possible. Day-to-day users can easily learn how to add fields to a database, create reports, add form layouts, and more. With FileMaker Pro, organizations can be less dependent on specialized software engineers.
- **Mobility—**With the companion apps, FileMaker Go for iPad and FileMaker Go for iPhone, you can access FileMaker databases wherever you are. FileMaker Pro is the tool you use to build and manage your databases, but with FileMaker Go you can access your databases directly from your iOS device or over a network. The layout tools in FileMaker Pro 12 provide an easy way to customize interfaces for multiple devices.

  For more information, see Jesse Feiler's companion book, Data-Driven iOS Apps Using FileMaker Pro, FileMaker Go, and Bento by FileMaker*, as well as the rest of this book.*

- **Interoperability—**FileMaker Pro supports many common, open standards for data exchange, including Structured Query Language (SQL), Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), and Extensible Markup Language (XML), and allows users

to connect their database solutions to the greater world of standards-based applications—both within their organizations and online on the Web using Hypertext Transfer Protocol (HTTP).

- **Modern data architecture—**FileMaker, despite being "just" a productivity application that lives on your computer along with Pages, Microsoft Word, and Solitaire, allows users to create fully relational data structures and to properly build architectures that correctly manage real-world data.
- **Modern developer tools—**With its recent versions, FileMaker has added modern programming and scripting features, including parameters and results for scripts, parameterized calculations, and script triggers that allow event-based scripting.

Ultimately, FileMaker exists between the world of desktop applications and high-end, enterprise-level server systems. It is a third option: a flexible, robust workgroup application that can quickly come together, evolve over time, and be dramatically cost effective.

Mobility

Today, mobility is not just about mobile devices—it is about the mobility of data and solutions across a variety of platforms. FileMaker has been evolving into a structure that supports structured solutions with the data and related features in one set of files and interfaces in another set of files. It is relatively easy to implement or change the interface for a single device or platform without affecting others.

The evolving structure mentioned in this paragraph is supported and easy to implement, but it is not required.

On Filemaker.com, you will find a number of case studies showing how people have used FileMaker to power solutions that are accessed with mobile devices such as the iPad and iPhone.

FileMaker's web publishing features, which have been present in this architecture for a long time, allow other mobile devices to access shared solutions on mobile and desktop devices by using the Web.

The features of iOS mobile devices are integrated into FileMaker Go, allowing mobile users to take photos, capture signatures, and automatically retrieve their location. All of these mobile features can be accessed through FileMaker Go, and the data they capture (photos, signatures, and location) can be automatically stored in FileMaker databases on the device, on a shared FileMaker network server, or over the Web.

Rapid Application Development

In the world of software development, flexibility and speed are critical. We live in the world of Internet time, and usually businesses embark on a development project only when they need something yesterday.

The practices and experiences of the past three decades have proven software development to be a risky, unpredictable business. New job functions have developed in software quality assurance and project management. Certification programs exist to sift the wheat from the chaff.

FileMaker Pro exists in many respects to help organizations take on less risk and navigate the waters of software development without having to embark on massive engineering efforts when they aren't warranted. Because this is a rapid application development platform, it is possible to build a system in FileMaker Pro in a fraction of the time it takes to build the same system in more classic, compiled software languages or by using enterprise-level systems.

Low Total Cost of Ownership

FileMaker Pro is focused around offering a low total cost of ownership for organizations. In September 2008, The Industry Standard published "The 25 Best Business Software Tools and Services." It included FileMaker Pro on the list, saying "This database application is more intuitive than Access, while offering high-end features like live SQL data source support and easy Web-publishing capabilities, so your whole team can access the database via a browser."

FileMaker Is a Seasoned Platform

FileMaker Pro is now 25 years old. It was first developed in the mid-1980s by Nashoba Systems, which was purchased by Claris Corp., an Apple subsidiary. Originally for the Mac only, by FileMaker Pro 3.0 (1995) it was ported to Windows as well.

Version 4.0 introduced web publishing to the platform, and version 6.0 offered significant support for XML-based data interchange.

In 2004, FileMaker Pro 7.0 was released. This major release featured a reengineered architecture from the ground up, a new model for working with relationships, modern security capabilities, and the capability to hold multiple data tables within a single file. Since then, five major releases have continued the evolution of the product.

The addition of FileMaker Go for iOS devices (iPad, iPhone, and iPod touch) has built on the common architecture. FileMaker 12 and FileMaker Go, released in 2012, have expanded the multiplatform and multidevice implementation of the basic architecture.

You're Not Alone

FileMaker, Inc., has sold more than 18 million units worldwide as of this writing. Users range from a single magician booking gigs in Denver, Colorado, to Fortune 500 companies such as Bristol-Meyers Squibb and Coca-Cola. Just like any tool, FileMaker is noteworthy only when it has been employed to build something—and its builders come in all shapes and sizes. The only true common element among these builders seems to be that they own computers and have information to store.

There are some trends: FileMaker Pro is widely used in the worlds of K–12 and higher education. The top 100 undergraduate/doctoral universities in the United States and many more around the world, along with the top 250 school districts in the United States, use FileMaker Pro. The nonprofit industry is also a key focal point for FileMaker, as is the creative-professionals industry.

The most recent trend, based on anecdotal reports, is that FileMaker may be the most effective way to deploy mobile solutions that scale from one-person, on-the-go businesses to enterprises that deploy many devices in the field.

Introduction to Database Software

At its heart, FileMaker Pro is database software; databases are useful for keeping track of contacts and their addresses and phone numbers, the students in a school, the sales and inventory in a store, and the results of experimental trials. Although this sort of information can be kept in spreadsheets and word processor documents, a database makes it much easier to take on the following tasks:

- **Organizing your data into reports—**Databases can organize information into reports sorted by city, last name, price, or any other criteria necessary.
- **Finding one or several items in your collections—**Visually scrolling through a document with flat data displayed soon becomes unwieldy—even if you use a search command to do the scrolling for you. Databases make it relatively simple to search for one record (or row) of data within potentially millions of others.
- **Creating related associations among data—**Rather than duplicating the name of a company for multiple people, for example, or perhaps having to reenter an address in a dozen places, users can utilize databases to create associations between data elements (using a form of addressing) and preserve the integrity of their information.
- **Sharing data with other systems—**Databases are often built to exchange information with other systems; many become one component in a multitiered technology solution for companies—even small businesses often exchange data between QuickBooks, for example, and FileMaker Pro.
- **Describing the data—**Databases contain metadata, which is data about the data values. Whereas a spreadsheet allows you to format data in various date formats, a database allows you to specify that a certain field actually is a date and that no other types of data are allowed in that field.

There are other advantages to using database software, not the least of which is the capability in FileMaker Pro to construct a user interface that can map to an organization's workflow. The members of an organization often outgrow the documents of desktop applications when they need to support multiple authors; track data in structured, interrelated ways; or manipulate data sets based on differing criteria. Often the first herald of the need for a database occurs when users are frustrated with not being able to find a given piece of information.

The rest of this book gets into the details of how to do everything just mentioned and much more as well. You get a more detailed look at what a database is and how it works, how to build databases, and so on. But before we dive into the mechanics of databases, it's important to understand how they—and FileMaker—fit into the overall software computing world.

Database Software

A huge variety of software is on the market today. FileMaker generally falls into the category of business productivity software; however, it really is a hybrid application that marries application

productivity to a server-based architecture and database. It is as accessible as programs such as Microsoft Excel, Intuit's QuickBooks, and Apple's Numbers. It also enables developers to create complex workgroup databases that deploy in the same manner as other IT server–based applications.

The idea of managing a collection of structured data is what database software is all about. Some database products on the market manage specialized collections, such as business contacts. Products such as Act and Goldmine are good examples of those. Quicken, QuickBooks, and Microsoft Money all manage collections of financial transactions. On the Web, Salesforce, Microsoft CRM, SAP, and NetSuite provide services in the same general category, but with a very different type of implementation, interface, and customization environment.

FileMaker and other nonspecialized database products such as Microsoft Access are used to create database systems just as word processing software is used to create specific documents and Microsoft Excel is used to create spreadsheets. In fact, Microsoft Excel is often used as a database because it has several strong list-management features. It works well for managing simple databases, but it doesn't work well in managing multiple lists that are related to each other.

Often, simple grids of columns and rows of information (such as spreadsheets) are called *flat file* or *list* databases. Simple databases like these are generally self-contained; they usually don't relate to each other, so keeping information up to date across many such databases can become unwieldy or impossible. In such cases, a relational database is called for. FileMaker is a fully relational database system and allows developers to associate a row (or record) in one area of the database (a customer list, for example) with records in another area of the database (a list of purchase orders, for example). To take another example, users of a relational database system can tie a single company entry to multiple contact people or even associate a single person with multiple company entries. Rather than this information needing to be entered in a dozen different places, relational databases, using a form of internal addressing, simply associate one item with another (customers with their orders, companies with their contacts). In this way, FileMaker graduates from a single-user productivity tool to a fully realized database development platform.

Off-the-Shelf Software

There are many relational database products on the market: Specialized products such as Act and Quicken are also relational database products, but the difference is that those products are finished systems, offering a specific set of functionality, whereas products such as FileMaker are tools used to create custom systems tailored to the individual needs of an organization or a person.

It is certainly possible to re-create the functionality of Act or Quicken by using FileMaker, and some organizations choose to do so when faced with the fact that such specialized products are relatively inflexible. If an organization has nonstandard ways of doing things, its members might find it difficult to work with specialized products. Although FileMaker Pro comes with several database templates that might be perfectly suitable for an organization to use right away, most users instead turn to FileMaker to create custom database systems that exactly match how their organizations operate.

Custom Development Software

With a database development tool such as FileMaker Pro, a person can build a system to be exactly what is needed. It's the difference between buying a house that is a pretty good match and building a custom home that has exactly the features one wants (or at least can afford).

Home construction is actually a great analogy for building a database because both follow similar trajectories. A home has to be designed by an architect before it can be built. An owner has to wait for the home to be built before he can move in, and questions or issues often arise during the construction process. After the home is built, the owner's needs might change and he might have an addition built onto the house to accommodate changed circumstances.

Building a custom home often follows a similar path: The foundation needs to be laid, and the walls and plumbing need to be stubbed in before the final coat of paint can be applied to the walls. Software development often is a complex layering of interdependent parts.

Finally, imagine that a home's construction is well under way, and the owner decides to move the living room wall six feet. Although that is always possible, the impact of that change will vary a great deal depending on the stage at which the crew is working.

This last point is an important one, and it is also where we diverge from the home construction analogy because real-world environments always change. This is especially true for today's email-driven, connected-network world. One of the key advantages to developing database systems in FileMaker Pro is that these systems can be rapidly redesigned, even while the system is in use by other users. Almost any aspect of a FileMaker system can be changed while it's live, if need be, although doing so might not always be advisable. FileMaker's greatest strength is its inherent flexibility.

note

People frequently wonder when to turn to custom development, when to use built-in FileMaker Starter Solutions, and when to start their own development from scratch. The author provides his own simple rule of thumb: The closer any operation is to the core of an organization, the more it is a candidate for custom development. If your organization prides itself on customer relations, a customized contact management system might be for you, whereas inventory control can chug along quite happily with an off-the-shelf product or a FileMaker Starter Solution (perhaps with a few tweaks).

On the other hand, if your organization really shines at managing its complex inventory process, that might be where your customized software should be focused, and your contact management software may be a Starter Solution, an off-the-shelf product, or even a web-based FileMaker solution accessed from your smartphone.

What Database Software Does

FileMaker is database software. The thing that makes it unique in the market is the ease and means by which it allows developers to present information. However, it's important to grasp the fundamentals of how all database software—including FileMaker—works. The simplest kind of database is a list. It could be a list of employees or products or soccer teams. Consider an employee list example. The information a Human Resources department might want to keep track of could look like the information shown in Table 1.1.

Table 1.1 Employee Table

First Name	Last Name	Department	Extension
Jane	Smith	Marketing	327
Calvin	Russell	Accounting	231
Renee	Frantz	Shipping	843

In database parlance, a list like this is called a *table*. Simply put, a table is a collection of like things—in this case, people. After a table for people is established, one might extend it to include other attributes (or columns) for, say, phone numbers. Table 1.2 shows the result.

For a thorough understanding of data modeling and the definition of tables, **see** *Chapter 5, "Relational Database Design."*

Table 1.2 The Growing Phone Directory

First	Last	Department	Ext.	Work	Cell	Fax
Jane	Smith	Marketing	327	555-1234	555-4453	555-3321
Calvin	Russell	Accounting	231	555-8760		
Renee	Frantz	Shipping	843	555-1122	555-1123	

As mentioned earlier, this type of database is called a *flat file* database because everything is in one table. Although having everything in one place is nice, this kind of structure has shortcomings. In this case, every time someone thinks up a new type of phone number to track, another column has to be added to the table. This is likely fine for phone numbers—in the real world people usually have only a handful—but imagine what would happen if the example were tracking people's previous job titles? The spreadsheet or list would have a potentially unlimited number of columns, and there would be no logical correspondence between one person's "Job #1" column and another's.

Furthermore, if someone doesn't have a particular type of phone number, that cell is left blank, resulting in a "Swiss cheese" look to the table, as shown in Table 1.2. Unused cells take up space in the database and can slow things down for larger data sets.

In a relational structure, only the first three columns would be in the employee table itself. The last four columns, which all represent phone numbers of some kind, would be moved to their own table. A label field could be added to identify each type of phone number, with the resulting two tables looking something like Tables 1.3 and 1.4.

Table 1.3 The Revised Employee Table

Emp ID	First	Last	Department
1	Jane	Smith	Marketing
2	Calvin	Russell	Accounting
3	Renee	Frantz	Shipping

Table 1.4 The New Phone Table

Emp ID	Label	Number
1	Extension	327
1	Work	555-1234
1	Cell	555-4453
1	Fax	555-3321
2	Extension	231
2	Work	555-8760
3	Extension	843
3	Cell	555-1122
3	Fax	555-1123

Note that a field has been added: Emp ID. Think of this field as an internal address within a table. It is used to match employees with their phone numbers. In relational database terminology, this column is called a *key field*. The FileMaker Pro help system refers to it as a *match field*, but they are one and the same. Key fields are used to identify specific records.

Both Figure 1.1 and Figure 1.2 show Starter Solutions that ship with FileMaker Pro. In Figure 1.1, you see the Contact Management solution; it uses a flat file approach with separate fields for the various phone numbers, just as you saw in Table 1.2.

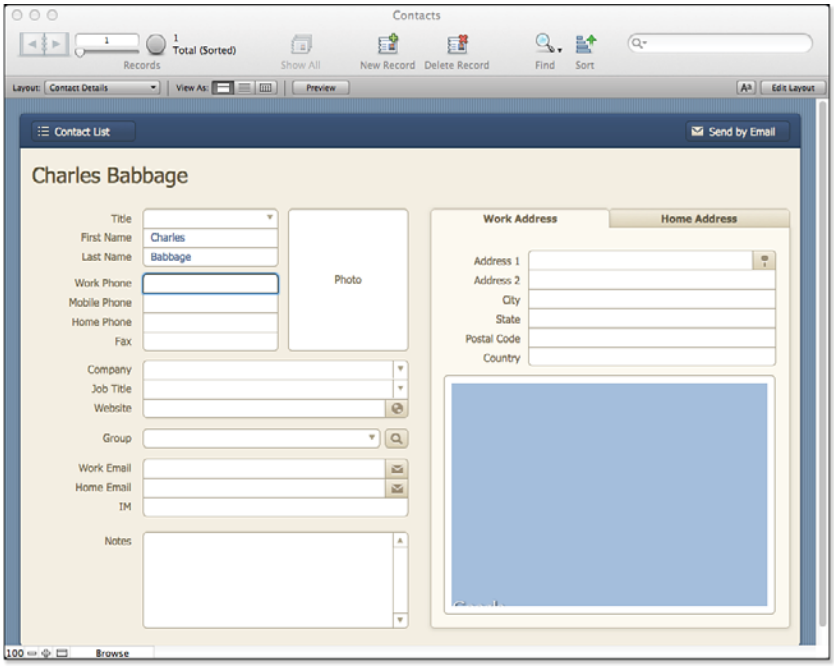


Figure 1.1
FileMaker can be used to construct simple flat-file databases.

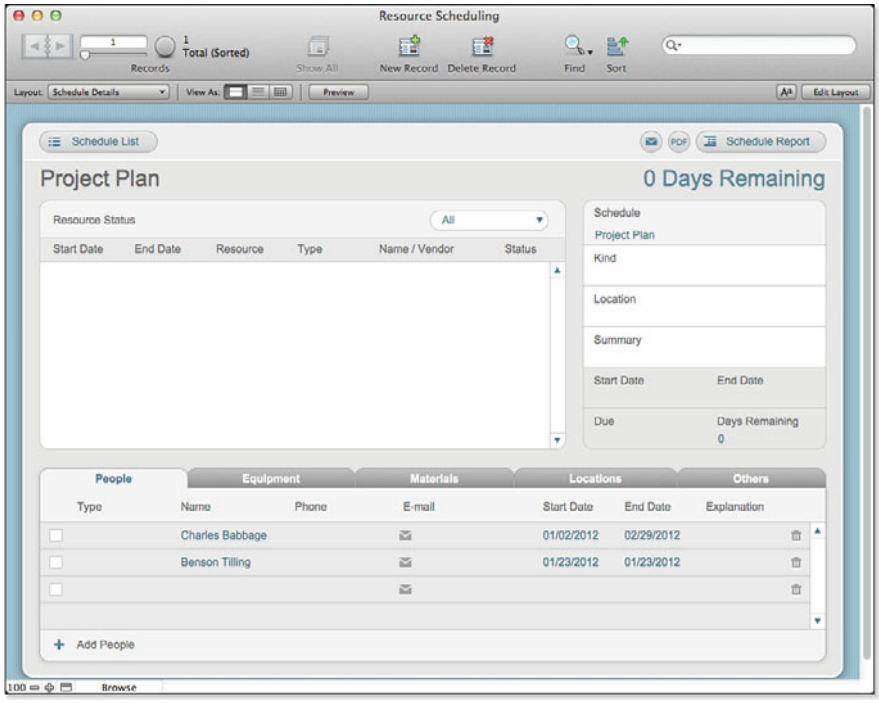


Figure 1.2
FileMaker Pro can easily add related data.

In Figure 1.2, the People Management solution lets you track multiple goals for a person. The data for a goal is more complex than a simple phone number; in addition, the number of possible goals for a person can be quite large. A flat-file approach would not work. In the People Management solution, a single table contains the employee information, and a separate, related, table contains the goal information. Still a third table contains issue information for that person. Logically, there is no difference between relating phone numbers and relating goals to a person.

For a thorough introduction to database application development with FileMaker Pro, **see** *Chapter 3, "Defining and Working with Fields and Tables," and Chapter 4, "Working with Layouts."*

Overview of the FileMaker Product Line

FileMaker Pro is just one product in a broader product line. It's worth noting the differences among the products and how they work together:

- **FileMaker Pro—**This is the regular desktop client version of FileMaker. It can be used to author new database systems, host systems for a limited number of guests (currently nine), or serve as a guest of a hosted system. It can also publish as many as ten database files to up to five users with Instant Web Publishing (IWP).

- **FileMaker Pro Advanced—**This version has all the capabilities of the regular version of FileMaker Pro; it also has additional functionality aimed at application developers. With FileMaker Pro Advanced, developers can create custom functions, add custom menu sets to a database, and create external function plug-ins (all these enhancements to the files can be used by both FileMaker Pro and FileMaker Advanced users). A Debug Scripts feature and Data Viewer allow developers to walk through scripts one step at a time and test calculations, watching the effect of each script step or process. The Database Design Report (DDR) enables developers to document and troubleshoot development issues from a systemwide perspective. The Advanced version also enables developers to create runtime versions of single-user solutions and enable kiosk mode.

  The author strongly recommends developing with FileMaker Pro Advanced. The additional functionality in custom functions alone makes it well worthwhile, not to mention the added capability to control all menu selections in a solution. The debugging tools are invaluable, and the DDR is a great source for documentation and troubleshooting alike.

- **FileMaker Server—**This software hosts FileMaker files on a hardware server and offers support routines, evaluates server-based calculations, and provides for a larger user load: FileMaker Server can host a maximum of 125 database files and 250 FileMaker Pro or FileMaker Pro Advanced client connections. In addition, it can manage database backup schedules, log usage statistics, disconnect idle users, and manage FileMaker plug-in updates. It supports Custom Web Publishing XML or PHP (PHP: Hypertext Preprocessor) to up to 250 web sessions for users accessing the database with browsers. With FileMaker Server, scheduled scripts can import and export data; FileMaker Server itself can also send email at the request of clients.

  For a complete discussion of Custom Web Publishing, including session handling and server capacity, **see** *Chapter 25, "Custom Web Publishing with PHP and XML."*

- **FileMaker Server Advanced—**FileMaker Server Advanced has all the features of FileMaker Server and can also host ODBC/JDBC and provide Instant Web Publishing for up to 100 web sessions. In addition, the limit of 250 clients is removed: The limit is based solely on the capabilities of the hardware on which it is deployed.

  For a complete discussion on ODBC, **see** *Chapter 21, "Connecting to External SQL Data Sources"; for Instant Web Publishing,* **see** *Chapter 24, "Instant Web Publishing."*

- **FileMaker Go—**These apps for the iPhone and iPod touch, as well as the iPad, enable you to access FileMaker databases that are stored on the device or are available over a network. The layout and design features of FileMaker Pro are not available on FileMaker Go, but you can use the layouts and databases that you design with FileMaker Pro on your mobile devices.

Consult the FileMaker website for more information on the products and special offers that might be available. Also, use the website to locate consultants who might have additional information on pricing and bundles.

FileMaker Deployment Options

After a database application has been developed in FileMaker Pro or FileMaker Pro Advanced, it can be deployed in various ways and on various operating systems. FileMaker Pro 12 runs on Mac OS X and Microsoft Windows. The following sections describe different ways to deploy a FileMaker database system.

For detailed technical specs, hardware and software requirements, and other issues, **see***the FileMaker website at* **http://www.filemaker.com.**

Single User

Many people get their start in FileMaker development by building a small application for their personal use. Although FileMaker Pro is inherently a networkable application, there's nothing wrong with a single user working with a system on his computer. These solutions often grow over time—sometimes by being networked to other computers, and other times by additional functionality being provided (oftentimes, both happen).

Peer-to-Peer Hosting

The next stage in the evolution of a typical system is that other members of an organization notice the system that a single person made and want to use it also. It's a simple matter to enable FileMaker Network Sharing on a file; after that's done, other FileMaker users can become guests of one user's shared file. This kind of FileMaker hosting is called *peer-to-peer* because the database host and the database clients all use the same application: desktop versions of FileMaker Pro or FileMaker Advanced.

Only one user can open a FileMaker database file at a time. When you are using peer-to-peer hosting (or FileMaker Server hosting, as described next), one user or FileMaker Server opens the database. The other users connect over the network to the copy of FileMaker that opened the database; networking transactions between that application and the "client" users provide access to the database through the first copy of FileMaker or through FileMaker Server.

note

FileMaker, Inc., also provides a personal database app called Bento. Like FileMaker Pro, Bento comes with a number of built-in templates that you can use with or without modifications for your own purposes. Bento can be synchronized with up to five of your mobile iOS devices and your Mac. It can be an excellent tool for managing your life or small business; it can also be a wonderful tool for prototyping what will eventually become a FileMaker solution.

tip

Considerations to keep in mind with single-user hosting include the following:

- Only ten files at a time can be hosted on a single machine this way.
- Up to nine users can be guests of a file hosted in this fashion. If you're the host of a file, you can't close the file while other users are working with it.
- Performance might suffer for other users as the hosting user puts her computer through its daily paces.

tip

FileMaker Go can be used to access peer-to-peer shared databases.

FileMaker Server Hosting

FileMaker Server is optimized for sharing FileMaker databases, and it can host (share) more files (125) for more users (250) than FileMaker Pro peer-to-peer can. Administrators can perform the following tasks:

- Remotely administer the server
- Set up groups of administrators with their own privileges and passwords
- Create schedules for automated database backups
- Set the server to encrypt the network traffic between the server and the clients
- Log server actions

FileMaker Server also provides Custom Web Publishing with XML and PHP.

FileMaker Go can access databases hosted on FileMaker Server.

For more information about hosting database files with FileMaker Server, **see** *Chapter 27, "FileMaker Server and Server Advanced."*

FileMaker Server Advanced Hosting

FileMaker Server Advanced can host files for FileMaker users just as FileMaker Server can, but it can also allow ODBC/JDBC clients to access hosted files and provide service as a web host, allowing up to an additional 100 user connections for Instant Web Publishing clients.

Kiosk Mode

Using FileMaker Pro Advanced, you can configure FileMaker databases to run without the menu bar or operating system controls, effectively making a solution take over the entire computer screen. Developers will need to build whatever user interface controls users might need, given that menus are no longer available.

FileMaker Single-User Runtime

FileMaker Advanced also allows developers to bind files into a runtime application that allows a single user to work with a FileMaker solution without needing a copy of FileMaker. No authoring capabilities exist (a user cannot access layout mode or make schema changes via the runtime engine), nor can the application serve as a host (peer-to-peer or server based). In addition, the PDF output and External SQL Data Sources are not provided in runtime versions. However, this is a great option for creating a commercial application without requiring that customers purchase copies of FileMaker Pro.

To learn more about kiosk mode or the runtime engine, **see** *Chapter 26, "Deploying and Extending FileMaker."*

Extending the Functionality of FileMaker Pro

FileMaker solutions can be enhanced by incorporating plug-ins that extend the functionality of FileMaker Pro. The functionality that plug-ins offer varies widely and is determined by the third-party developers who write and market plug-ins. Some plug-ins provide advanced math capabilities, some generate charts from FileMaker data, some manipulate image files, and others provide security or user-interface enhancements. Literally dozens (if not hundreds) of plug-ins are actively supported by the FileMaker industry at large.

To get more information about plug-ins, **see** *Chapter 26, "Deploying and Extending FileMaker."*

Technical Specifications

FileMaker 7 represented a complete re-architecting of FileMaker's file format and dramatically extended the platform's capabilities. The transition from the file format supported by FileMaker 3.0 through FileMaker 6.0 is still going on for some users; for others, it is more or less complete as they move to the latest versions of FileMaker Pro.

Table 1.5 summarizes the capabilities of FileMaker Pro 12.

Table 1.5 FileMaker Pro 12 Capabilities

Feature	FileMaker 12
Number of tables per file	1,000,000
Maximum file size	8TB
Maximum amount of data in a text field	2GB of data, or 1GB of Unicode characters
Number of significant digits in a number field	FileMaker Pro indexes the first 400 significant digits (numbers, decimal points, or signs) of the field, ignoring letters and other symbols
Number of characters in a number field	800
Maximum number of files allowed open on the client	Limited only by memory
Maximum records per file (theoretical limit)	64 quadrillion over the lifetime of the file
Maximum amount of data allowed in a container field	4GB
Maximum number of fields in a table	256 million over the lifetime of the file
Number of script steps supported by Instant Web Publishing	79
Number of FileMaker clients hosted by FileMaker Server	999
Number of web clients hosted by FileMaker Server Advanced	100

This page intentionally left blank

2

USING FILEMAKER PRO

Getting Started

It's time to roll up your sleeves and actually put FileMaker Pro to use. Most of this book deals with being a FileMaker developer—someone focused on the programming side of creating and managing FileMaker solutions. However, development makes up only a small percentage of the overall time a given database is used. Much of the time a FileMaker solution will simply be in use and its users will care nothing for scripting, calculations, or the vagaries of user interface design. They will simply be involved in working with a developer's creation and will not need to know anything of the programming side of FileMaker.

This chapter introduces you to how to make the most of FileMaker databases that have already been built. All FileMaker databases—often called *solutions*, *systems*, or *applications*—have certain common elements, and becoming adept at using FileMaker Pro solutions will not only help you manipulate and analyze data better, but will assist you in extending what you can accomplish with that data.

When it comes to getting started with FileMaker, you need to know a few basics. Installing FileMaker Pro is automated, as is the case with most software today. Whether you have a CD with FileMaker on it or have downloaded the software from filemaker.com, you'll find an installer on the disk or disk image. If there is a Read Me file, do just that before you continue.

Even after you have installed FileMaker, you might find a minor barrier before you can use it. Automated software updates might present you with a window after you launch FileMaker and before you can get to work. Software updates occur whenever updates are available.

The Quick Start screen is what you normally see when FileMaker starts. From there, you can open or create databases and get help. After you look at these aspects of FileMaker, it will be time to move on to actually working with databases and their components.

Registration

You can choose to register your copy of FileMaker; this also provides FileMaker with personal information, including your address, which can be used to notify you of new products, updates, and the like. During the registration process, you can indicate to FileMaker what sorts of communications—if any—you would like to receive about FileMaker products. FileMaker can also use the information from the registration process to find out more about the people who use FileMaker and the purposes to which they intend to put the product.

Registration is required for the use of free trial software. You might also be prompted to register your software during the installation process.

If you choose not to register at this time, you can always decide to register later by choosing Register Now from the Help menu. Registration is optional, meaning you never have to register.

tip

Most software is sold as a download, which means that fewer and fewer cartons, manuals, and CDs need to be produced and stored. This provides savings to users and vendors, and uses fewer raw materials and provides less trash when products are discarded in favor of new ones. In this environment, registration is increasingly important to prove your ownership of a product. If you have a credit card receipt, you might be able to track through the process of the purchase to prove that you did, indeed, purchase a product, but if you have registered the product, the process is immensely easier. Some people are hesitant to register because they are afraid of receiving too much unsolicited commercial email, but FileMaker, like all responsible companies, respects your wishes in this regard. Just make certain to check the communication options you prefer in the registration process.

Software Updates

You might be prompted to download updates to FileMaker software. This accounts for a screen that you might see when you first launch FileMaker Pro. The choice of downloading the update is up to you—as is the choice of whether to perform this automatic check, as shown in Figure 2.1. Choices in this dialog are part of your preferences, which you can get to in OS X from the Preferences command in the FileMaker application menu. In Windows, the Preferences command is at the bottom of the Edit menu. This is one of the few interface differences between the two operating systems in FileMaker Pro.

tip

Many people automatically check for software updates right after the installation of a new application. For any product, some minor revisions are often released shortly after the main release of the product, and it makes sense to start your adventures with a new version of the software with the latest code.

Using the Quick Start Screen

When you launch FileMaker Pro, you see the Quick Start screen—generally the first screen after registration and software updates (if any) are disposed of. The Quick Start screen provides a simple interface to a variety of FileMaker Pro tools, as shown in Figure 2.2.

Figure 2.1
You can control checking for software updates.

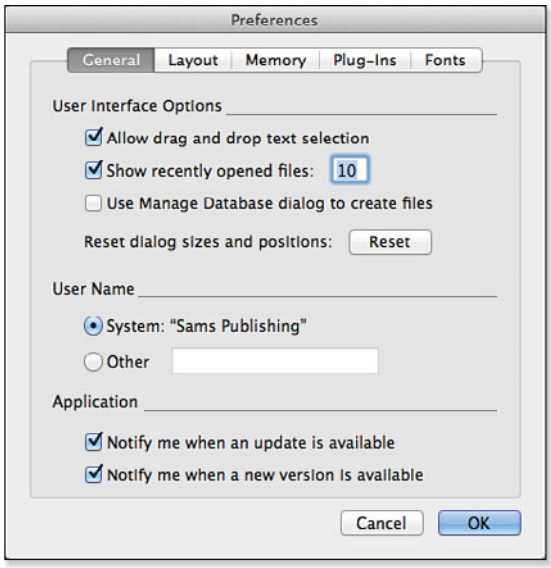
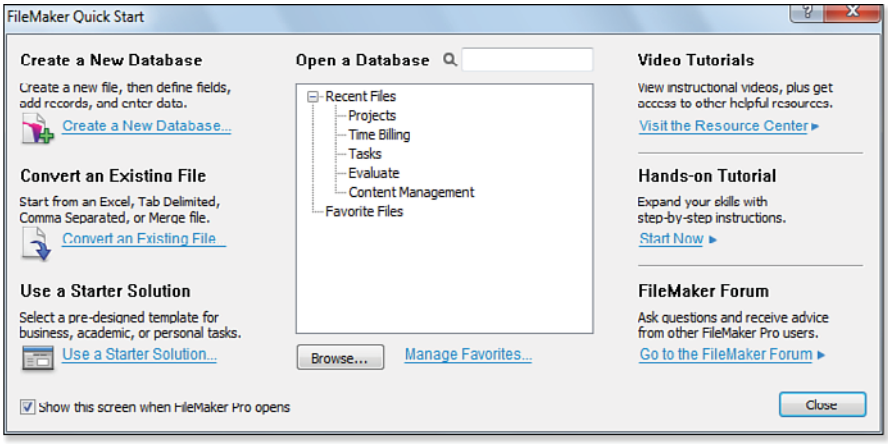


Figure 2.2
The Quick Start screen is your gateway to FileMaker Pro.



At the left, three icons let you choose from tools to create a database, open a database, or get assistance. The Quick Start screen opens to whichever view you last selected.

You can create a database from scratch or from one of the Starter Solutions; you can also choose to create a database directly from an existing document in a non-FileMaker format, as shown in Figure 2.2. The *Starter Solutions* are a set of FileMaker Pro databases that you can use as is or with modifications for your own customized solutions. FileMaker categorizes the Starter Solutions into a variety of areas (some are in more than one area). In addition to the Quick Start Screen, you can choose File, New From Starter Solution to create a database from a Starter Solution.

In the center of Quick Start, you can open files and servers that you have recently used. You can use a Browse button to open your standard Open File dialog.

*For more information about opening remote files, **see "Working in FileMaker Pro," p. 39**.*

Getting Help

For FileMaker users, help consists of a variety of tools ranging from online help to the FileMaker website and books such as this one. For most people, help begins with the Help menu, shown in Figure 2.3.

The Resource Center command takes you to the FileMaker website where additional information is provided.

In addition to the Help menu, you will find Learn More links on many of the FileMaker dialogs. They are discussed at the appropriate points of this book.

note

As you can see from the check box at the bottom left of the Quick Start screen, you can choose not to have this screen shown at startup. If you choose that option, you can always reopen this screen by choosing Quick Start Screen from the Help menu.

tip

Until FileMaker Pro 12, the Starter Solutions used a common user interface. Those Starter Solutions served as the basis of many solutions that have been developed over the years. With FileMaker Pro 12, a wider variety of interfaces has been created to show you the various possibilities from which you can choose.

tip

As you look at recent files, you can select one to add to your Favorite Files (either local or remote). Just select it and click the Manage Favorites link. This technique can save you a lot of time, particularly if a file is on a remote server and it would take several mouse clicks to select the server, the appropriate folder, and then the file.

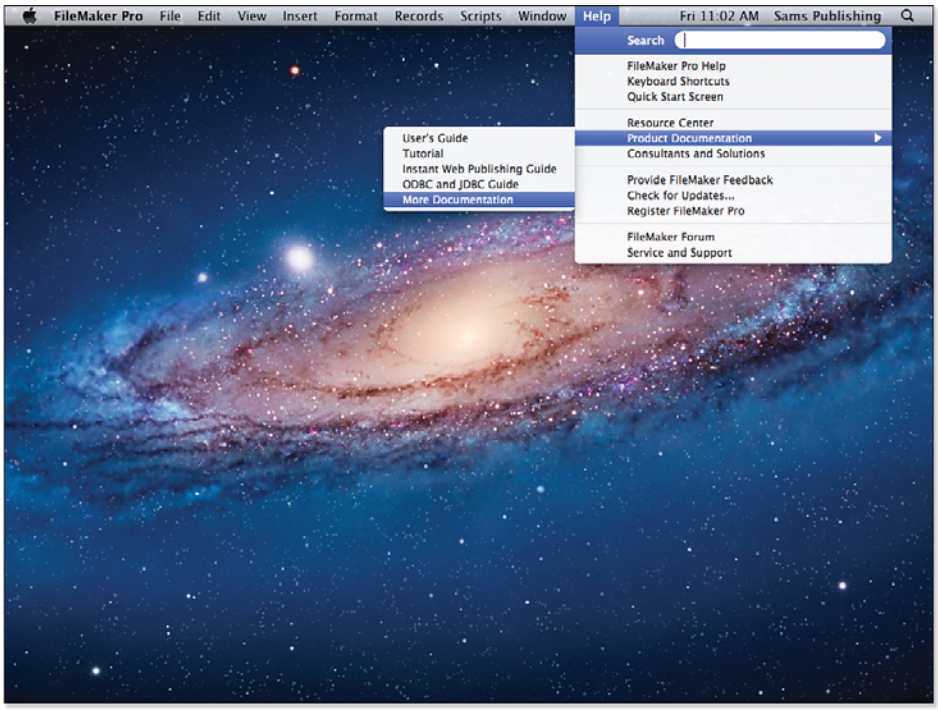


Figure 2.3 FileMaker Pro's Help menu is just the beginning of built-in assistance. It provides you with a variety of assistance, ranging from simple keyboard commands all the way to developing your own solutions.

Understanding FileMaker Pro Features

FileMaker is a vibrant ecosystem of software products, developers (at FileMaker and at third parties), designers of FileMaker databases, and the users thereof. The heart of this ecosystem is FileMaker databases, which can include user interfaces, scripts, and other elements that work with the FileMaker software to help people manipulate the databases. On the software side, there are two major products, each of which has two versions:

- FileMaker Pro and FileMaker Pro Advanced let you build and use databases (in database-speak, the *schemas* of your databases); you also can build interfaces, scripts, and other elements with these products. These are the only tools that let you create FileMaker databases.

  In addition to building databases, you can share them with other people over a network with these products. Sharing is limited to nine other people (but read on for details of FileMaker Server).

- FileMaker Go lets you access your databases from iOS devices. You can use iTunes or email to install your FileMaker databases on an iOS device, and then you can use the database anywhere. You also can use FileMaker databases that are published on the Web using FileMaker Pro or FileMaker Server from FileMaker Go. This scenario means that changes you make to the database are reflected immediately across the Web or network, and those changes show up on other FileMaker Go clients as well as in the view of the database that people see in FileMaker Pro.
- FileMaker Server and FileMaker Server Advanced let you share databases over a network with up to 250 people running FileMaker Pro. You also can publish FileMaker databases so that people can access them over the Web with web browsers (in other words, they don't need to have their own copies of FileMaker Pro). FileMaker Server Advanced has no fixed limit on users: The actual number is restricted only by your hardware, but 250 has been tested.

  FileMaker Server allows web publishing with ODBC/JDBC and PHP. This lets people access your solutions with a web browser. This means that if someone wants to access a FileMaker database that you publish, they can do so with FileMaker Go on an iOS device, but on other mobile devices, the built-in browser can do the trick.

  FileMaker Server Advanced implements an additional technology called Instant Web Publishing, which lets people use their browsers to access screens that look very much like the actual FileMaker Pro interface (very, very much like the FileMaker Pro interface).

There is an additional component of FileMaker that you can create using FileMaker Pro Advanced. After you have created your FileMaker database and its interface, you can generate a runtime solution with FileMaker Pro Advanced. That solution can run on OS X or Windows (you must create separate copies for each operating system), and users can get almost the entire FileMaker Pro experience without installing a copy of FileMaker Pro. No restrictions on the distribution of this software exist, but there are some restrictions on the features that are supported. Perhaps the most significant feature that is not available in a runtime solution is networking: Runtime solutions are single-user solutions.

Understanding FileMaker Databases

FileMaker databases have evolved over the years. Today they consist of several components, not all of which might be present in every database you use:

- A database contains one or more *tables* that actually contain the data. The next section describes tables in more detail.
- A database can contain references to tables in other FileMaker database files or in other databases accessed via ODBC.
- A database can contain *layouts* that provide the user interface with which to access tables either in this database file or in other locations.
- There might be *scripts* that contain commands created to automate various processes. Scripts are often connected to layout elements, such as buttons, but they can be invoked automatically when a database is opened or closed, as well as when certain other events occur (or are *triggered*).

- The database includes security features in the form of user identifiers and passwords as well as descriptions of what privileges each set of users has to access the database and its components.
- A variety of other, smaller components that support these major features are also part of the database.

With the exception of tables in other files or databases referenced from a database, all the database elements are stored in a single file that can be moved from place to place.

If you have references to other files, you might break these links if you move the database file. For more information, ***see*** *Chapter 7, "Working with Relationships."*

Understanding Tables

Before FileMaker 7, no serious distinction was made between database files and tables; this was one way in which FileMaker differed from other relational databases. Beginning with Version 7, FileMaker could handle multiple tables within a single database file. The number of tables that a single database file can contain is essentially limitless.

A *table* is a collection of data—the records and fields described in the following section. Data in a given table is all in a single logical format. The simplest case of a Starter Solution is Contacts: It contains a single table called Contacts.

A large part of the power of a relational database such as FileMaker Pro is its capability to relate data in one table to another. The Projects Starter Solution, for example, contains three tables: Projects, Tasks, and Personnel. As you might expect, tasks are part of projects, and personnel are assigned to tasks. This is all governed by the design of the database: FileMaker keeps the relationships organized.

Tables need not be in the same database file to be related to one another, but it makes sense to combine tables that are closely related in a single database file. For example, if you have a complex Contact Management database, you might have tables for names, for addresses, and for phone numbers, with all those tables being related to one another to combine the data for a single contact. In FileMaker Pro, every layout is based on a single table, although it can use data from other tables as well.

To dig deeper into working with multiple tables and understanding relational data models, ***see*** *Chapter 6, "Working with Multiple Tables," and Chapter 5, "Relational Database Design."*

Understanding Records and Fields

A table stores information about many items with similar data characteristics: many to-do items, many contact items, and so forth. Each of these items is called a *record* (sometimes *data record*), or, in relational database parlance, a *row*. Each record or row has data elements that are called *fields*, or, in relational database parlance, *columns*. Fields for a contact record can include a name, an address, and the like; for a to-do item, fields might contain a due date and the name of the task to be done.

Particularly if you use the row/column terminology, it is easy to think that you are talking about a spreadsheet, but a database is much more powerful than a spreadsheet. Much of that power comes from two major aspects of a database:

- You can describe the database so that the data it contains must adhere to strict rules. Numbers must be numbers, if you choose to enforce such a rule, and values must be within a specific range of values if you choose another type of rule.
- Furthermore, you can set up rules to relate data within the database so that, for example, the person charged with carrying out a to-do item must be someone who is already entered into the contacts database. You'll see how to create such relationships shortly.

The combination of these two aspects of databases—along with many more—make them more powerful than spreadsheets.

The FileMaker Pro User Interface

The FileMaker Pro interface consists of basic elements:

- **Layouts** display data and let you edit it.
- **Modes** change the behavior and appearance of the interface to let you browse data, find specific data, display reports for printing or interactive use on the screen, and create or modify layouts.
- **Views** are available in both Browse and Find modes. They let you see one record at a time, a list of records, or a spreadsheet-like table view of records.

Each of these interface elements is described in this section. In a later section, the Status toolbar and associated menu commands are described. They let you control the user interface itself, switching among layouts, modes, and views as well as navigating through your database.

Figure 2.4 shows the FileMaker Pro user interface with the Status toolbar at the top of the window (you can show and hide it). The main part of the window is a layout displaying data from the Assets Starter Solution.

There are two parts to the status toolbar, one above the other. The main part of the Status toolbar shown in Figure 2.4 has navigation tools, buttons to create and delete records, as well as buttons to find and sort data. Below it, the narrower *Layout bar* lets you select layouts and control how to view the data (as a form, list, or table). Buttons let you enter Preview mode, show or hide the *Formatting bar*, or edit the layout. These features are described in more detail in "Using the Status Toolbar," later in this chapter.

Layouts

Most FileMaker Pro databases open to a data-entry layout, such as that shown in Figure 2.5. Generally, you have access to fields, commonly designated by a field border of some kind, including rounded corners, where you can set the corner radius (beginning in FileMaker Pro 12). Fields are usually labeled. FileMaker Pro provides some specialized data-entry tools, such as the calendar shown in Figure 2.5.

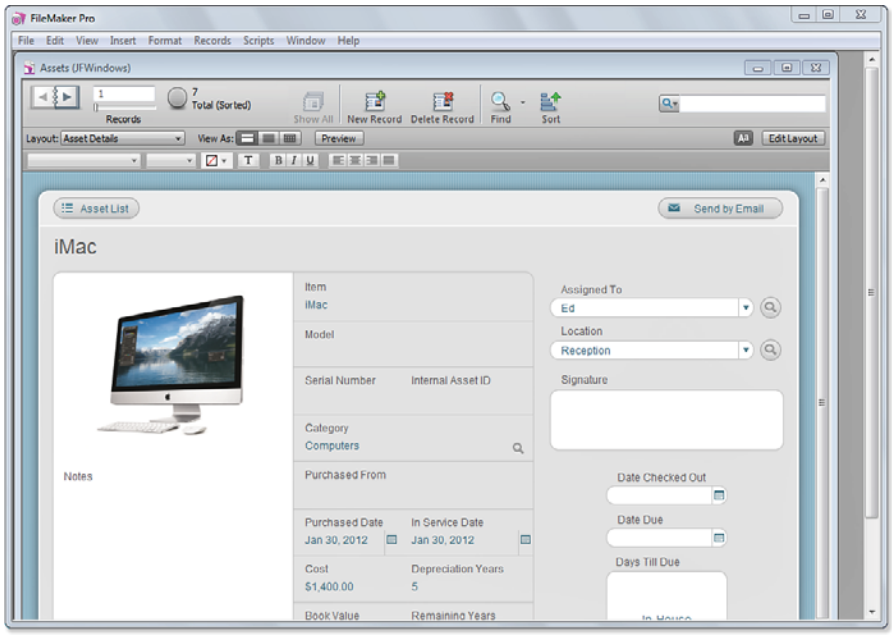


Figure 2.4
The FileMaker Pro user interface provides a Status toolbar at the top of the window.

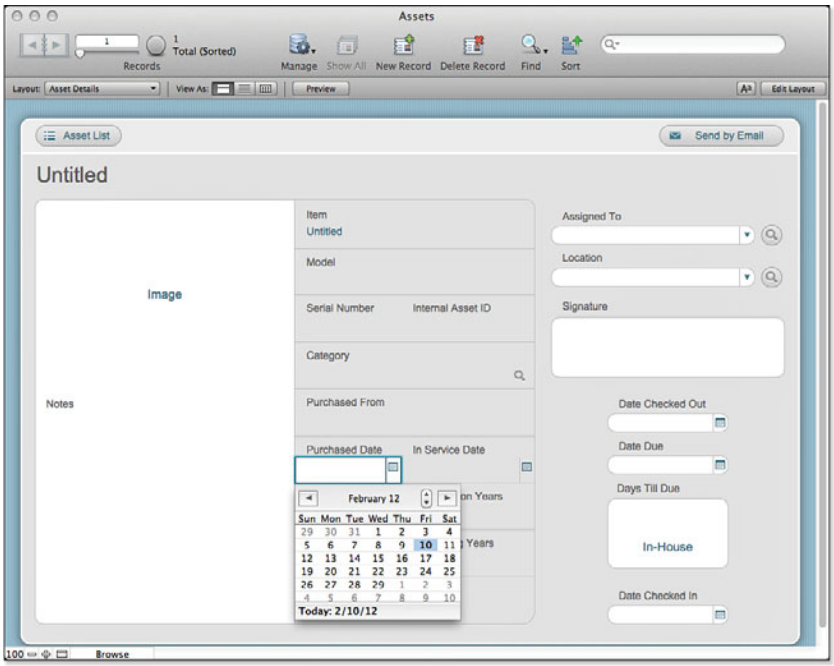


Figure 2.5
FileMaker Pro has specialized data-entry tools.

Figure 2.6, based on the Meetings Starter Solution, shows a tab control at the right of the layout. There are two tabs: Topics and Action Items. Each tab displays its own set of data when it is clicked. This makes for a very efficient use of the screen.

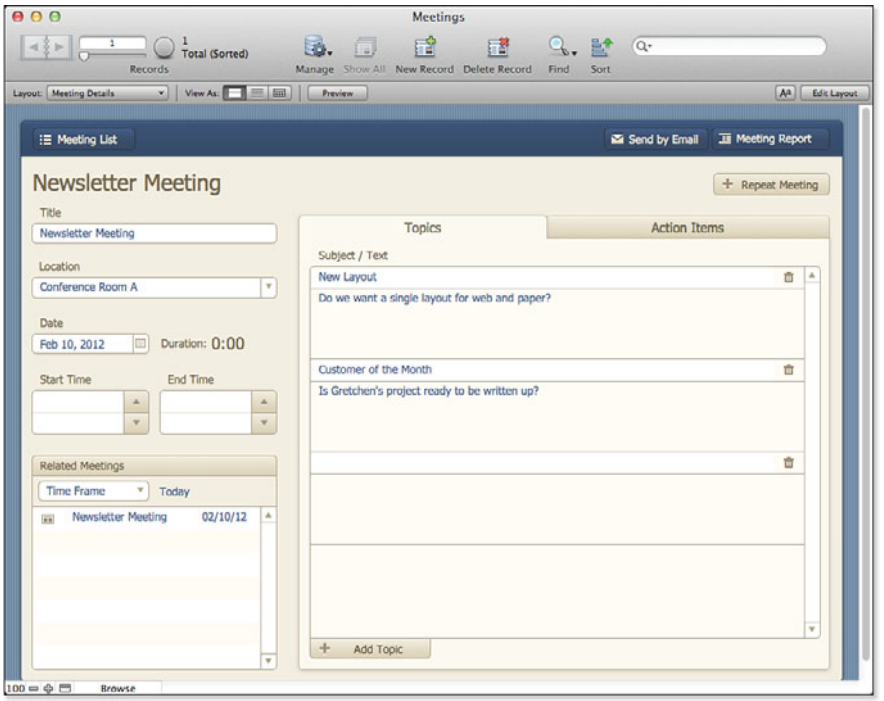


Figure 2.6
Use a tab control to save space.

Developers often provide tooltips that appear when you hover the mouse pointer over a specific layout element.

*For information on creating Layouts, **see** Chapter 4, "Working with Layouts."*

FileMaker Pro Modes

At any given time, you interact with your FileMaker Pro databases via one of four modes. At times, developers choose to tailor a layout for use with a specific mode, but more often than not, you can use layouts effectively with all four modes. To switch between modes, use the View menu or the Status toolbar, described later in this chapter. To familiarize you with the four modes, here's a simple description of each:

- **Browse mode**—Browse mode is FileMaker Pro's primary mode, where all data entry occurs, and generally is the principal mode you'll use in a given solution.
- **Find mode**—Here, you create and then perform find requests to search for specific sets of records.
- **Preview mode**—When preparing to print from FileMaker Pro, you can opt to switch to Preview mode to see what a given layout will look like after it is printed.

Certain aspects of reports such as subsummaries were only available in Preview mode until FileMaker Pro 10. Now, you can interactively modify data in reports in Browse mode and see the subsummaries dynamically respond to the changes.

The primary information on reports, including interactivity in FileMaker Pro 10, is provided in Chapter 10, "Getting Started with Reporting and Charting."

- **Layout mode**—It is in Layout mode that a great deal of development occurs. Here, developers can manipulate all the elements of a given layout, including controlling all the things that appear on that layout.

Views

In addition to the modes of FileMaker Pro, there are three views as well. A *view* is a particular way of displaying record data on the screen. To change among them, use the View menu. As you will see later, layouts can have headers and footers; the view refers to the layout shown between the header and footer. These are the three views:

- **Form view**—This view enables you to see and manipulate only one record at a time, as shown previously in Figure 2.5.
- **List view**—Here, you can display multiple records. At any given moment, you are working with only one specific record while still being able to scroll through the rest (see Figure 2.7). A black bar at the left of the Layout area shows you which record is active.

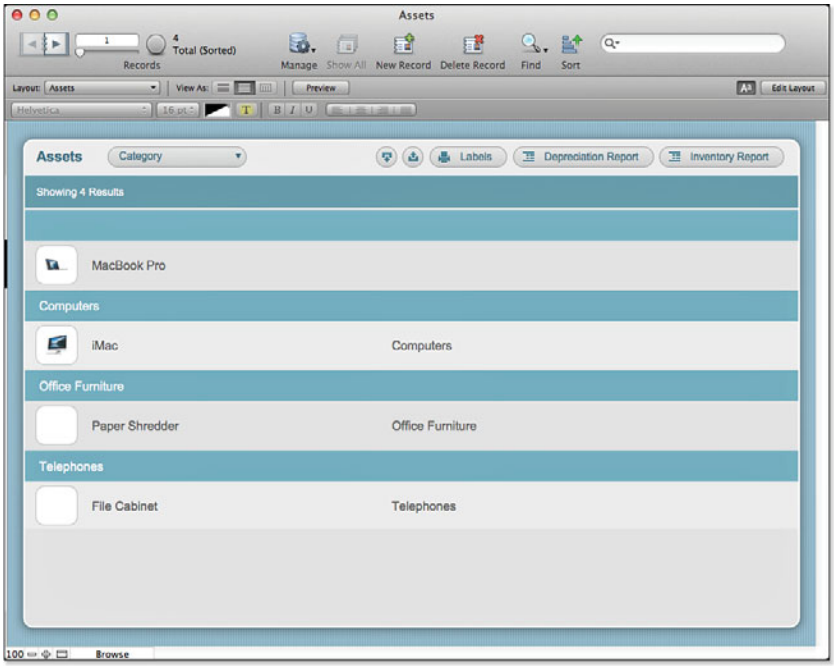


Figure 2.7
List view lets you work with more than one record at a time.

- **Table view**—Table view simply displays the raw data for a given record (depending on what fields have been placed on a layout). It looks quite similar to a spreadsheet application (see Figure 2.8). You can move, resize, and sort the columns by clicking the column headers.

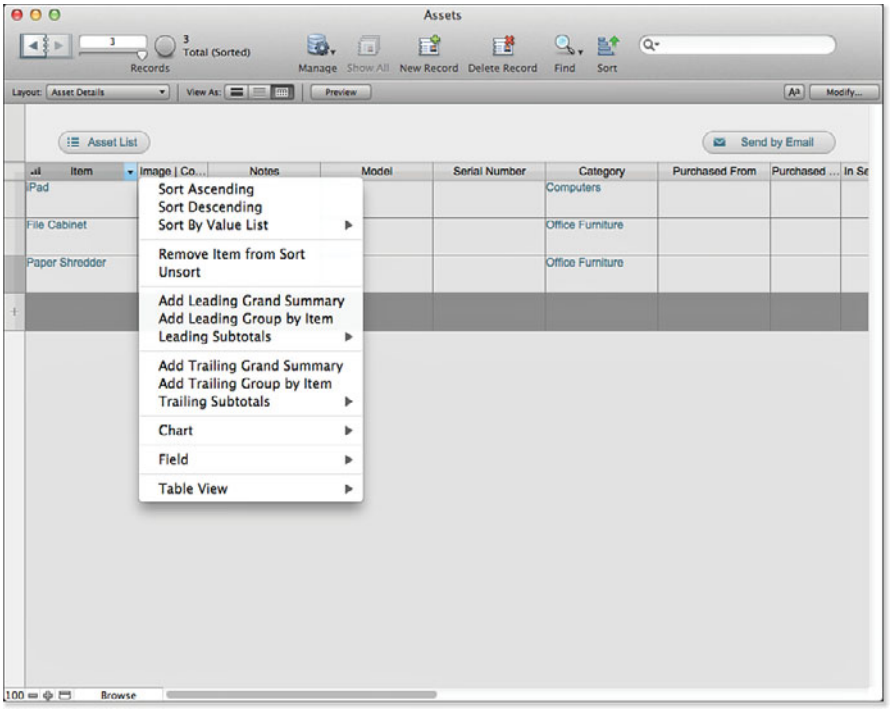


Figure 2.8
In Table view, you can resize, move, and sort with column headers.

Buttons

Notice that we've largely been talking about fields on layouts. Most FileMaker Pro solutions also include buttons. Figure 2.5, earlier in this chapter, shows a few examples.

Buttons can come in all shapes and sizes in FileMaker Pro. Text can be a button, a field can be a button, and even just a mysterious blank area in the middle of a layout can be a button (although that is a very poor interface).

In Figure 2.6, the top of the layout contains buttons named Meeting List and Send by Email. Buttons such as these can invoke standard FileMaker Pro commands or scripts that you write yourself.

Today, FileMaker solutions are often designed for use not only on desktop and laptop computers but also on mobile devices, such as iPhone and iPad. As you start to design your own layouts, remember that on mobile devices there are no menu commands; in these cases, your buttons make your solution easier to use.

tip

In general, it is a good idea to avoid *decoration* in layouts—interface elements that do not immediately suggest that they are useful. For example, a small envelope and arrow icon next to an email field can be configured as a button to send email to the associated address. This is not decoration, and, because the layout contains no irrelevant decoration, the user can reasonably assume that the envelope and icon mean something.

Buttons trigger actions, often by launching scripts that developers write; these actions are usually specific to a given FileMaker database. Buttons can perform dozens of actions, such as creating a new record, deleting a record, navigating to another layout, performing a calculation, performing a find request, controlling windows, and even spell-checking and emitting a simple beep. The possibilities are endless.

You'll have to become familiar with the specifics of a given FileMaker Pro solution to come to understand what its buttons do. A good interface suggests what items are hot; furthermore, information the developer can provide, such as tooltips for each button, should assist you. If all this fails, the person who built the system should have those details or should have provided some form of training or documentation.

Using the Status Toolbar

The Status toolbar replaces the old Status Area and provides much more powerful control and feedback. The Status toolbar at the top of FileMaker Pro windows combines controls and information displays in a compact structure. Most of the objects in the Status toolbar accomplish tasks that can also be accomplished with menu commands and their keyboard equivalents.

The View menu, shown in Figure 2.9, lets you control how the current window is displayed.

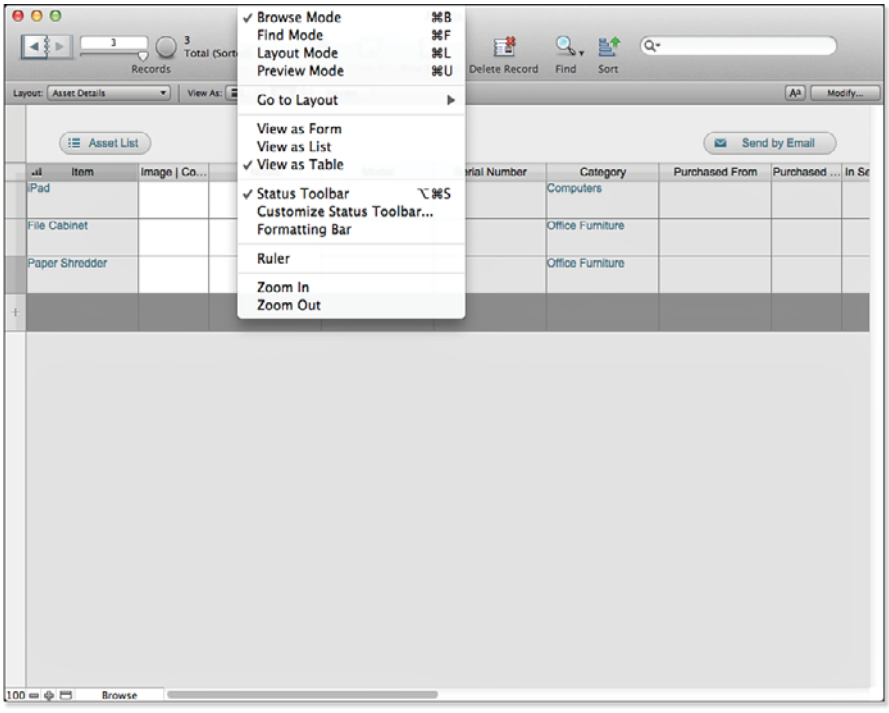


Figure 2.9
The View menu controls the user interface.

The Status toolbar itself can be shown or hidden for any window. Simply select the window and choose Status Toolbar from the View menu.

As you can see in Figure 2.9, you can control the visibility of the Status toolbar separately from the visibility of the Formatting bar. In Figure 2.10, you can see the Formatting bar at the top of the window; the Status toolbar is hidden.

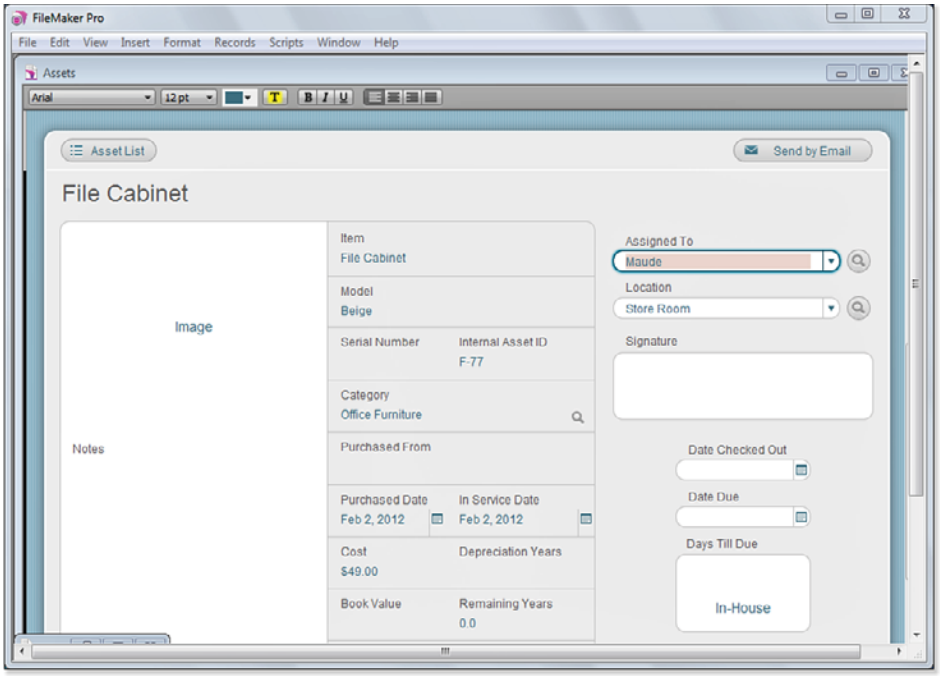


Figure 2.10 The formatting bar lets you format selected text.

The formatting bar is available only when text is selected. This means that its controls are dimmed if you select a field containing a graphic. In Browse mode, which you use to enter data, only a single text field can be selected at a time. Whatever text you have selected in that field is affected by the controls in the formatting bar.

tip

If you select a text field and do not select any text within it, you will set the formatting for the insertion point and any text that you subsequently type. To change the formatting of all the text in a text field, double-click in the field to select the field and all its text.

Customizing the Status Toolbar (OS X)

Toolbars are an important part of the OS X interface; they are supported deep within the operating system.

When an application provides a toolbar, it generally enables you to customize the toolbar. You do this by choosing View, Customize Status Toolbar (the command is available only when the Status toolbar is shown). You will see a sheet with the customization options, as shown in Figure 2.11.

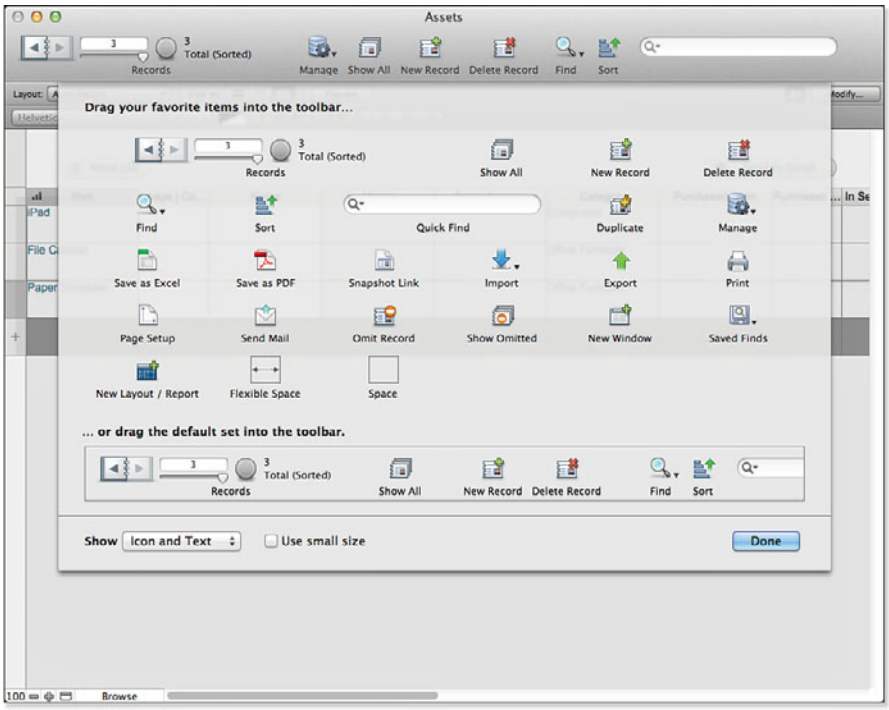


Figure 2.11 Customize the Status toolbar in OS X.

You can rearrange the items in the toolbar by dragging them back and forth. Remove items by dragging them out of the toolbar, and add new ones by dragging them up into the toolbar. If you want to revert to the original toolbar, the default set at the bottom of the customized display moves as a single unit when you drag it to the toolbar.

Just above the default set are two important special items: a space and a flexible space. You can insert them as many times as you want into the toolbar to organize it.

Finally, at the bottom of the customization display, you can choose small or large icons, icons alone, icons and text, or text only.

Each icon can appear only once in the toolbar. You do not have to worry about putting too many icons into the toolbar. If the window is narrower than the toolbar is wide, a double arrow appears at the right of the toolbar, and the icons that do not fit are shown off to the right of the window as text only, as Figure 2.12 demonstrates with the New Record and Delete Record commands. In addition, commands (icons or text) are dimmed if they are irrelevant.

Your settings for the Status toolbar apply to the database; all windows for that database reflect your settings for its toolbar. Toolbars for windows based on other databases are not changed.

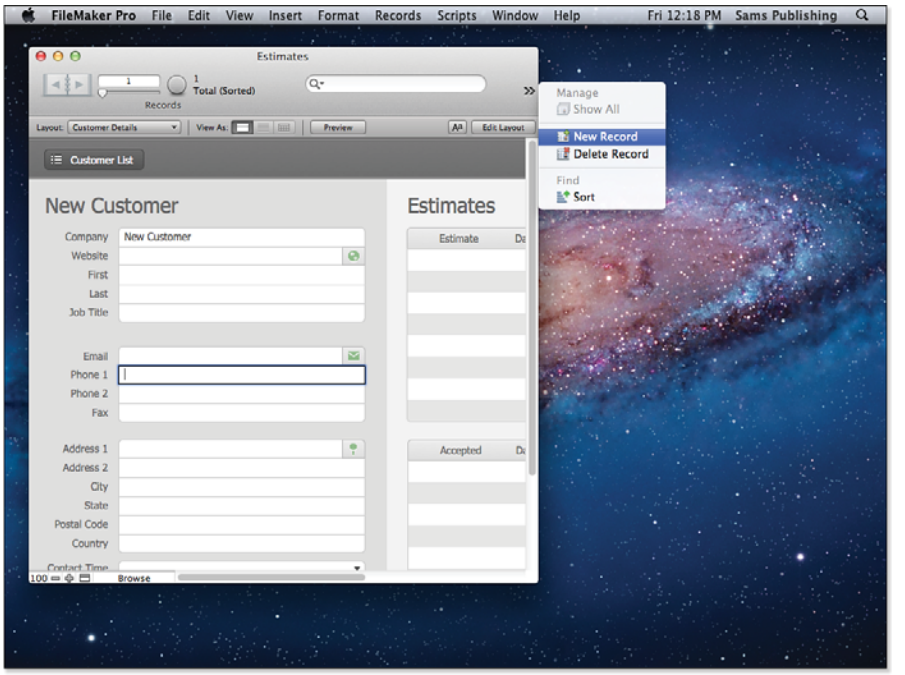


Figure 2.12
Toolbar commands can extend beyond the window.

Customizing the Status Toolbar (Windows)

Toolbars are one of the few areas in which the Windows and OS X interfaces differ. As you can see from the figures in this chapter, the Status toolbar itself looks much the same on both operating systems, but the way in which you customize it differs.

On Windows, you begin by choosing View, Customize Status Toolbar, just as on OS X. This opens the dialog shown in Figure 2.13.

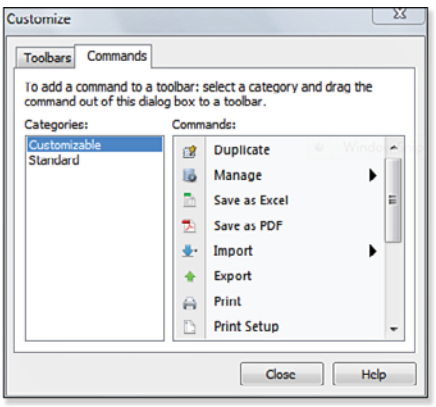


Figure 2.13
Customize the Status toolbar on Windows with Customizable commands.

The first tab, Toolbars, lets you select the toolbar to customize. There is currently only one choice: Status. The second tab, Commands, shown in Figure 2.14, lets you choose either the customizable commands or the standard commands; the standard commands are shown in Figure 2.14

In either case, drag the command up into the toolbar.

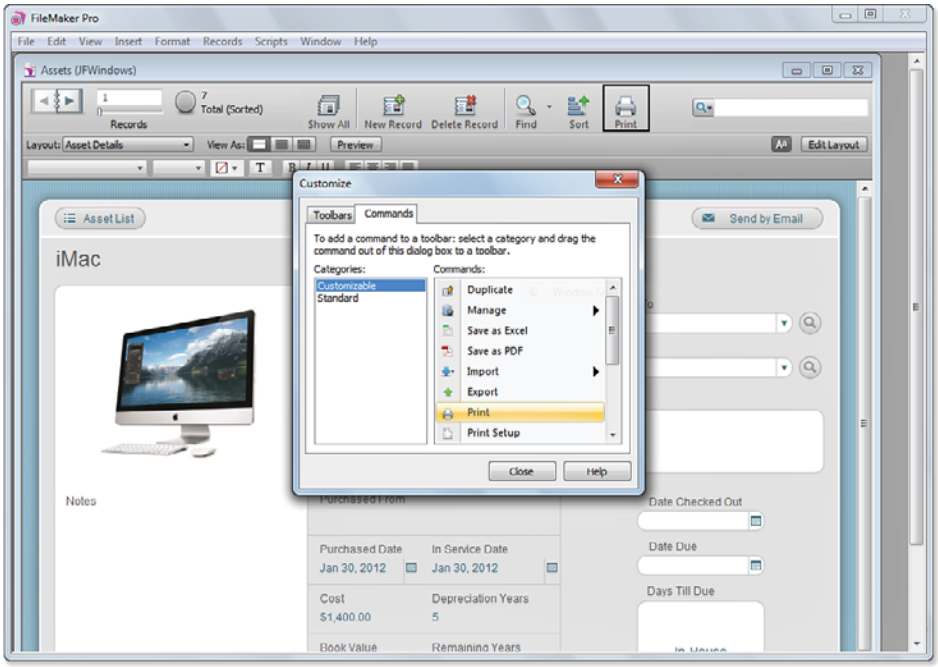


Figure 2.14 Customize the Status toolbar on Windows with Standard commands.

Working in FileMaker Pro

The following sections walk you through working in some typical FileMaker Pro situations and address many of the common tasks you must be able to perform.

*For more information on using other tools to access the data, **see** the chapters in Part IV, "Data Integration and Publishing," which begins on **p. 519**.*

Opening a Database

The first step in working with FileMaker Pro, obviously, is opening a database. FileMaker Pro databases can live in various places. They can sit on your own computer, just as any other document might; they can be hosted by another computer; or they can be

note

Over the past few revisions of FileMaker Pro, functionality has been added that lets you create databases in FileMaker but access them through other applications. With web publishing, you can use a browser to access FileMaker data. You can use ODBC import and export to share data with SQL-compliant applications, and you can even import and export Excel spreadsheets. This section focuses on working with FileMaker data using FileMaker Pro itself.

served by FileMaker Server. On any of those computers, they can be housed on shared volumes or external devices (although there are constraints for the FileMaker Server database locations).

Local Files

Opening a local file is a simple matter of double-clicking its icon in either your Windows environment or the OS X Finder. You can also use FileMaker Pro's File, Open command or the Quick Start screen, as described previously.

You can use the Open command to navigate to any database file to which you have access—whether it is on your own computer, somewhere else on your network, or on the Internet—if you can get to it from the Open File dialogs shown in Figure 2.15.

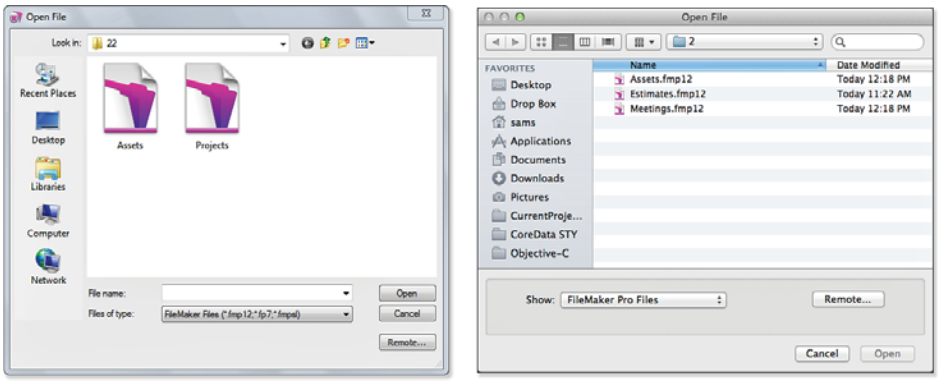


Figure 2.15 Use the Open command to open a FileMaker Pro database.

Note that in addition to the Open button, there is also a Remote button; it is discussed in the next section.

Remote Files

Working with remote files requires connecting to a server. That server could be a database hosted on FileMaker Server (the software that allows you to host a FileMaker database for use across a LAN or WAN by up to 250 users) or a FileMaker database file that is set to multiuser and running in FileMaker Pro on another person's workstation. After you connect to a remote database, everything works just as it would with a local connection (although over a busy network, there might be a slight lag in response). The only distinction that you will note is that the title of the window shows not only the name of the database but also—in parentheses—the name of the server on which it is hosted. Compare the title of the window shown in Figure 2.16 remotely with the same database shown in Figure 2.7.

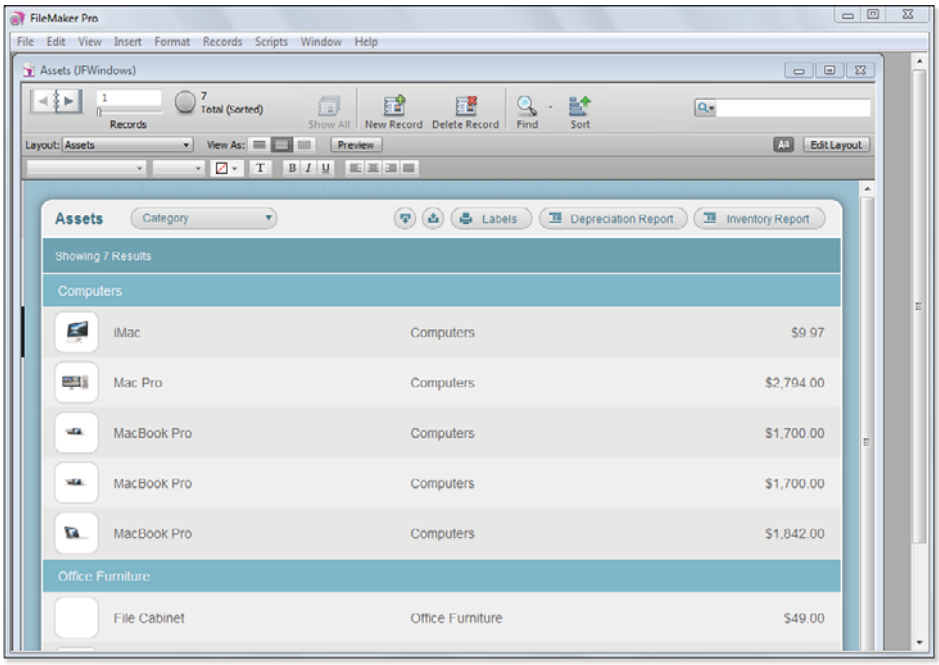


Figure 2.16
A FileMaker Pro database opened remotely shows the name of the host in its title.

Exploring Remote Connections

A remote connection is one that lets you connect to a copy of FileMaker Pro from your own copy of FileMaker Pro. If you use the standard Open dialog as described in the previous section, you open a database with your own copy of FileMaker Pro, whether the database is on your computer or on the network. When you use the Open Remote dialog by clicking the Remote button in the lower right of the Open File dialog shown in Figure 2.17 or a remote favorite from the Quick Start screen, the database to which you connect has already been opened by a copy of FileMaker Pro or FileMaker Server. Actually, it is to that copy of FileMaker Pro that you are connecting, and through that copy of FileMaker Pro or FileMaker Server to the database.

Only one copy of FileMaker Pro or FileMaker Server can attach to the actual database file at a given time; all other users (if any) attach to that copy of FileMaker. The same applies to users of FileMaker databases connecting through ODBC or the Web: They are connecting to the copy of FileMaker Pro or FileMaker Server that has the database open. (Sharing databases with FileMaker Pro is possible on a small scale, but performance significantly improves with dedicated hardware resources and FileMaker Server.)

Although the most common scenario involves sharing databases across a network or the Internet, other configurations are possible. You can use Open Remote to open a database on your own computer as long as another copy of FileMaker Pro on your own computer (perhaps FileMaker Pro Advanced) has opened it. In that case, both the server and the client are the

same computer, but there is no question which is the server (the version of FileMaker Pro that opened the database with the Open command) and which is the client (the version of FileMaker Pro that opened the database with the Open Remote command).

While discussing these matters, it is worthwhile to point out that *server* is sometimes used to designate one or more computers on a network that provide shared resources such as common disks. The term *server* might also refer to one or more computers that provide shared services such as email, web hosting, and the like on a network. In the FileMaker context, *server* refers to a computer running either FileMaker Server or a version of FileMaker Pro or FileMaker Pro Advanced and configured to share databases. It can be located on a server (hardware) that runs server software (network administration, email, and the Web), but it might be any computer on the network. In fact, there are often significant advantages in a high-volume environment to using a dedicated computer to serve FileMaker databases so that no other demands are made on that computer's disk and processing resources.

*For more information on FileMaker Server, **see** Chapter 27, "FileMaker Server and Server Advanced."*

To open a remote database, click the Remote button in the Open File dialog, or choose Open Remote from the File menu. As you can see in Figure 2.17, you can choose from those hosts available to you locally (those on your network, within your domain in corporate environments, or accessible on the Internet), or you can navigate to a particular server via a Lightweight Directory Access Protocol (LDAP) server. You can also view servers and databases that you have previously marked as favorites. Finally, you can type in the address of a file in the box at the bottom. Make certain in this case to use the prefix **fmnet**.

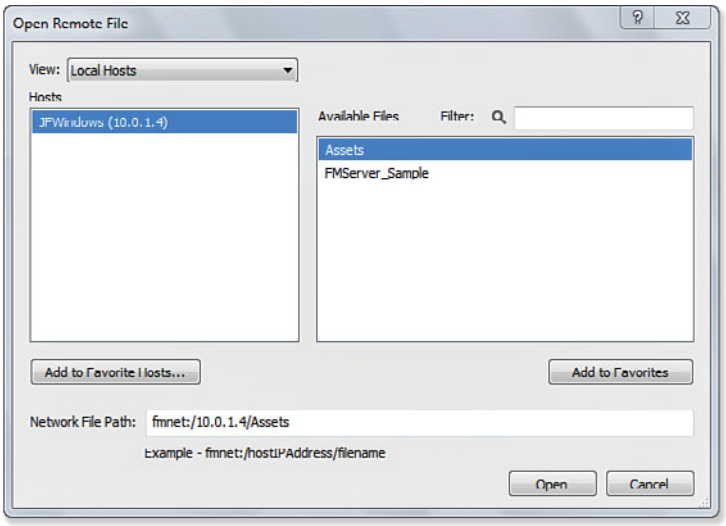
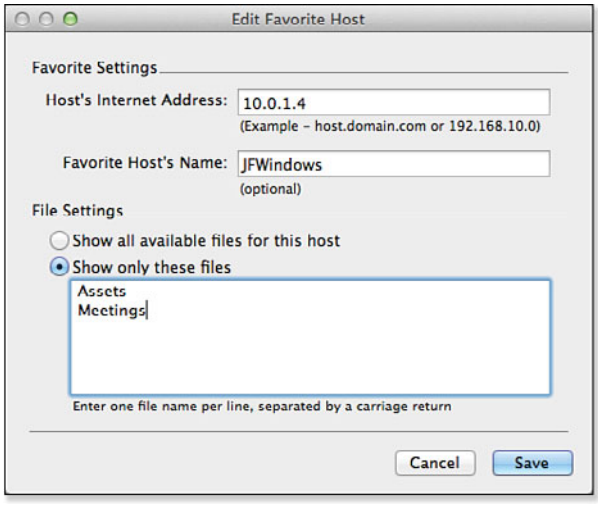


Figure 2.17
Use the Open Remote File dialog to open a database on a LAN, in a corporate domain, or (with a proper IP address) across the Internet.

When you choose Local Hosts, you will first see a list of all the FileMaker servers running on your local network. It might take some time for FileMaker Pro to locate all these servers, so be patient. After you click a hostname (either a local host, an LDAP host, or a Favorites host), FileMaker Pro interrogates that host for the list of databases to display in the list at the right of the dialog. This, too, might take some time. As you navigate through hosts and to an individual database, the fmnet address at the bottom of the dialog fills in automatically.

If you add a database or a FileMaker host to the favorites, it shows up not only in the Favorites of this dialog, but also in the Favorites for the Quick Start screen (shown previously in Figure 2.4). When you select a host or database, you can click Add to Favorites to open the dialog shown in Figure 2.18. It is a good idea to rely on favorites—particularly for remote databases where the network file path might be a lengthy string of numbers and/or words that are easier to select from a Favorites list than to retype.

Figure 2.18
Use Favorites to organize your databases and hosts.



note

If you are browsing files on a remote FileMaker server, you will see all the opened FileMaker databases that the developer has specified should show up in the Open Remote dialog. You will not see closed databases or those marked not to be listed in this dialog.

tip

If you want to open a database not shown in the list, you can quickly create the appropriate fmnet address. Select, with a single click, a database shown in the list and simply erase the name of that database from the end of the address; then type in the name of the database that you want to open.

Working with Records

The steps involved in creating and deleting records in FileMaker Pro are simple. Under the Records menu, choose New Record, Delete Record, or Duplicate Record. Notice also that there's a Delete All Records option. For now, let's explore how to take care of simple data entry. Many solutions (including the Starter Solutions) provide buttons—usually at the top of the layout—to create and delete records; some might also provide a duplicate record button.

If you are in the midst of entering data in a record and want to undo the entry, use the Revert Record command under the Records menu. A record is saved—or *committed*—automatically when you click outside a field for the first time (or in another field), change modes, change layouts, or press the Enter key. FileMaker Pro uses the term *commit* to indicate when a record is posted, or saved, to your database. Using the Revert Record command before committing a record allows you to roll back all the changes you've made, returning that record to its last committed state.

If Revert Record doesn't seem to do anything, ***see*** *"Reverting Records" in the "Troubleshooting" section at the end of this chapter.*

If you have trouble with data you believe to be lost, ***see*** *"Data Loss" in the "Troubleshooting" section at the end of this chapter.*

tip

The flow of processing in FileMaker Pro differs from that of a web interface. In a web-based application, including FileMaker's Instant Web Publishing and Custom Web Publishing, all data is shown on the screen and is sent when you click a Submit button—that is the moment when data is committed. When working directly with FileMaker, you might be committing (or submitting) data many times as you work on the record. In practice, this is usually not a serious issue, just a reflection of the differences in how two technologies handle the issue of data updates.

caution

Keep in mind that even though there's an Undo command in the Edit menu, it doesn't work at the record level. After a record is committed (saved to the database), it and its changes are a part of your database. However, until you click out of the field, you can undo and redo changes to your heart's content. Also remember that after you delete a record, it's gone forever.

note

You never need to save a FileMaker Pro database. As users commit records, those records are automatically stored in the database file. If you want to save a copy of your database or create a duplicate for backup purposes, the Save As option under the File menu will serve.

Working with Fields

If you are used to other productivity applications or have ever filled out a form on the Web, you should find data entry quite familiar in FileMaker Pro. Fields generally look like embossed or bordered areas with labels off to one side or the other, underneath, or above the field. Keep in mind that developers control the look and feel of their systems, so it's entirely possible that someone could build a database with no labels, fields that are the same color as their background, and white text on a white background. When a field is being actively edited, its border is highlighted (generally darker), and the other fields on a given layout are shown with less prominent highlights, indicating that you're in the midst of editing a record (see Figure 2.19). Editing fields is as easy as clicking into them, typing some text, and clicking out again. (As with many aspects of FileMaker, these behaviors are customizable in Layout mode.)

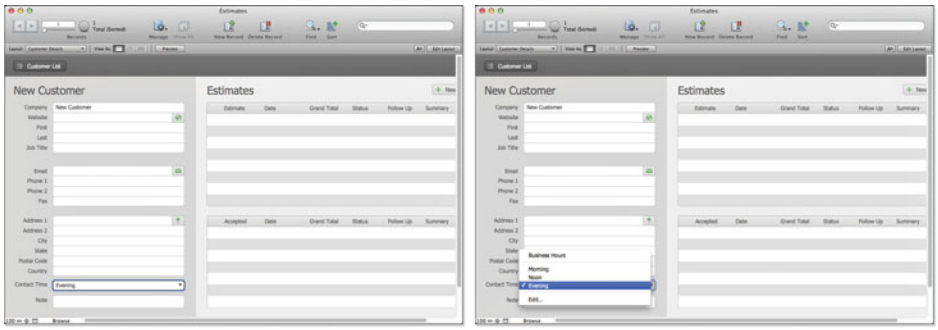


Figure 2.19
A downward-pointing arrow might indicate a drop-down field.

FileMaker includes spell-checking for text fields, although you can turn it off if you want. When a field is active—when the cursor is in the field itself—FileMaker underlines in red any words it concludes are misspelled. If you right-click (or Control-click on a Mac with a one-button mouse) the word, you can choose from among possible other spellings or save a word to your local dictionary file.

Moving from field to field can be managed on your keyboard if you simply press the Tab key. Some solutions also support the Return and Enter keys. You can, depending on how the developer of a database has set things, tab from button to button or tab panel to tab panel. To execute an action associated with an active button or tab, press the Enter key or spacebar on your keyboard.

*For a discussion of how to control object behavior from a development perspective, **see "Working with Fields," p. 44**.*

You'll work with a few different formats of fields in FileMaker Pro:

- **Edit box**—This allows standard keyboard entry and sometimes includes a vertical scroll bar.
- **Drop-down list**—When first clicking into a field, you are presented with a list of options from which you can select. Alternatively, you can type directly into the field.
- **Pop-up menu**—A pop-up menu is similar to a drop-down list, except that a pop-up menu does not allow typing directly into the field.
- **Check box set**—Check boxes allow multiple values per field.
- **Radio button set**—These are similar to check boxes, with the difference that they are mutually exclusive. A user can select only one value at a time.
- **Pop-up calendar**—Some date fields might open to show a calendar that you can page through from month to month. To input a date into your date field, click a specific day.

caution

Shift-clicking allows a user to select multiple values in certain input types, such as pop-up menus and radio buttons. Selecting multiple values in a pop-up menu or in radio button sets is generally a bad idea. You will end up with unpredictable results because you're making an exception to a formatting choice meant to allow for only one value in a given field.

As the FileMaker Pro interface has evolved, new interface elements have been introduced. Many of the traditional interface elements immediately reveal their functionality: Radio buttons, for example, are instantly recognizable. Some newer features now allow developers to provide hints of functionality that the interface provides. Often this is done by using a light gray for some of the interface elements. Drop-down menus are good examples of this. Even when the drop-down menu is not selected, the small downward-pointing arrow at the right of the field is visible. For the purpose of this book, the arrow is shown darker than it would be in most interfaces; it is usually quite subtle. And do remember that this is the developer's option: There might be no indication of the field's capabilities until you click in it and activate the drop-down menu.

A date field can contain a pop-up calendar, as shown in Figure 2.20. As is the case with the drop-down menu shown later in Figure 2.21, the little calendar icon at the right of the date field might be shown in light gray even when the field is not active.

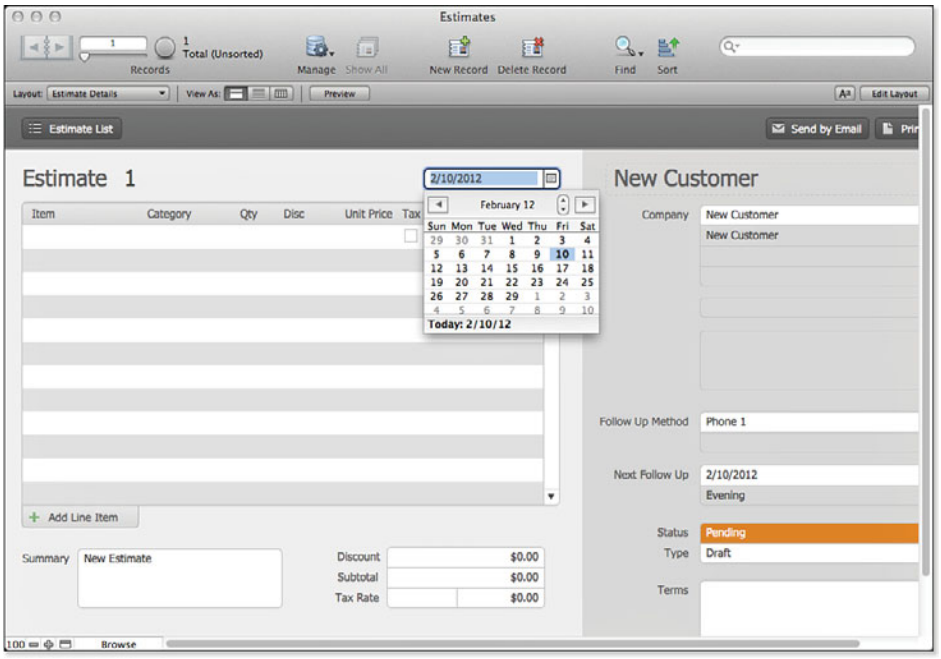


Figure 2.20 Pop-up calendars can ease entry of dates.

Data in Formatted Fields

You might find it helpful to understand *how* multiple-value data is stored in fields: Remember that check boxes, radio buttons, drop-down lists, and pop-up menus are all nothing more than data-entry assistants. The actual data stored is a collection of values delimited by line returns. This means that you can accomplish the same result, from a data perspective, by simply entering a Return-delimited list of values into your fields. This is an important point for you to remember when performing find requests, which we cover later in this chapter.

To understand more about how multiple values in a field can lead to relational data structure problems, ***see*** *Chapter 5, "Relational Database Design."*

Modifying Value Lists

Often, you might need to add new values to a value list—the list that is used to create drop-down lists and pop-up menus, check boxes, and radio button items. Developers have the option of including an Edit option at the bottom of a drop-down list or pop-up menu. Selecting Edit brings up a dialog that you can use to change or add to a list as needed (see Figure 2.21). You can add a separator line to the list by using a single hyphen, as shown in the figure.

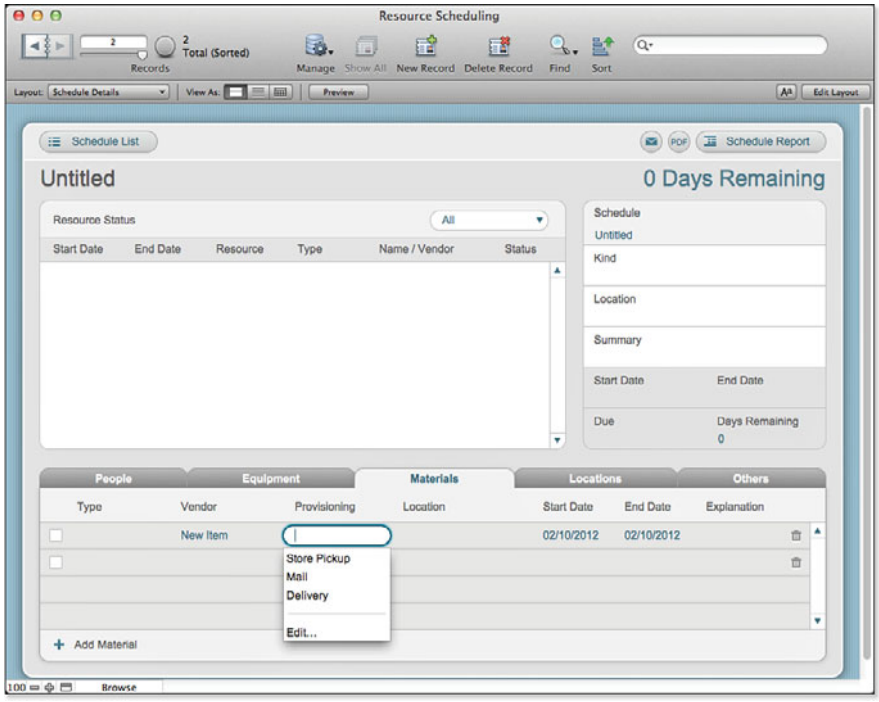


Figure 2.21
Editing value lists is a simple way to fine-tune a database to your specific needs without having to dig into programming.

To edit the items in a value list, simply type text into the Edit Value List dialog, followed by a carriage return.

note

Keep in mind that just because you replaced an old menu item with a new category—for example, "autos" became "cars"—doesn't mean that you changed the actual values stored in your database's records. Remember that field formatting is nothing more than a data-entry assistant. By changing the assistant menu, you have *not* changed any data stored in your database.

Using the "Other" Value in Value Lists

Radio button sets and check boxes work a bit differently than drop-down lists and pop-up menus. Developers do not have the choice to add an edit function to these formats; rather, they can include an Other option. This allows users to enter virtually any custom text they want, from a single value to hundreds of lines of text. Regardless of the value, the check box or radio button option visibly displayed is Other; however, the data stored and included in the field's index includes whatever your other data is. In contrast to adding values to a value list and changing the options available on all records, the Other function simply enables you to enter custom text into a specific record's field.

As you can guess, developers often disable this feature. Data can get buried behind another entry and can be difficult to account for. Just remember that all you're doing is using field formatting to help in entering consistent data. These fields are no different from standard fields that accept text data.

Field Types

In addition to enabling you to control how data is entered into a field, FileMaker Pro databases use specific field types for different types of information. Field types are independent from the field formatting discussed in the preceding section. For example, it's entirely possible to format a calculation field as a check box. Calculation fields are different from standard fields; they do not accept data entry and instead present the results of a formula. Although you, as a user, might expect to be able to click on a check box, if you do so, FileMaker Pro prompts you and explains that calculation fields are not modifiable.

It's incumbent on the developer to sensibly identify, for a given system's users, which fields expect what sort of data. Often field labels make this clear. For example, you can often expect a Price field to be a number, and an Invoice Date field will no doubt be a date type. You can also use cues (such as the icon for a pop-up calendar) to distinguish fields.

The following list describes the field types available in FileMaker:

- **Text**—The most common data type, text fields allow a user to enter approximately 2GB of information, including carriage returns. Sorting by a text field is alphabetical.
- **Number**—Number fields store up to 800 digits, 400 on either side of the decimal, and sort as typical numbers.
- **Date**—Dates are managed in FileMaker by the Gregorian calendar, 1/1/0001 through 12/31/4000. It's a good practice, but not required, to use four-digit years when doing data entry. Sorting is by year, month, and day, with the sequence of elements determined by the system settings in effect on your computer at the time the database file was created. This matters only if you commonly

tip

FileMaker Pro is adept at converting data from one type of field to another. If you have a calculation that requires a number, FileMaker Pro happily takes the contents of a text field and converts it to a number. Often this is exactly what you want, although the default conversion of "ten" to a number will not give you the result you might expect.

Despite the built-in conversions, you will get the best performance out of FileMaker Pro if you use the strictest representation of data. If data is a number, use a number field. Dates should be date fields, not free-format text fields.

deal with date sequences that vary (year/month/day, day/month/year, and so forth). If that is the case, consider using separate fields for month, day, and year to avoid ambiguity.

- **Time**—Time in FileMaker is stored in hours, minutes, and seconds, like so: HH:MM:SS. Sorting is based on a typical 24-hour clock.
- **Timestamp**—A timestamp is a tool generally used by database developers to identify exactly when a record was created or modified. It combines a date with a time and looks like "6/28/2008 2:00 AM." For the user, you might occasionally want to use a timestamp when performing a find.
- **Container**—Container fields hold just about any binary information, be it an image, a movie, a PDF document, a Word document, or a file archive. You cannot use these fields for sorting purposes. Container fields are capable of holding files of up to 4GB in size, making it possible to use FileMaker Pro for managing all sorts of digital assets.

  Data entry for container fields is slightly different from other types: You need to either paste a file or image into the field or use the Insert menu.

- **Calculation**—A calculation field stores the result of a formula, which might be based on other fields or related information in your system. The resultant data is assigned a type so that one can return a date, time, and so on. It's even possible for a calculation field to return container (binary) data.

  Calculations can also format data: From FileMaker Pro's standpoint, there is no difference between a calculation that adds two fields and one that rounds a number to two decimal places and then turns the result red. You will find out more about this in the chapters about layouts and about sharing FileMaker Pro databases.

note

The data in calculation fields is not modifiable by an end user; you can, however, access calculation fields for performing finds, sorts, and so on.

- **Summary**—Summary fields are similar to calculations, but they return information from your found set, or current group, of records. A summary field performing a Total operation, for example, totals a field across your current set of records. Other functions include averaging, totals, maximum, and minimum.

tip

Layouts have front-to-back ordering of elements. Mouse clicks are handled by the frontmost object underneath the mouse. If you place a field that is not editable on top of a field that is editable, you can often provide very intuitive and sophisticated interfaces without troubling the user.

For example, if you place an editable phone number field on a layout, you can create a calculation field of exactly the same size and shape on top of it and specify that the calculation field is not editable. Clicking the calculation field does nothing; the mouse click passes through to the next object that accepts mouse clicks at that spot—the editable phone number field. The user can enter an unformatted phone number; then, when another field is selected, the calculation is performed, adding punctuation to the phone number, and it appears that the phone number field itself has been formatted. Actually, the raw phone number field is simply obscured by the formatted phone number created by the calculation.

Saving and Retrieving Information in Container Fields

Container fields work differently than other fields. You cannot type data into them; rather, you have to insert whatever file or media you want to store or display in them.

Note that a container field can do more than just store documents. For many image types, it can display the image within FileMaker; for many sound types, it can play the sound within FileMaker; and for QuickTime movies, it can allow users to play the files. Whether you store something as a document or as a media type that FileMaker can play depends on how you save the information to the container field. There are three general ways to store a file or media in a container field:

- **Paste**—You can place an image or a document on your Clipboard and simply paste it into a container field. FileMaker makes its best guess as to what kind of information is on your Clipboard and either stores a document or displays an image, a sound, or a QuickTime movie.
- **Insert**—Using the Insert menu, you can choose from among Picture, QuickTime (movie), Sound, and File. If you choose from the first three, FileMaker displays the media in question. If you choose File, FileMaker loads a document.
- **Import**—Under the File menu is the Import Records menu item from which you can further choose to import a file or a folder. If you choose to import from a folder, you can point FileMaker to a directory of images or files and load them into a container field. You can also import container data directly from other FileMaker files.

In both the case of inserting a single media file and the case of importing many, you have the choice of inserting only a reference to the file or of inserting the document itself. In the case that you insert a document itself, that document is physically stored in FileMaker and is accessible by all users. They can select the container field in question and choose Export Field Contents from either the Edit menu or the contextual menu available from the field itself. In addition, as you see in Figure 2.22, you have the option to compress a file that you are inserting.

If you choose instead to store only a reference to the file, the file is stored physically elsewhere—for example, on a shared hard disk. To have access to the file, users must have access to the same shared directory on which the actual file sits. In this case, you are performing the same sort of task as saving a shortcut or an alias to the file: It remains on whatever storage device you found it.

The benefit of leaving documents on an external storage device and storing only references within FileMaker is that these documents require far less space within FileMaker for storage. Storing images and other files outside of the database itself is a common strategy used by efficient database management systems including FileMaker.

*See "Container," **p. 89**, to see how you can manage external storage and other settings for files stored in container fields. (This feature was implemented in FileMaker Pro 12.)*

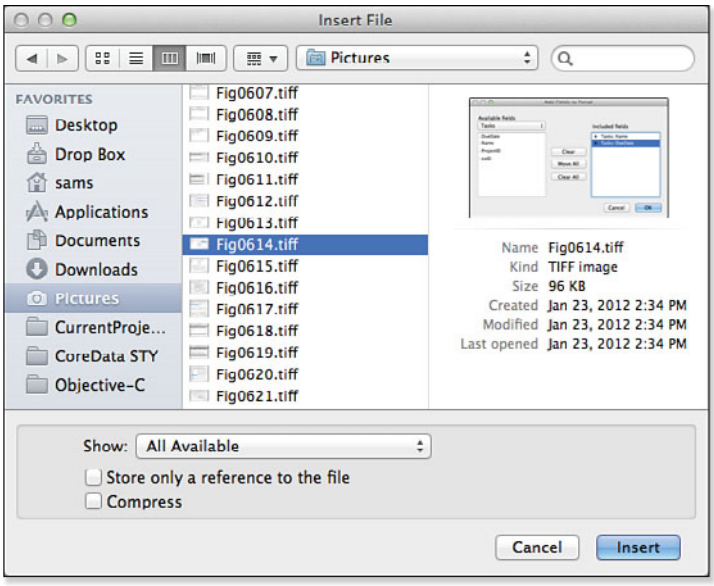


Figure 2.22
Insert files and (optionally) compress them.

Global Storage

Field data in your database generally pertains to a specific, individual record. The baseball team field for your San Francisco record holds the data "The Giants," whereas for Chicago, it's "The Cubs." In some cases, however, a developer opts to define a field as globally stored. Developers often use a shorthand, *globals*, to describe these sorts of fields. The value in that field is constant throughout the database, regardless of which record is currently under inspection. Some common examples might be fiscal year start and end dates, your company name, report headers, or a fixed commission rate. As a user, you might not always be able to tell which fields in your database are defined to store global values and which are record specific.

An important point to keep in mind about global fields is that their behavior varies depending on how you're hosting a database. If you're using a database on your own local machine, with sharing set to single user, all global data is preserved from session to session. In other words, the next time you open the database, your global details remain from the last time you worked with the system.

If you're working with a database hosted on a server, all global information is session specific. It might contain default values, but if you change some data in a global field, other users of the system do not see that change, nor is it preserved for the next time you use the database. If a developer has added global storage to a field in your system, it is quite likely that there are routines to manage what information it holds when necessary.

note

Before the introduction of script parameters and script variables, it was impossible to create certain types of scripts without the use of global fields. As a result, older FileMaker Pro databases use global fields more often than modern ones.

Data Validation

Data integrity is one of the primary concerns of any database developer or of the team using a given system. If duplicate records appear where they should not, misspellings and typos plague your database, or worse yet the wrong data is entered into the wrong fields, your system will soon become unreliable. For example, if you run a monthly income report, but in a few of your transaction records someone has entered a date value where in fact a transaction amount belongs, your monthly totals will be incorrect.

FileMaker Pro—or any application, for that matter—cannot read users' minds and fully safeguard against bad data, but developers do have a wide range of tools for validating information as it is entered. If your organization can come up with a business rule for validation, a developer can apply that rule to a given field or fields. Consider the following examples:

- Transaction amounts can be only positive numbers, can have only two decimal places, and cannot exceed 100,000.
- Employee hire dates may be only equal to or later than 1/1/2001.
- Data in a given field must match established values in a status value list containing the values `open`, `closed`, and `on hold`. The field will not accept any other status descriptions.
- Company names in the database must be unique.

Understanding that these rules are in place will help you understand the underpinnings of your database application. When a validation check occurs, the system might prompt you with an appropriate message (see Figure 2.23).

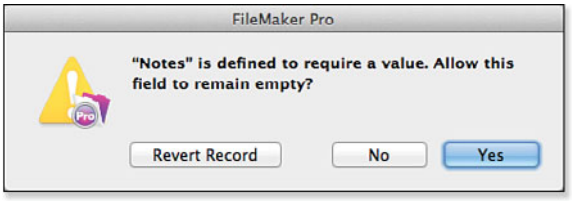


Figure 2.23
This is an example of a default validation message. If you choose Revert Record, whatever data you've entered into the field reverts to the state it had before you started editing.

In addition to the default dialog shown in Figure 2.23, developers can create their own custom text, as shown in Figure 2.24.

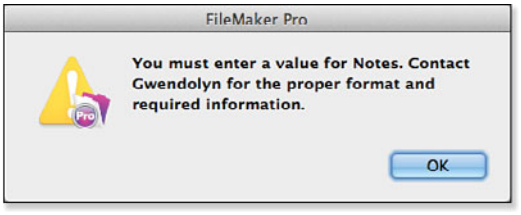


Figure 2.24
This is an example of a customized validation message.

If you choose Yes rather than Revert Record, your data is accepted as is and overrides the validation requirement. In some cases, you might not have the option of posting an override, and the Revert Record button will not be shown.

*To explore additional thoughts on addressing data problems, **see** "Data Integrity" in the "Troubleshooting" section at the end of this chapter.*

caution

Never, ever, under any circumstances, no matter what happens, use data to describe data *conditions*. Values of 0, 99, and –1 are *values*, not shortcuts for data conditions. You can use validation rules to prevent the storage of invalid data, and having done so, you can rely on the fact that the data is data. If you need to store imperfect data, consider using pairs of fields: as-entered data (which may be invalid) and validated data (which is always valid if it exists). FileMaker Pro correctly handles missing data (empty fields); anything other than valid data or missing data belongs elsewhere in the database.

Millions of dollars were spent at the end of the twentieth century tracking down what came to be known as the Y2K problem, much of which arose from the fact that programmers decades earlier assumed that it was safe to use 00 to indicate missing data for a year. The fact that 00 in the peculiar arithmetic of two-digit years was the result of 99 + 1 did not occur to many people until it was necessary to review every line of code in critical applications to see whether this had happened.

Working with Related Data

One of FileMaker's core strengths is how it allows you to view and work with related information from a different but connected contextual set of records from other tables.

The Tasks Starter Solution has related tables for assignees and attachments. A project can consist of one or more tasks; a task can be assigned to one or more assignees. And because these relationships have been set up in the database, it is easy for the FileMaker Pro layouts to display them and allow them to be edited.

Figure 2.25 shows the tasks table in the database. At the right of the layout is a *portal* that shows the people assigned to that task. A portal lets developers display related information to users.

You can have multiple relationships, and they can be displayed in multiple portals. Figure 2.26, for example, shows the Resource Scheduling Starter Solution. Tabs at the bottom display portals with relationships from each project to people, equipment, materials, locations, and other items.

note

Relationships are bidirectional. This means that, just as you can display the people for a project, you can turn it around and display the project for a person if the database is so configured. As you will see in Chapter 3, for the case in which the same people are assigned to several projects, there is a slightly more complex database structure (it is referred to as a *many-to-many* relationship.)

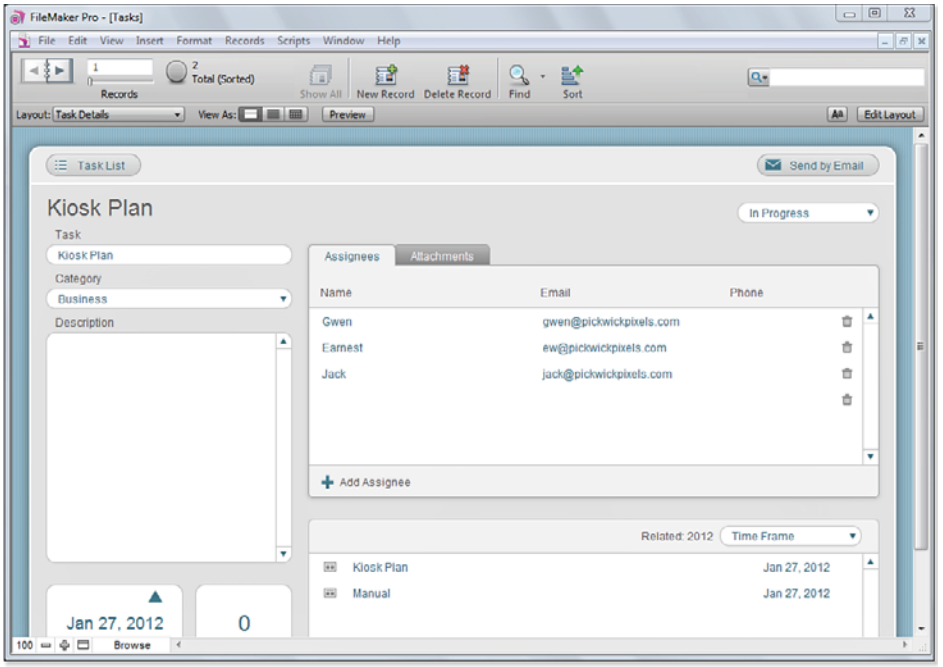


Figure 2.25 The Form View – Tasks layout contains a portal of related contacts.

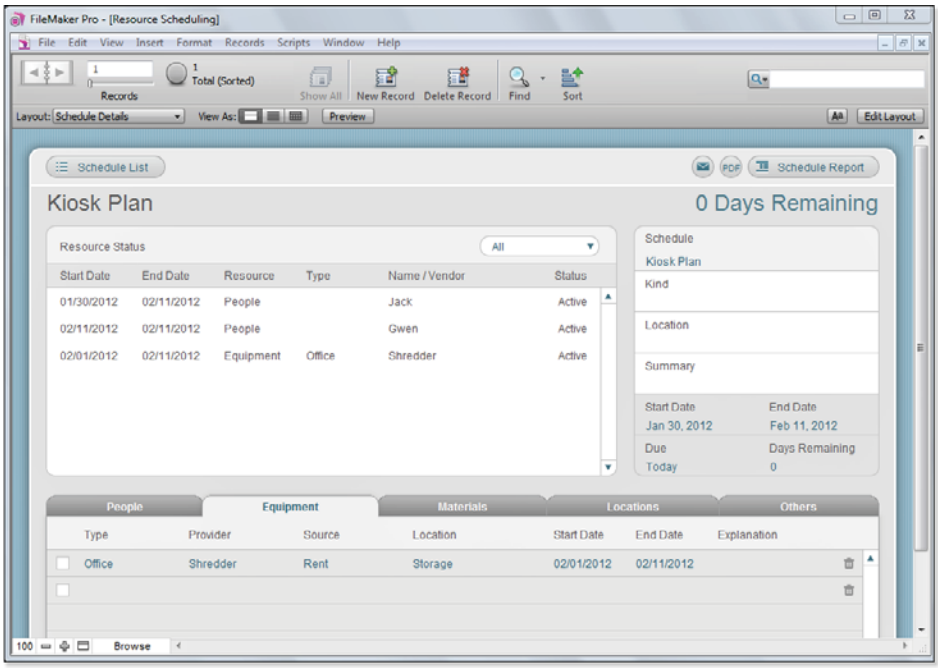


Figure 2.26 Tabs can display multiple relationships in multiple portals.

Understanding the Mechanics of a Portal

A *portal* is simply a view into another table and includes rows of specific related records. Developers determine the rules by which records appear in portals, and at times the records displayed can dynamically change or a portal might display other records in the same table you're currently viewing.

Most portals have a scroll bar on the right. They feel a bit like List views and act much the same way. To browse through your related records, simply scroll up and down through the list. Data entry works the same way it does in other areas of FileMaker: Simply click into a field and enter whatever data is appropriate.

At times, developers include buttons in portals. When they place a button within a portal, the button in question appears on each portal row, and each row's button will act on that row's data or record. Common buttons are a delete button (often a red X, a trash can, or a red circle with a slash through it at the right of the portal row) to delete that portal row and a disclosure button (often a triangle at the left of the row) or a checkbox at the left of a portal row. Figure 2.26 shows both of these interface elements.

Creating and Deleting Portal Rows

To create a new portal row—which then creates a new child record—scroll to the first empty row of a portal. If there are blank fields, click any of the blank fields there. *Child records* is a term often used to describe related, hierarchically dependent records—for example, Company and Employee. Employee records are considered children of Company.

If a developer allows for it, you can delete a portal row by following these steps:

1. Click outside the fields of a given portal on the row background. (You might have to mouse around a bit.) You should see the row become highlighted.
2. Press the Backspace or Delete button on your keyboard. You can also use Records, Delete Record or the Delete Record command on the Status toolbar. You are prompted as to whether you want to delete that one related record. Click Delete or Cancel to close the dialog box.

note

Your developer might have turned off the ability to add or delete portal rows, in which case there should be an alternative means of adding related records, such as a + and perhaps explanatory text so that you know what you'll be adding. Likewise, your developer might have disabled the ability to create new records using the first available row. In addition, as you will see in the discussion of related records in Chapter 7, not all relationships are simple. In the case of complex relationships (nonequijoins, to be specific), FileMaker Pro cannot allow this method of adding rows to portals because the relationship would be ambiguous. For several of these reasons, portals increasingly allow the creation of new portal rows with a button outside the portal.

Portal Sorting

Sorting records is covered later in the chapter. For now, simply note that a developer determines by what means a portal sorts and that there is no way for you, as a user, to change a portal's sort order unless the developer creates a specific mechanism allowing for that option. A developer can build a dynamically sortable, command-driven portal in various ways, but this is not the default behavior in FileMaker Pro.

note

It's important to remember that the developer of a given file must have turned on this portal behavior. It is increasingly considered good practice to explicitly provide a delete row icon in a portal row, rather than asking users to understand the mechanics of deleting portal rows.

Finding Data with FileMaker

Up to this point, we've discussed working with a single record and the fields on a given form layout, but at all times FileMaker holds a found set of records—anywhere from none to all the records in the table that is the basis of the current layout.

This is an important point to remember: Even though you might be able to see the contents of only one record's fields (more than likely in Form view), you can still work with either all the records in your table or a subset of such. Think of it as working with a deck of cards. There are 52 total cards in your deck, some of which are in your hand, and one of which is frontmost (visible). Your current record would be akin to that front card and your found set like those cards in your hand. In FileMaker, many functions apply to a found set. A good example is sorting: You are ordering only those records in your found set.

Many FileMaker Pro databases offer layouts tailored to be viewed either in Form view, where one record encompasses the information on the screen, or in List view, where layouts display multiple records at once. The figures in this chapter have shown both styles. Today, with the knowledge that FileMaker databases may be used on a wide variety of devices, many people are adhering rather strictly to this list/form structure. It is a familiar interface paradigm on the Web, and it is easily adaptable to desktop and mobile devices.

Figure 2.27 shows the Assets Starter Solution in Form view on an iPad running FileMaker Go; Figure 2.28 shows the corresponding List view.

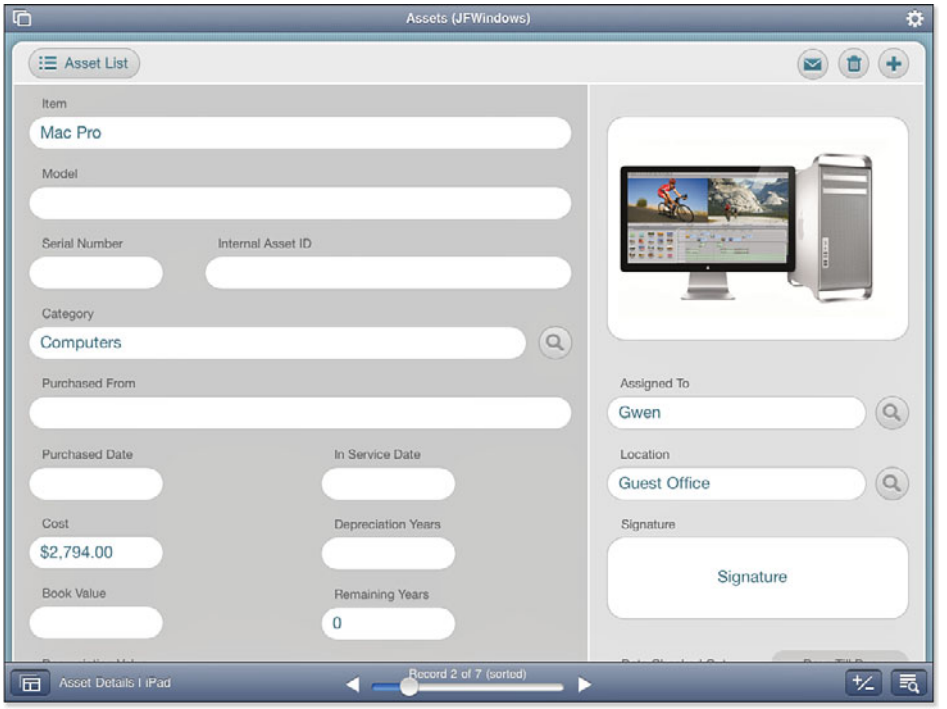


Figure 2.27 Assets Starter Solution in Form view on an iPad running FileMaker Go.

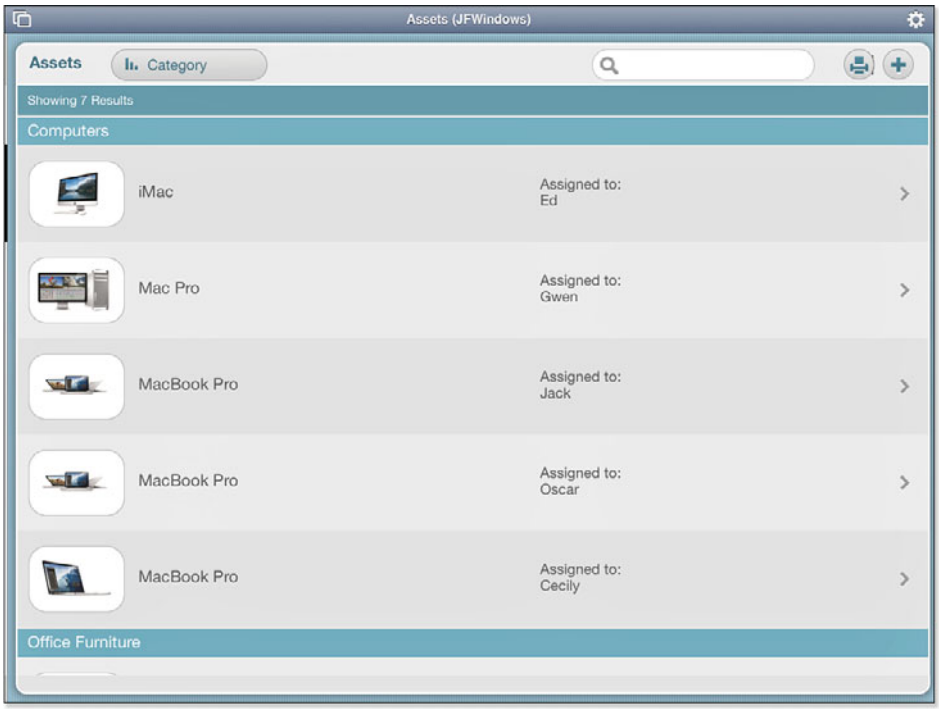


Figure 2.28
Assets Starter Solution in List view on an iPad running FileMaker Go.

Working with groups of records is important mainly for comprehension and processing of your information. Data entry occurs on one individual record at a time, unless you're importing or performing some other function that applies across multiple records. It's in the reporting and analyzing stage that working with multiple records becomes necessary.

One of the first ways to work with a group of records is simply to scan the list visually. Summary fields might lie at the bottom of a List view and can total numeric data based on a current found set, or perform other summary operations such as counting or averaging.

For a quick example of how this might work, imagine a sales database. If you were to find or search for all records in January, your summary fields could total January's sales. If you were to find again for the year 2013, your totals would be annual. The value of the summary field varies depending on your found set. If you perform different find requests, the information on your screen can deliver different results, specific to a given group of records.

note

Summary fields are quite powerful, but they require processor time. If you have a large found set of thousands of records, waiting for a summary field to evaluate can take some time. You can press the Esc key (or Command in OS X) to cancel the summary, or simply avoid scrolling or viewing that portion of a layout. Summary fields evaluate only when they are visible on the screen.

One last important note about found sets: They can be composed of records from only one table. You cannot, for example, display records from an automobile table and a manufacturer table in the same List view or Table view, although they can be shown as related fields to the main table's records.

Using Quick Find

At the right of the Status toolbar in Browse mode, a search field lets you perform a Quick Find. Type the word or phrase you're searching for, and FileMaker searches for it in the current layout's table. FileMaker doesn't care what field the data is in: it will find "York" equally well in a field for city name, a field for last name, and a field for plays in Cricket. You type it, FileMaker Finds it.

Using Find Mode to Perform a Find Request

You can also use Find mode to search for data in specific fields. This lets you work with found sets of records.

To create or change your found set in FileMaker, you must perform a find request or search. This entails getting into Find mode and entering some set of search criteria into the field (or fields) by which you want to search. FileMaker takes you back into Browse mode after your search is complete.

To perform find requests in FileMaker, you have to use one of three options to change to Find mode: the Find icon at the right of the Status toolbar, the menu on the bottom left of your application window, or the View menu. Developers might also opt to put various Find buttons into their systems. When you have entered Find mode in any of these ways, the Status toolbar changes, as shown in Figure 2.29.

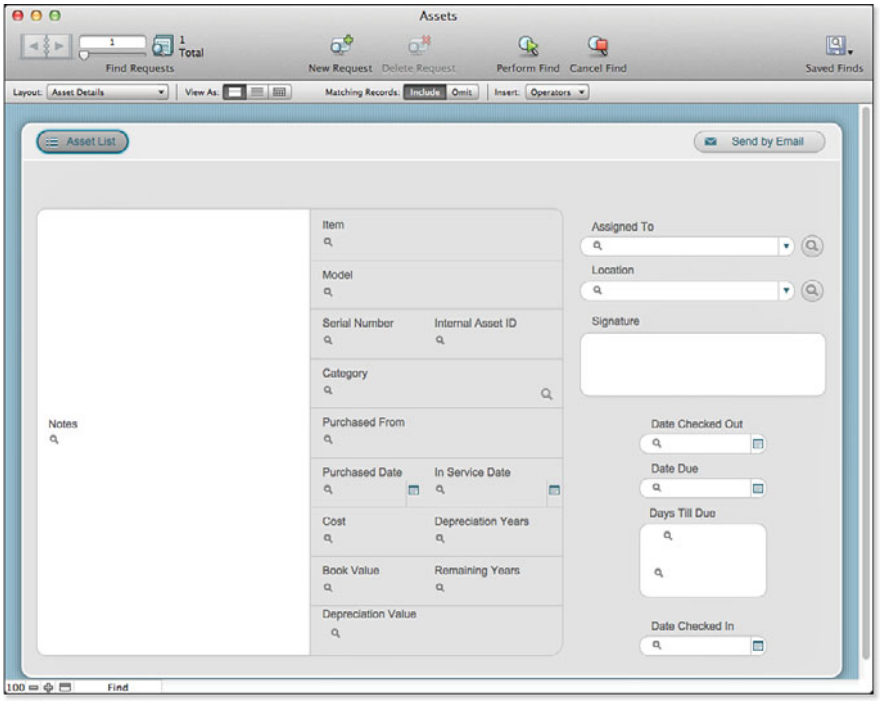


Figure 2.29
The Status toolbar changes when you click the Find icon.

At the left of the Status toolbar, the Records section that is visible in Browse mode changes to a Find Requests section. Instead of paging through records, you can now page through find requests. Similarly, instead of seeing the number of records, you now see the number of find requests.

To the right, icons let you create new find requests and delete old ones; others let you perform a find or cancel it to return to Browse mode. At the bottom right of the Status toolbar, buttons let you indicate that a find request should return records that match the request or that it should omit records that match the request. Logical operators for the find requests are also available. All of these are described in this section.

After you're in Find mode, you create a find request that describes the criteria by which you want to search. You enter data into fields just as you would in Browse mode, but instead of creating or modifying records, these requests serve as instructions for finding your data. You can add a new request, create multiple requests, and delete requests from the Status toolbar or from the Requests menu that replaces the Records menu in Find mode.

FileMaker Pro shows a small magnifying glass in data-entry fields when in Find mode, as you see in Figure 2.29. As soon as you click in the field, the magnifying glass disappears and you can type your find criterion. This prevents users from accidentally entering data in Browse mode when they think they are in Find mode. For example, you can type iMac into the Item field. Figure 2.30 shows the results of the find. Notice that you can tell from the Status toolbar that you're automatically back in Browse mode. At the left of the Status toolbar, you can see that one out of seven records has been found (that is the *found set*).

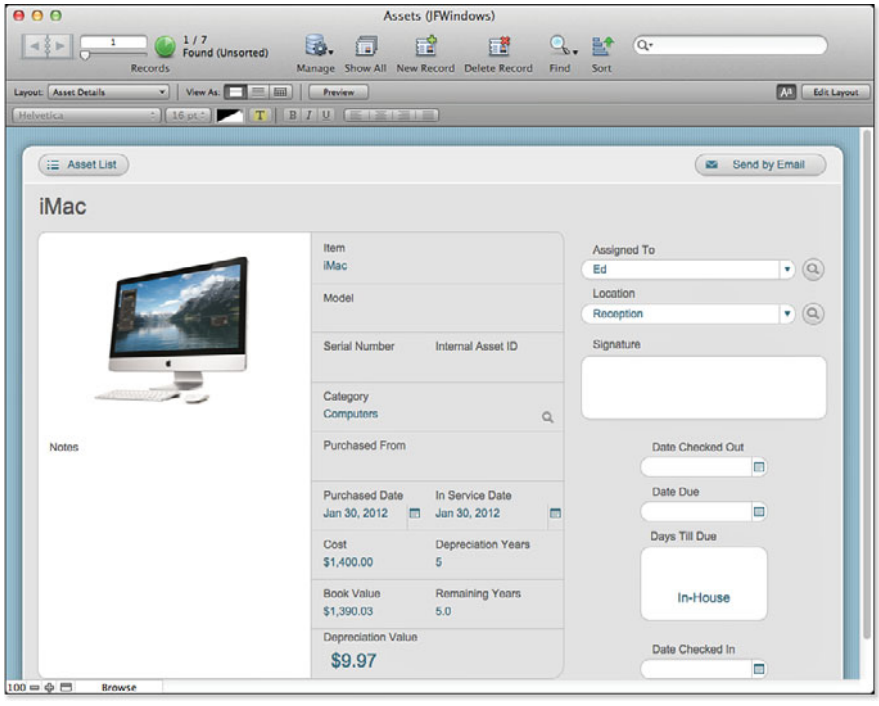


Figure 2.30
FileMaker matches the find criterion "iMac" in the Item field against the data in the database after you click Find to find all assets with "iMac" in the field item.

If you perform another find, the records matching your new request replace your found set.

You can perform a complex find by entering data into more than one field. FileMaker Pro finds all the records with both of those find criteria. In database parlance, this is called an *and* query. Figure 2.31 shows a request with two criteria: MacBook is entered in the Item Name field, and oscar is entered in the Assigned To field in the portal. Both of these conditions must be fulfilled for a record to be found.

tip

Note that by default, FileMaker doesn't require strict matches (MacBook, for example, matches MacBook Pro). Also, by default, capitalization doesn't matter. You control this behavior with search operators, as described in the following section.

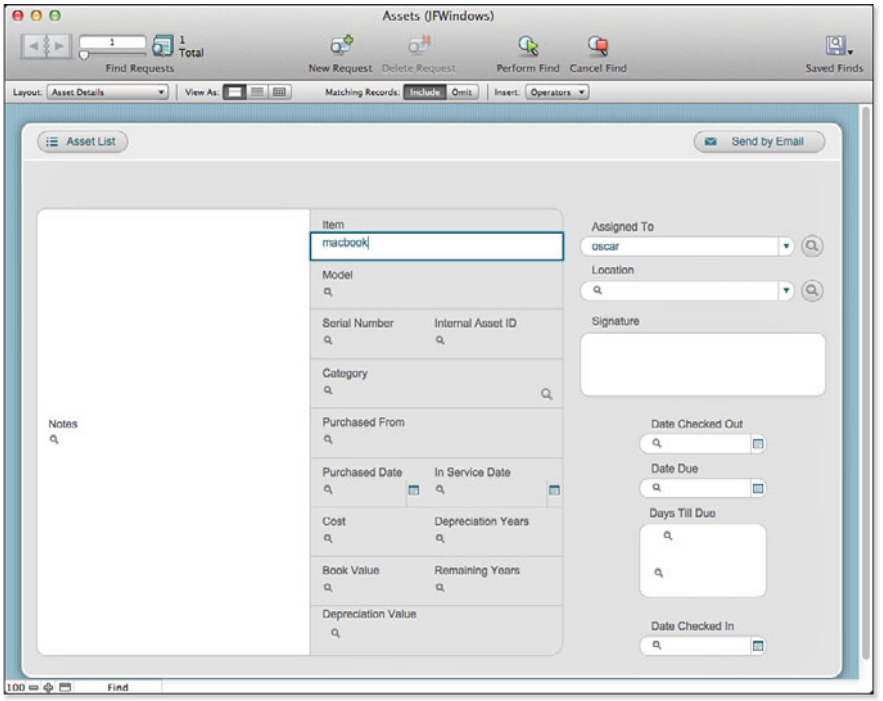


Figure 2.31
A find request with multiple fields specified retrieves data in which all the criteria are met—an *and* query.

Search Operators

In addition to text that you type in, you can use operators to construct queries. Figure 2.32 shows the Operators menu in Find mode.

The less than, less than or equal, greater than, greater than or equal, and exact match symbols should be obvious. An entry of `>3` finds all records with a value 4 and above. An entry of `<=100` finds all records with a value of 100 or lower (including zero and negative numbers).

note

You need not use the Operators menu at all: < and = typed from your keyboard work just as well as inserting the composite symbol from the pop-up menu in the Status toolbar.

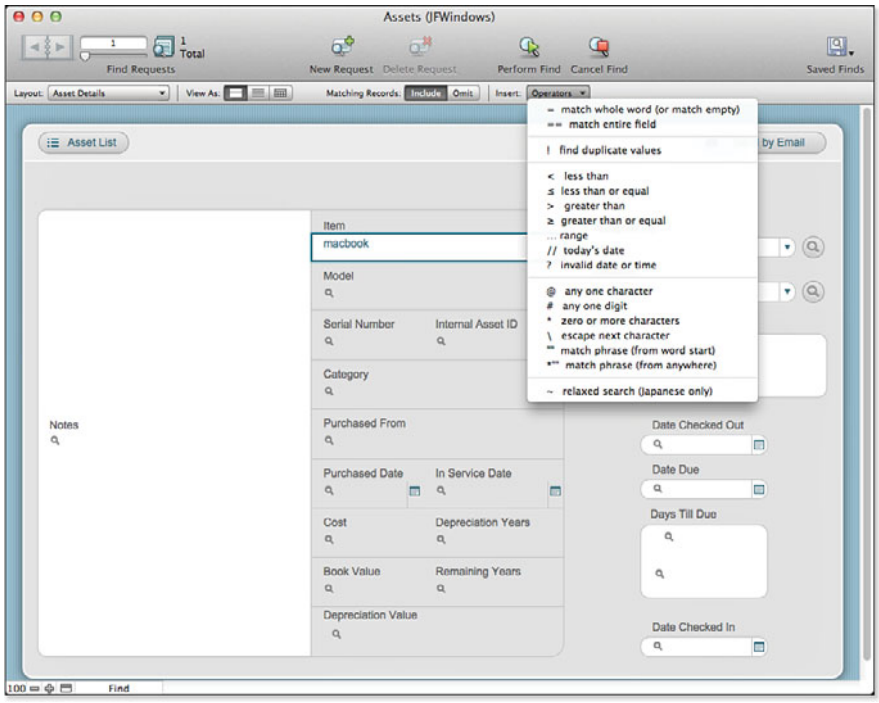


Figure 2.32
Special symbols enable you to search for a wide range of match criteria.

The ellipsis (`...`) for ranges is a commonly used search symbol. The search criterion `1/1/2003...12/31/2006` returns all records for the span of four years. (Two or three periods from your keyboard work just as well.)

Use * and # for wildcards. The # symbol is for one digit exactly. An entry of `5#` finds all whole numbers from 50 to 59. The # alone finds just numbers 1–9. A `1#1` criterion finds 101, 121, 131, and so on, but not 211 or 1211.

The ~ for relaxed search looks intriguing, doesn't it? Some fuzzy logic, perhaps? No such luck. It's used to search for common base characters in two-byte Asian phonetic alphabets. It doesn't do anything for any other languages.

Shortcuts for Fast Finding

The right-click (Control-click for Macs using a one-button mouse) contextual menu for a field in FileMaker will show three "fast match" commands: Find Matching Records, Constrain Found Set, and Extend Found Set. Here, FileMaker performs a find request on the data in the field in question. If, say, you right-click a field containing the term *porch* and choose Find Matching Records, your found set changes to show all porch records. Likewise, you can constrain and extend your found set based on the value in the field. We'll cover these concepts in the next section.

FileMaker also has some date-searching capabilities that save time. You can type `2005` in a date field and FileMaker correctly interprets that to be a `*/*/2005` search that results in all the records for a given year. Likewise, you can enter `1` through `12` in a date field and FileMaker assumes that you're searching for records within that month for a given year.

Finally, you can search for the names of the days in a date or timestamp to pull up records specific to the days of the week.

Multiple Find Requests

FileMaker Pro also enables you to perform complex searches involving multiple find requests. To find both the MacBook and the Oscar records, a user would simply enter Find mode, type `MacBook` into the item field, and then create a new record/request. Just as you can create new records in Browse mode, you can create and delete requests in Find mode. This process is identical to creating a new record in Browse mode. In the second record, a user would enter `Oscar` in the assigned to field.

A user can flip between requests, using the book icon in the Status toolbar, and can delete requests as necessary. As soon as the user is satisfied with a series of requests, clicking Find on the left performs the find and returns the user to Browse mode with a new found set.

Multiple find requests can also include requests meant to be omitted. Use the Include and Omit buttons in the Matching Records section of the Status toolbar to do this. Thus, you could find all painting tasks for the house—or by omitting *house* from painting tasks, you could find those that refer to the garage, garden shed, and other non-house areas. Whereas typing find criteria into multiple fields in a single request produces an *and* query (all the criteria must be true), using multiple requests creates an *or* query—all the criteria in the first request must be true or the criteria in the second request must be true, and so forth.

Figure 2.31 shows a typical *and* find request: It searches for records that contain an item named "MacBook" and an assigned-to named "Oscar." If you create two separate requests, one of which searches for the item and the other of which searches for the Assigned To name, you will have an *or* request. Because they are separate requests, they find all records where the item name contains *MacBook* or the Assigned To name contains *Jack*. In general, *or* queries return as much or more data than *and* queries because they are less restrictive.

tip

If you are analyzing data in a database with FileMaker Pro, it is important to be able to categorize the data properly. For example, imagine that, in a school enrollment database, you find 15 students enrolled in Nineteenth Century History and 25 students enrolled in Physics—two facts you could determine with simple find requests. It might be more useful to know that of 15 students enrolled in Nineteenth Century History, 12 are also enrolled in Physics—something you could determine by constraining the Nineteenth Century History found set.

Constrain and Extend Requests

Performing find requests is all well and good, and as you can imagine, they can become quite complex. Instead of developing complex find requests, you can work through the complexity in stages. Rather than clicking the Find button in the Status toolbar, choose Requests, Constrain Found Set. FileMaker Pro performs this new find request on only the existing found set rather than on the entire database.

Using Requests, Extend Found Set works in a similar fashion by retaining the existing records and simply adding more to them. This way of working helps you simplify complex queries. It also is a powerful way for you to explore and analyze the data in your database.

Modify Last Find

Modify Last Find is a great feature for find requests. In Browse mode, choose Records, Modify Last Find. You are placed in Find mode with the last set of find requests you performed. This capability is handy if you want to continue to play with a particularly complex set of find requests or are simply performing a series of similar requests.

Finding on Multiple Layouts

FileMaker's find functionality is flexible. While you are in Find mode, it is entirely possible to change layouts. As long as all the layouts on which you enter your requests are associated with the same source table, your find performs just as though you had a layout with all the fields on it you needed. Finding is not layout specific.

Finding is, however, always table specific. Some more advanced FileMaker Pro solutions comprise multiple tables. Although it is possible to search across related information in FileMaker Pro, your find results will always display a found set of records from a single table.

*To learn more about working with multiple tables, **see** Chapter 6, "Working with Multiple Tables."*

Omitting and Showing All Records

After performing a find, you can opt to omit individual records from the resultant found set. Choose Records, Omit Record to omit a single record or Omit Multiple to omit a specified number of records. To restore your found set to the full set of records in your current table, choose Records, Show All Records.

With the Status toolbar, use the button shown in Figure 2.33 to switch between the found and omitted records. Choosing whether to find or omit records while you are constructing find requests in Find mode is done with the Matching Records choice; after you have performed the find, use this icon in Browse mode to switch between the found and omitted records. For example, you can find all records assigned to Oscar and omit those that have not been checked out (in other words, they might have been assigned to Oscar but he has returned them).

Saving Find Requests

You can save find requests for future use. This is particularly helpful with complex find requests. The Save Finds icon at the right of the Status toolbar brings up the menu shown in Figure 2.34.

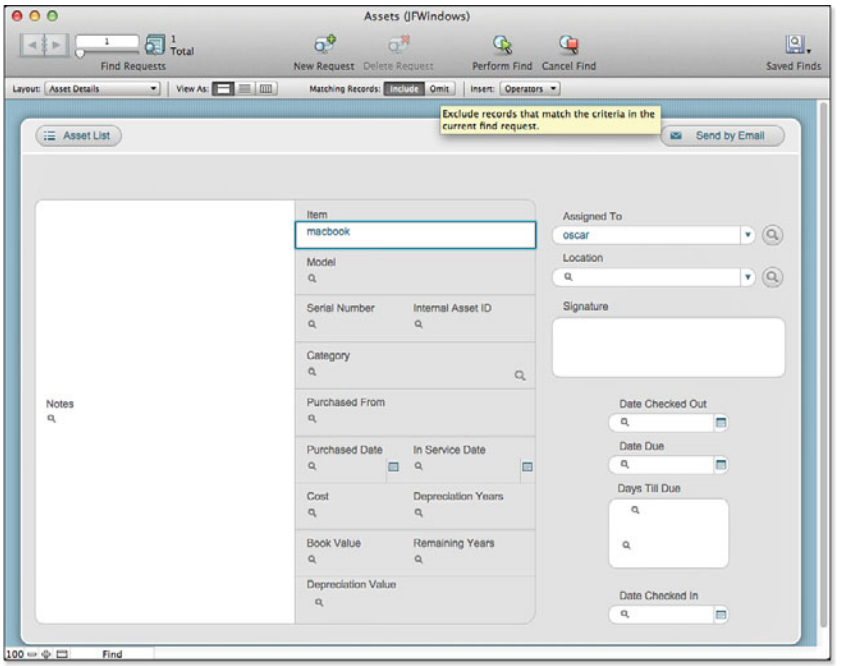


Figure 2.33
Switch between found and omitted records.

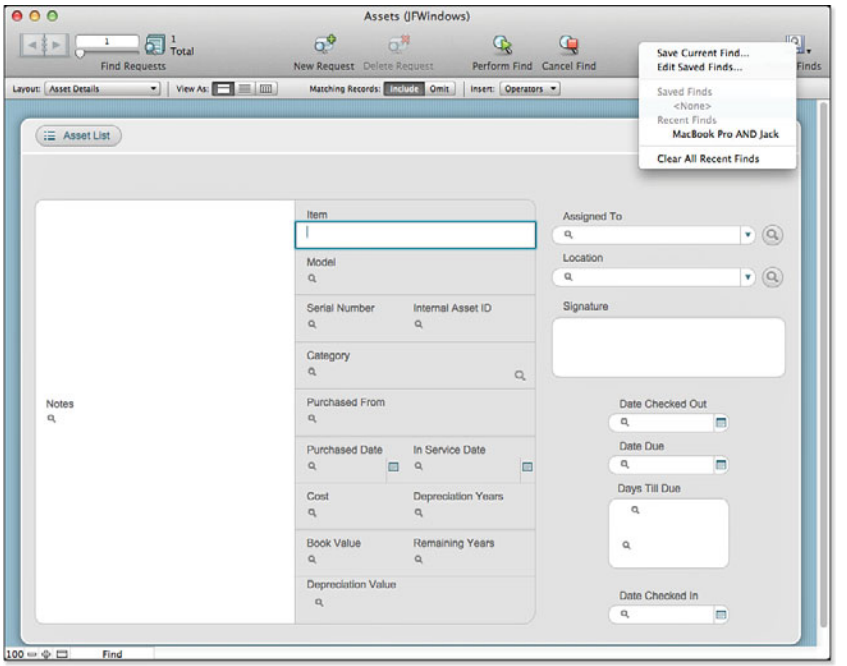
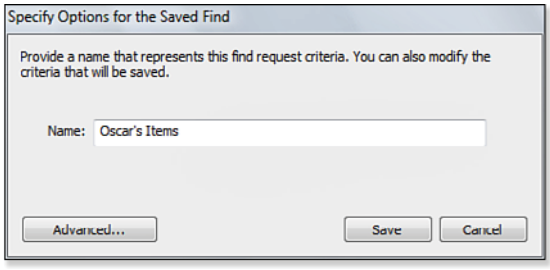


Figure 2.34
You can save finds in FileMaker Pro.

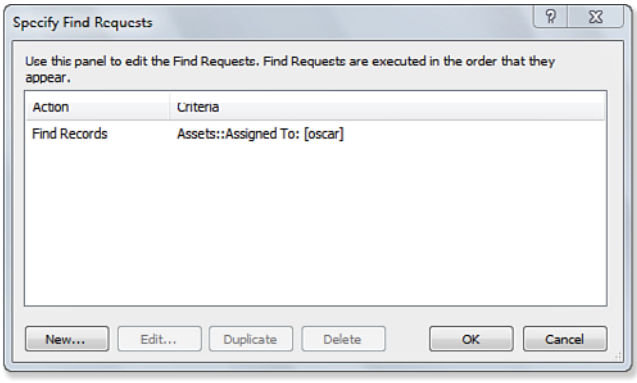
If you choose to save a find, you are prompted to name it, as shown in Figure 2.35.

Figure 2.35
Save finds by name.



The Advanced button in the lower left lets you open the details of the saved find. At first, all you see is the find command you have just performed, as shown in Figure 2.36.

Figure 2.36
See the find you have just performed.



You can edit the find command, as shown in Figure 2.37, to customize it before you save it.

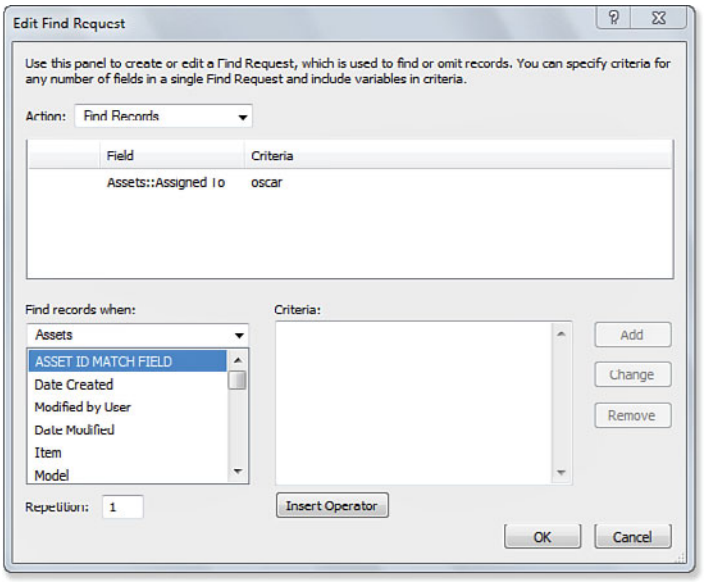


Figure 2.37
You can modify the find before you save it.

Sorting

When you're working with multiple records, an obvious requirement is the capability to sort them so that they are arranged in a logical order.

note

Before you go further, a few words of caution are in order. Sorting is perhaps the most overused feature in databases such as FileMaker Pro. Sorting is expensive in terms of computer resources and, particularly in the case of shared databases, can slow down performance for everyone. Many people think nothing of sorting a large database so that the records are in alphabetical order (last name first, first name last, or whatever). They then scroll through the list view to find the record they want and proceed to edit the record using a form layout.

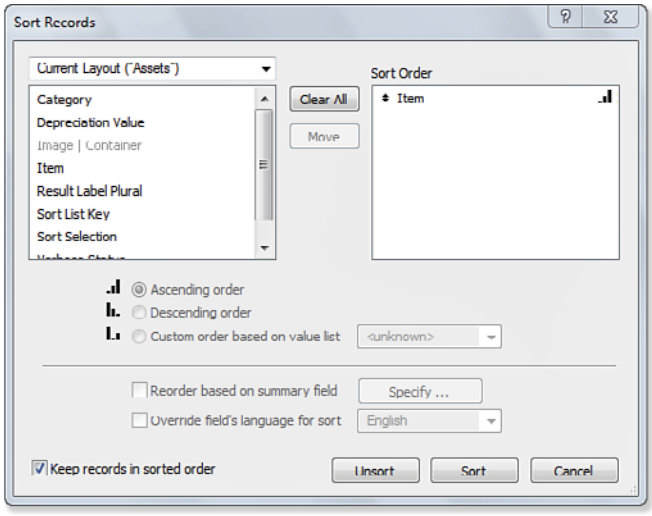
In almost every case, it is much, much faster to eschew the sort and simply use a find request to find the record you wanted. Doing so takes fewer computer resources, and you do not have to spend your time scrolling through the alphabetized list. Sorting is essential for printed reports, but for routine data manipulation, finding is often what you should be doing.

If you do need to sort data, it can be very useful to precede the sort with a find. Find the general set of records in which you are interested and then sort it. Doing so substantially lessens the demands on the computer.

FileMaker doesn't store its records in a sorted order; it stores them in the order in which they were created. When you first open an unsorted table, the records follow that order. There aren't any real mysteries here; for a view of the Sort Records dialog, see Figure 2.38. By default, the Sort Records dialog shows only those fields available on your current layout, but you can use the menu in the

upper left of the dialog to choose from among all the fields in your database (including those related to the records in your found set).

Figure 2.38
You can control how a field is sorted: ascending by type (alpha or numeric generally), descending, or in custom order by value list.



To sort the records from a table in your database, move fields from the left side of the dialog into the right. There, you can choose to have a field sort ascending, descending, or based on the order in which values appear in a specific value list. Choosing Descending, for example, sorts a number field from largest to smallest.

If you move multiple fields into the dialog, FileMaker sorts all records by the first field. In cases where records contain the same values in the first field, FileMaker then uses the second field as an additional criterion.

By adding multiple fields to your sort criteria, you are specifying secondary sorts: First sort by last name and then by first name, for example.

Often, sorts are attached to buttons so that you don't have to enter the sort specifiers each time. For example, in the Assets list view shown in Figure 2.39, you can sort assets by item, category, or status.

tip
Sorting by value list enables you to set up your own order in which things should appear. For example, if you have a workflow process that flows from Pending to Approved to Complete, you can have your records sort in that order rather than alphabetically.

Sorting by summary field is a bit tricky. ***See*** *"Using Summarized Reports,"* ***p. 295****.*

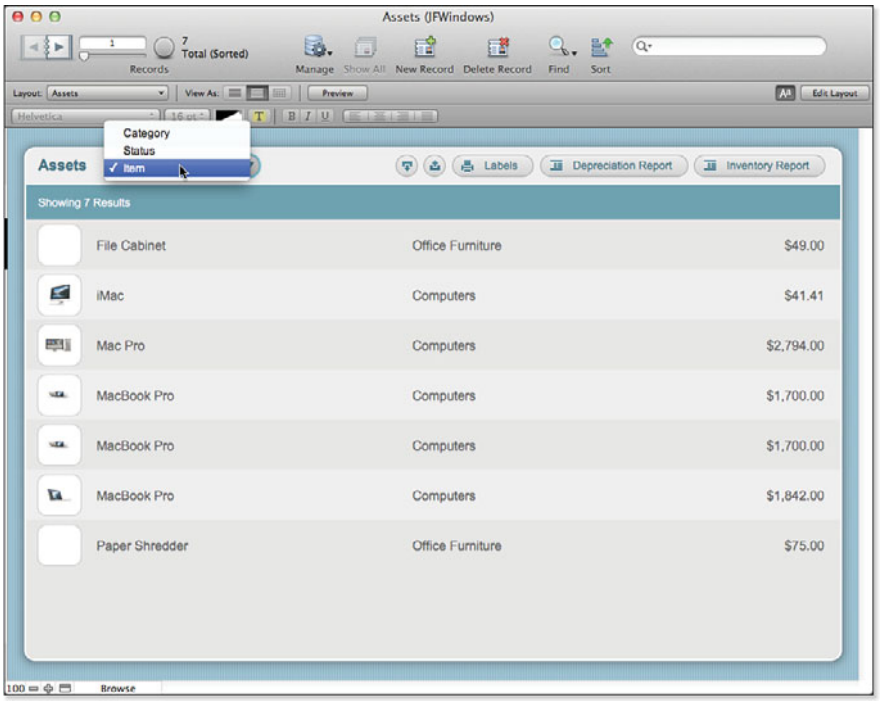


Figure 2.39
Attach sort scripts to interface elements.

Printing

Printing is straightforward in FileMaker. Choose File, Print. In the subsequent dialog that appears, you have the choice to print your found set, just the current record, or a blank record showing field names.

If you'd like to see what something will look like before wasting paper on something you don't want, use Preview mode via the Preview button in the center of the Status toolbar, or the View menu. Choose the layout from which you want to print and change to Preview mode.

After you're there, you can see where page margins will fall, and the Book icon enables you to step through the pages you will send to the printer. Keep in mind that Preview mode shows you what will be sent to the printer if you choose to print current records.

Presenting Data with Summary and Subsummary Reports

One prevalent type of report is a summary or subsummary report. A *subsummary report* enables you to group records that share some bit of common data.

You should design reports for the most restrictive printer on which they will be produced, which means adjusted paper sizes, color, and the like so that they will always look correct.

In Figure 2.40, you see the report that the Assets Starter Solution actually provides. It contains *sub-summary* parts that organize the data by category. Totals can be provided for subsummary parts if the report designer specifies, and, as you can see, FileMaker can generate a chart for the data. The report is accessed from a script attached to the button Inventory Report. It is very common to provide reports that are generated by a script that selects data (perhaps with a find) and then sorts it in preparation for the report. The script then pauses, and a Exit Preview button is displayed in the Status toolbar. Very few commands are available while the script is paused, but printing is one of them. If the report looks right, you can print it and then click Exit Preview to go on with your work. The script normally returns you to another layout.

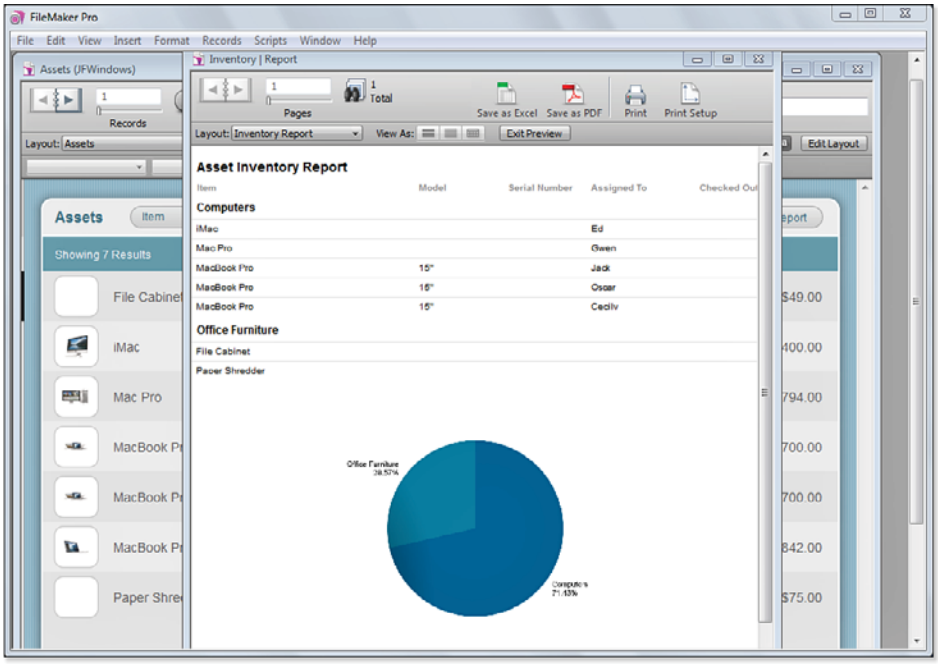


Figure 2.40
A summarized report can be more comprehensible than detailed data.

tip

Developing summarized reports very early in your database design process can be useful. Although you might want a totally interactive database, many people are used to seeing information on paper. Also, as you design a report that is sorted and contains subsummaries, the structure of the report sometimes makes omissions of categories and data fields obvious.

FileMaker Pro has the capability to display subsummary and summary data in Browse mode while you are viewing data in Table or List view.

More information on how to display subsummary and summary data is included in Chapter 4, "Working with Layouts."

Importing and Exporting Data

Having to manually type every bit of data into a database can be an excruciating experience. Fortunately, FileMaker has excellent capabilities for importing data from a wide variety of sources. Later chapters cover integration with other systems. For now, keep in mind that there are options other than spending all day at the keyboard.

Importing data moves data from one environment to another in a batch mode, and that is what the following sections summarize. FileMaker Pro now also provides a number of ways to share data in real time. You can use ODBC to access external SQL data sources in real time. You can also add SQL tables to your database graph, and you can even add supplemental fields to the external tables—your FileMaker database stores the supplemental fields.

*To explore how to bring data, including a directory of images, into your FileMaker Pro solution, **see** Chapter 22, "Importing Data into FileMaker Pro."*

*To learn about ODBC connectivity and exporting, **see** Chapter 23, "Exporting Data from FileMaker."*

Saving PDF and Excel Documents

Often you will want to prepare a report in FileMaker and create an electronic document you can then share with your colleagues. From the File menu, you can now save directly to PDF or Excel without having to use any third-party software.

Notice that you can automatically attach your documents to an email or even use FileMaker to compose an email message from data within your database. This function depends on your having an email client installed on your computer or having access to an SMTP server.

*For more information on saving PDF documents and sending email from FileMaker, **see** "Delivering Reports," **p. 309**.*

Using the Web Viewer

No overview of FileMaker's features would be complete without a discussion of the Web Viewer. First introduced in FileMaker Pro 8.5, the Web Viewer provides a way to display data from the Web on a FileMaker Pro layout. As long as an Internet connection exists, the data can be live, and the user can interact with it much as with a browser.

The Web Viewer does not replace a browser, but it does provide a very significant set of features to users of FileMaker Pro. Because the address used to set up the Web Viewer is a calculation, you can display a Web Viewer that presents a map of a customer location from your database. You can display a client web page containing contact information so that you do not have to continually update your Contact Management database with new telephone extensions and mail stops.

*For more information on the Web Viewer, **see** Chapter 13, "Using the Web Viewer."*

Troubleshooting

Most of the trouble you'll run into as a user will be with issues specific to your own database solutions. The best advice we can offer both developers and users is to work together! When you run into problems, knowing your developer will be a great first step.

Data Loss

I've noticed that I'm suddenly missing some data. What happened? What can I do?

One of the most critical aspects of your database is what gives it its name: your data. A wide range of possible problems can affect your data, but the most dangerous is accidentally deleting a record... or worse yet, discovering that you had the wrong found set when performing a Delete All Records command. FileMaker doesn't have an undo function, so if you delete a record, it's gone forever. Be sure that you haven't simply altered your found set to exclude the records you're looking for. Go to Records, Show All Records to recall all the data in your table.

Back up your data. We can't stress this point enough. FileMaker Server deployment best practices and backup routines are easy to learn. If you're not using FileMaker Server, you can make time-stamped copies of your files and store them on CD or on another computer. By far the best backup strategy is to use an automated procedure. That way, no one has to remember to do anything. It always seems to be the case that the only time you do not back up your data is the one time your hard disk fails.

Also, make certain you have practiced recovering data (backing it up and then restoring it to disk). It is common for someone to note a data loss and proceed to take all sorts of steps to recover the data (inadvertently destroying data in the process). And then, at the end of the day, it turns out that there was no data loss—all the day's data was entered with an incorrect date.

Data Integrity

How do I ensure that the data I have in my database is "good" data?

Making sure that good data is entered into your database is vital. If you properly put people's names in the first name and last name fields of a contact database, but your office assistant decides to enter nicknames and other random tidbits, your data will be compromised.

In addition, duplicate data is a problem that plagues all databases everywhere. If you've already created a record for, say, Uryas Forge, you won't want to create a second record for him. What happens if his phone number changes? You'll change one record, but not the other.

Dealing with bad data is a challenge and almost always requires the power of the human brain. Become adept at running find requests. Use the ! mark to find duplicates and use * characters for wildcard characters.

You can also work with your developer to put validation in place, or even build an approval process by which new data is added to your system in phases, with raw data in one set of fields and confirmed data in another. Alternatively, you can add a single check box field to each record that indicates whether the data has been reviewed.

Reverting Records

What does Revert Record do?

As you enter data into fields, that information is not saved (committed) until you exit the record in question. You do so by clicking outside any fields or by changing modes, changing layouts, and so on. Before the record is committed, you can choose Records, Revert Record. This command undoes all the data you've entered while working with active fields. If you've tabbed from field to field, it reverts all those not yet saved. If you have created a new record, it even reverts the entire new record if you haven't yet committed it.

FileMaker Extra: Becoming a FileMaker Pro Power User

Manipulating data can illuminate a wide range of information and can allow business users to draw conclusions they might not have been able to perceive anecdotally. For example, in our consulting firm, we were able to analyze our time entry data and calculate the average amount of time we need for testing. This helped greatly for future estimating.

Becoming adept at using FileMaker Pro enables you to understand what information you can pull from the system, but, most important, it enables you to know what to ask for. In working with a developer, you can guide that person's priorities (or your own) based on a solid understanding of the platform.

Technique 1: Using Your Keyboard for More Speed

This advice is obvious. Entering (Command-F) [Ctrl+F] brings you into Find mode. Tabbing takes you from field to field. The (Return) [Enter] key executes default values in dialog boxes, performs finds, and so on. (Command-up arrow) [Ctrl+up arrow] and (Command-down arrow) [Ctrl+down arrow] page through your data. You'll become much faster with FileMaker Pro if you take the time to learn your key commands. FileMaker's online help details all the key commands available.

Technique 2: Working with Table View

User interfaces have their purpose and more often than not greatly assist data entry and working with a given solution. But if you just need to look at the raw data in your system, you can opt to change to Table view from any layout in FileMaker Pro, assuming that your developer hasn't disabled the option. This gives you a bird's-eye view of your information. Don't forget that clicking a column header sorts for that column. A second click re-sorts in descending order.

Technique 3: Replacing Data

You'll often run across cases in which you need to globally replace some data with other data. For example, perhaps you've changed a value list of vehicle types to read "auto, bike, boat, plane," rather than "bike, boat, car, plane." If you leave things alone after changing the value list, you'll have both "car" and "auto" data in your system. Enforcing the consistent use of terms is important

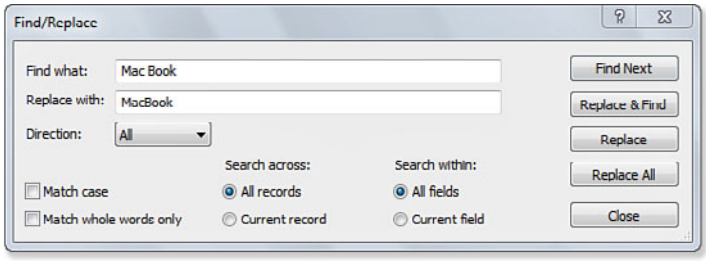
in maintaining your data integrity. To quickly take care of migrating from an old value to a new one, follow these steps:

1. Choose Records, Show All Records; otherwise, your change is applied to only your current found set.
2. Place your cursor into the field in question.
3. Choose Edit, Find/Replace to open the Find/Replace dialog box (see Figure 2.41).
4. Type your old and new values.
5. Choose All from the Direction drop-down menu so that your entire database will be covered.

note

In the case of a field that is shown with a pop-up menu, you're out of luck. FileMaker Pro doesn't recognize a cursor in a pop-up menu. You need to do a little development (covered in Chapter 3, "Defining and Working with Fields and Tables"), copy the field to an open spot on your layout, and change its formatting to a pop-up list. Then don't forget to delete the layout field when you're finished.

Figure 2.41
Find/Replace can step through your records or be applied across the entire database. Be careful: You cannot undo these functions!



6. Depending on your preferences, choose Current Field or apply your change to entire records. We recommend changing just the selected field because that's much safer than accidentally changing all instances of a text string.
7. Click Replace All.

caution

It's important to note that this is a function that you cannot undo! Be sure that you know what you're doing with your data.

Technique 4: Inserting Specific Information

The Insert menu is an often-ignored source of handy time-saving commands. From a single menu choice or keyboard command, you can insert the current time, the current date, or your username into an active field.

In addition to that, Insert, From Index enables you to select from all the values in a given field from all records in a database. If you can't quite remember the spelling of a given item, or you simply want to be perfectly consistent, this is a great way to see the data in your system and make a compatible selection. This works only if the field in question allows indexing.

To learn about field indexing, ***see*** *"Storage and Indexing,"* ***p. 104****.*

Finally, there's a handy way to pull data from another record in your database. If three or four fields need to contain data identical to another record in your database, visit the source record first and then, via a List view or Table view, jump to the destination record by clicking the appropriate row. Click into the specific fields you want and choose Insert, From Last Visited Record.

Technique 5: Getting to Know Your Entire Database

This item isn't so much a technique as it is just common sense: One of the best ways to make the most of a FileMaker database is to learn how it works. Review all the layouts in your system, take a look at the fields you see, and explore other files (if there are others) in the solution. Be sure to discuss with your developer how the information fits together.

Technique 6: Using Multitiered Sorts

Sorting can be a fairly powerful way to derive meaning and see patterns in data. To make the most of the Sort Records dialog, don't forget that you can provide multiple sort criteria. For example, in a contacts database, you could sort by Last Name, First Name, City, descending by Age, and finally by Pet Name.

You can also sort by the custom order of a value list. If you have, say, a status field managed by a value list of "open, pending, closed," you can sort by that order.

Technique 7: Using Multiple Windows

FileMaker provides you with a Window menu. If you'd like to work with multiple layouts at once, choose Window, New Window, and then navigate to the second layout in question by using either the Layout pop-up menu in the Status toolbar or the buttons provided by a developer.

Multiple windows are also useful when you open two windows looking at the same List view layout: It's possible for you to have two separate found sets. Imagine finding all the invitees of an event in one window and all the people you've not yet invited in the other.

If you are designing a solution that might be used on FileMaker Go, remember that multiple windows cannot be opened at the same time on iOS devices (you move from one to the other with the control at the left of the toolbar at the top of the screen).

Technique 8: Applying Text Styling and Tabs

You can apply a wide range of formatting options to text within FileMaker Pro fields: bold, italic, font choice, color choice, and so on. FileMaker Pro preserves this information, and you can copy and paste formatted text with other applications.

There is another neat trick in FileMaker Pro: In any field, you can establish an internal tab placement and apply tabs by using (Command-Tab) [Ctrl+Tab]. Choose View, Ruler. When you click into a field, a horizontal ruler appears above it, into which you can click to establish tabs. Double-click a tab to set its properties: left, center, right, align to character, and whether to use a fill character.

3

DEFINING AND WORKING WITH FIELDS AND TABLES

Working Under the Hood

Fields and tables are the heart of any database. By storing information in properly named fields within well-organized tables, you impart both function and meaning to what could otherwise be an incomprehensible pile of raw data.

This chapter describes what kinds of fields exist in FileMaker, how they store information, and how to ensure proper data integrity in your database solutions. You will also find naming conventions for fields and tables—techniques that you can use to make your FileMaker databases meaningful to yourself and others for the long period of time that they may be in use.

If you're new to development in FileMaker Pro, this chapter is a good place to start. Establishing a solid foundation in field definition is a vital part of becoming a practiced developer.

Creating New Databases

To create a new database, simply launch FileMaker Pro, and then choose File, New Database or File, New Database from Template. Alternatively, from the Quick Start screen, you can choose Create Database to get started. The Quick Start screen also lets you choose to create a database from an Excel workbook, a tab-delimited text file, a comma-separated values text file, a merge file, or a Bento source. As is the case with much of FileMaker, you can let FileMaker do a lot of the work or you can get deeply involved with the construction of your database. The simplest

strategy is to either use a Starter Solution or to let FileMaker Pro work with you to create a basic database.

In the following section, "Using the Manage Database Dialog," you learn how to get more involved with the building of a database. You also learn how you can switch back and forth between the simple and more complex ways of working.

If you use File, New Database to create a new empty database file, you are prompted to provide a filename and a location to use for that file. Based on that name, FileMaker creates the database and opens the new database in Table view and Browse mode, as you see in Figure 3.1.

note

The Use Manage Database Dialog to Create Files in FileMaker Pro, Preferences enables you to adjust this setting. Also note that when you hover the mouse over the table the + might appear; it might disappear when you move the mouse away.

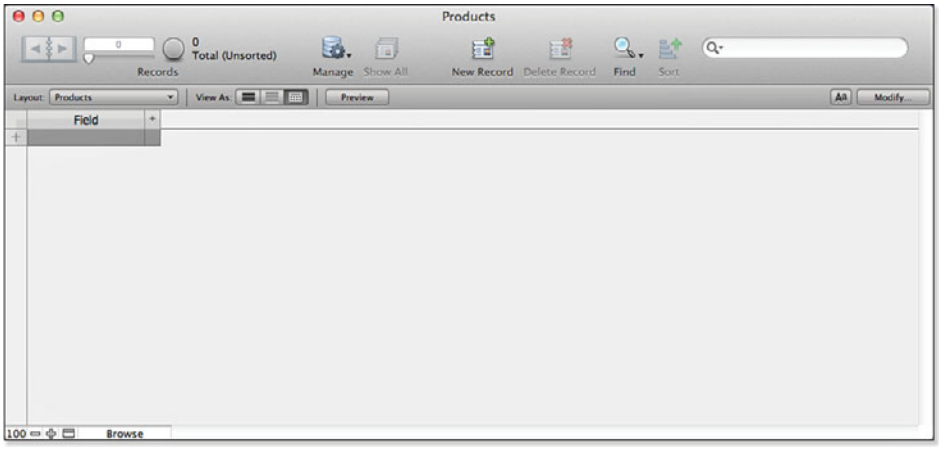


Figure 3.1 Create a new empty database.

As you can see in Figure 3.1, Table view looks like a spreadsheet. There is one field, labeled Field. There are no rows in this table, but there is a + at the left, indicating that you can click in that row to create an empty cell in that row.

If you click in that row, you'll see that the + moves down one row, and you can type in the data for the first field and first row. The + continues moving down, and you can always click next to it to add another row to the table. Figure 3.2 shows one entry in the table.

As shown in Figure 3.3, you can add another row in the same way—click next to the + and enter the data.

Figure 3.2
Create the first record.

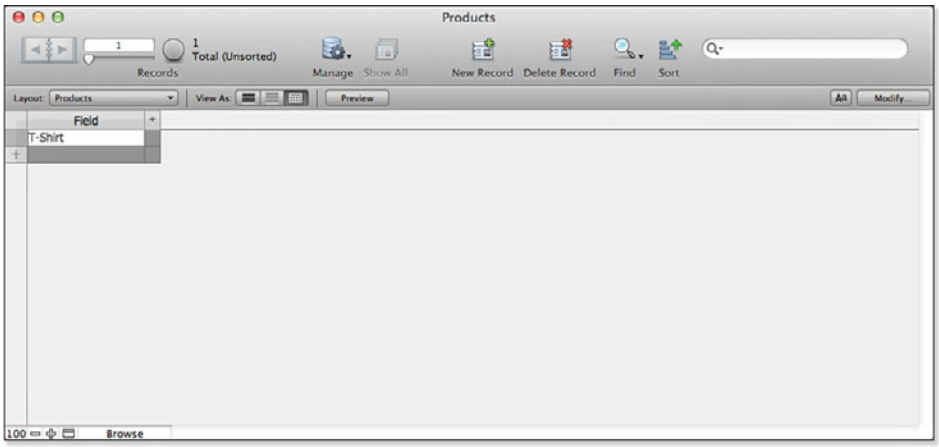
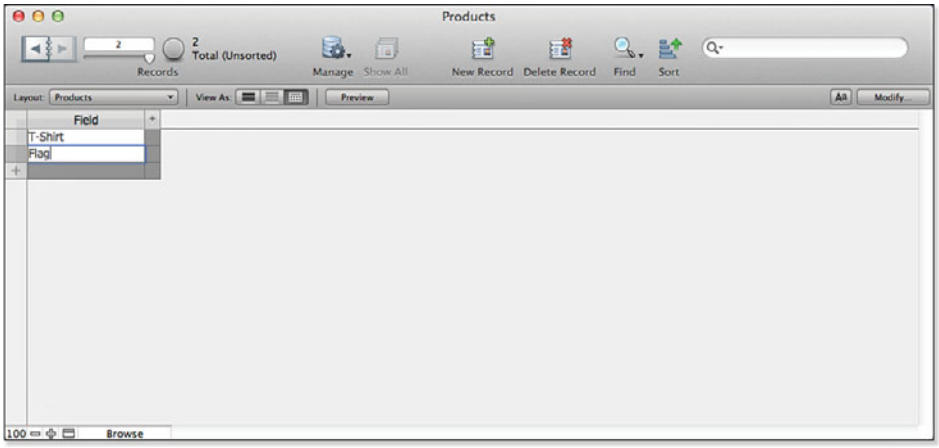


Figure 3.3
Add another record.



If you hover the mouse over the title of a column, as shown in Figure 3.4, you have access to a shortcut menu with commands to use in modifying the data.

Double-click in the top row to change the name of the field to something more meaningful than Field. Figure 3.5 shows the result.

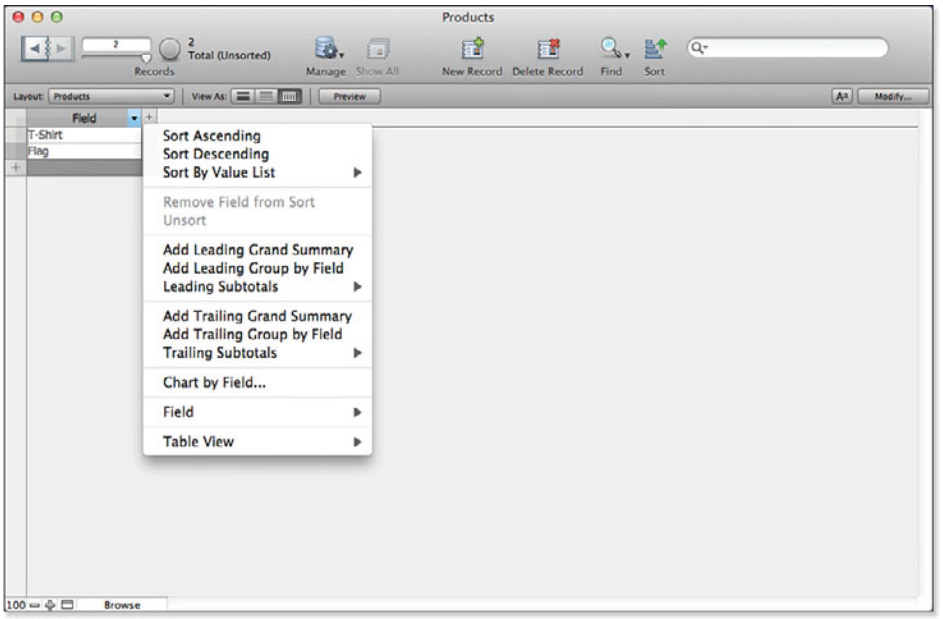


Figure 3.4 Access shortcut menu commands.

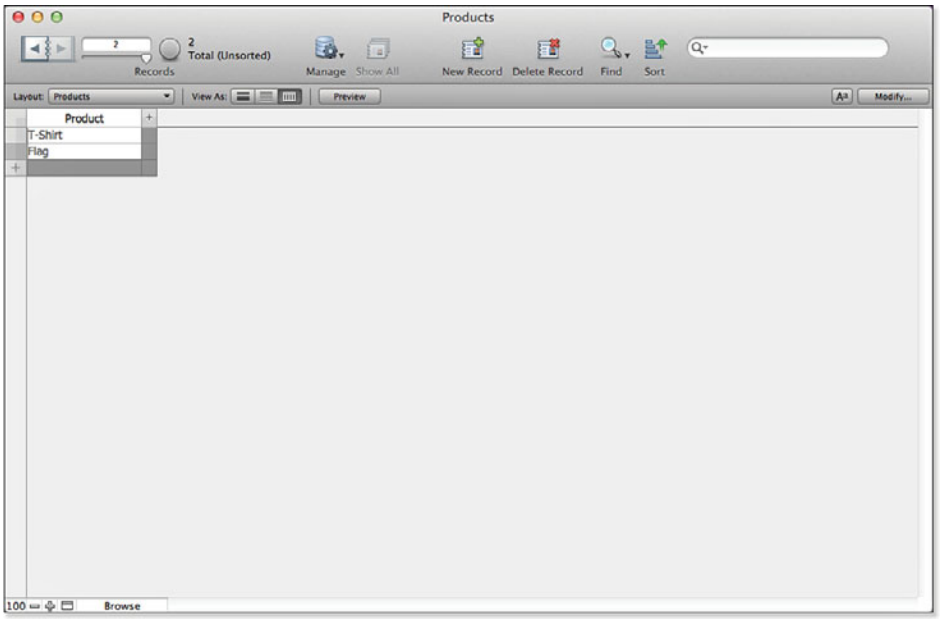


Figure 3.5 Provide a meaningful name for the field.

You can add another column or field by clicking the + at the right of the first row. Figure 3.6 shows the result.

Name the second field, as shown in Figure 3.7.

tip

Remember that *column* is spreadsheet-ese, whereas *field* is database-ese.

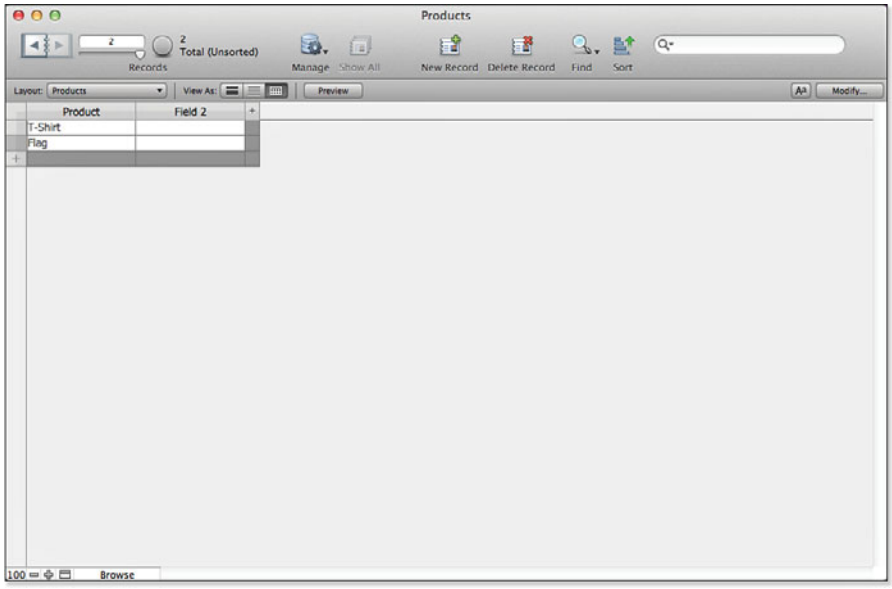


Figure 3.6
Add another field.

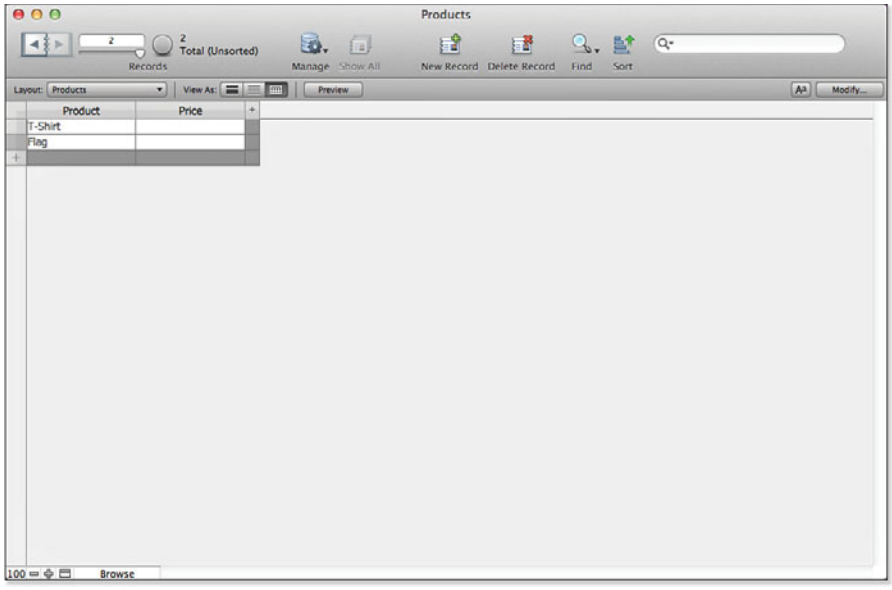


Figure 3.7
You should name fields as soon as you create them.

You now have your database. You can add more data (rows) and more fields (columns) in the same way. You also can use the down-pointing arrow to select commands for each column. Some of the commands, such as for sorting, are straightforward; others are described throughout this book.

You can work with a database constructed in this way very quickly. Behind the scenes, FileMaker has built the database that you described with a few mouse clicks and keystrokes. However, to unleash the power of FileMaker Pro, you can use other tools and techniques.

Using the Manage Database Dialog

You can gain access to the internal structure of the database using File, Manage, Database. The resultant dialog is shown in Figure 3.8. As a developer, you'll spend a good bit of time in the three tabs in this dialog. FileMaker Pro's Manage Database dialog allows you to create the fields, tables, and relationships you need to form your database. It also enables you to modify a wide range of attributes associated with fields, such as auto-entry functions, validation, storage, and calculation formulas. These elements comprise a database's structure or *schema*. It is here that you form your database behind the scenes.

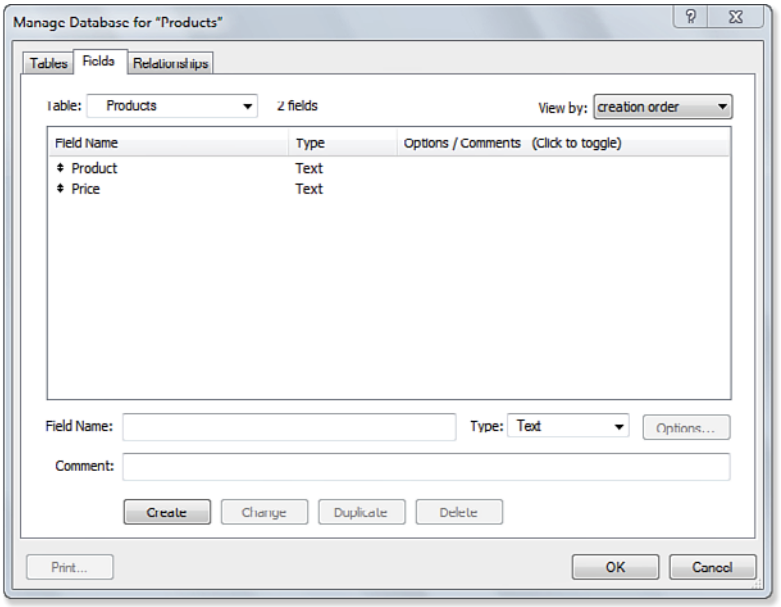


Figure 3.8
The three tabs enable you to switch among defining tables, fields, and relationships.

If you are creating a new database from a Starter Solution or an existing file such as an Excel workbook or a text file, FileMaker Pro automatically creates the necessary schema and opens the new database without going through the Manage Database dialog. While you are working with a database, you can open the Manage Database dialog at any time to modify the schema.

FileMaker Pro will have already created a default table for you, named the same as the file itself. Notice the Table pop-up menu on the Fields tab of the dialog in Figure 3.8. Any fields you create will be created in that table.

*For some basic information on tables, **see** "Understanding Tables," **p. 29**.*

*For a detailed discussion of multiple-table solutions, **see** Chapter 6, "Working with Multiple Tables."*

Notice the third tab in the Manage Database dialog: Relationships. We don't cover relational databases in this chapter, but it is on that tab that you would create the relational associations among tables in your solution.

*For information on relational data modeling, **see** Chapter 5, "Relational Database Design."*

Working with Tables

As you saw in the preceding chapter, your database consists of tables, each of which is made up of rows or records with columns or fields that contain the data. A database can consist of a single table or a number of tables.

By default, when you create a new database, a single table is created that has the same name as the database. That actually might not be what is best. You might want to rename that default table so that it fits into the naming convention of all the tables in your database.

Table Naming Conventions

The Manage Database dialog lets you create and name (and rename) fields and tables. It is a good idea from the start to enforce some naming conventions on both fields and tables.

FileMaker Pro's flexibility with regard to things such as legal characters in names and the length of names for tables and fields can be too much of a good thing. You can use up to 100 characters in a name, but chances are you will need far fewer for your actual names.

Stick with the conventions—that is, within a single database or even a project. One problem with implementing design conventions is that the world is a large place, and it is likely that your naming conventions will need to interact with naming conventions of other systems and databases. Being internally consistent keeps your own house in order. That is the most that you can hope for, unless you volunteer to serve on a committee that drafts conventions for your organization, industry, or other group.

note

This section includes some suggestions based on conventions used by various FileMaker developers. However, you can find more information about conventions in the Support area of the FileMaker website. In addition, FileMaker's TechNet membership gives you access to still more information and guidelines. Pick what are the most useful conventions, but stick with them.

Naming tables is simultaneously simple and almost irrelevant. The reason is that as soon as you have a database with more than one table in it, you will most likely be using the Relationships Graph (described in Chapter 7, "Working with Relationships"). The Relationships Graph initially

shows each table with the name you assign to it. However, you will create additional instances of your tables in the Relationships Graph, and you will name each of them. In practice, you will usually be working not with the base table, but with the additional instances.

For example, you might have a table called Personnel. In the Relationships Graph, you might have instances of this table called PersonnelByID, PersonnelByName, PersonnelByDepartment, and so forth. Practically, you could name the base table Table 1, and, as long as the other names appear in the Relationships Graph (and in your code), everything would be clear (keep in mind that this is presented only as a hypothetical example, not a good practice).

When you create a database, by default you will wind up with a database, a single table, and an instance in the Relationships Graph all with the same name. Many people begin by renaming that first table right away. Here are some of the suggested standards:

- Use only the characters 0–9 and a–z (both uppercase and lowercase).
- If table names contain several words, separate them with underscores or with intermediate capitalization (as in personnel_salary_info and personnelSalaryInfo, respectively).
- Be consistent in capitalization and number (that is, use table names such as Contacts or Contact, contacts or contact).
- Do not use special characters or reserved words in table names. Reserved words include FileMaker reserved words as well as words that might be reserved in SQL or other languages you can use to access the tables. *Select* is not a good table name because, although it might be useful for storing selection values for records in your database, it is a SQL reserved word.

In addition, consider whether you want to place any descriptive information in the table name. If you do so, the usual convention is to place it at the end following an underscore. This approach is particularly useful if you separate words within the table name using intermediate capitalization. For example, inventorySuppliers_pub and inventory_Quantities_pri are reasonable names for inventory tables that, respectively, contain the publicly available names and addresses of suppliers and the private quantities of inventory items on hand. You can enforce access to these tables with your security accounts and privileges, but it can be useful to indicate not only what is in the tables but also the sensitivity of the data.

Explore the Starter Solutions to see various examples of naming conventions. Note that some people disagree with the use of the pipe character (|) in some of the field and table names. As noted earlier, it can cause issues when interacting with SQL. Whether this is relevant to your project is up to you. Also, be aware that deep down inside FileMaker, the names of objects are converted to codes. Thus, a field named Address is ultimately stored as field number N. If you rename a script, field, layout, or table, its internal ID number is unchanged. This means that if you reference a field from a script or layout, changing the name of the script or layout does not break the script or layout: it simply uses the new field name. Thus, worrying about interoperability of field and table names is an issue that can easily be put off for the future.

Creating New Tables

To create a table, go to the Manage Database dialog (File, Manage, Database). Click the Tables tab to see the view shown in Figures 3.9 and 3.10. Note that this is one of the places in which FileMaker Pro and FileMaker Pro Advanced differ.

Figure 3.9
Use the Tables tab in Manage Database to create, change, and delete tables in FileMaker Pro.

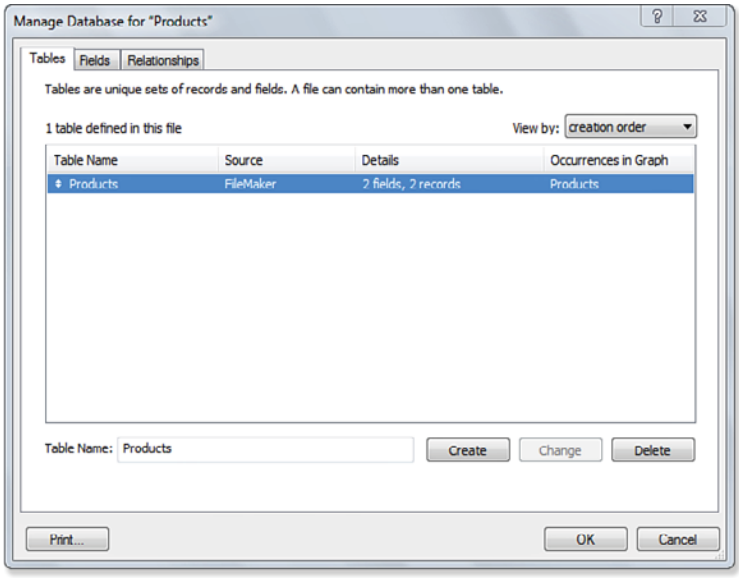
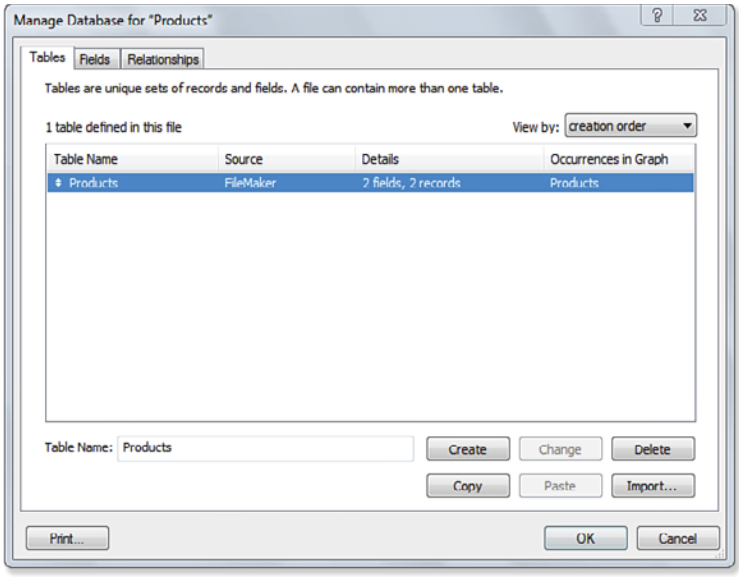


Figure 3.10
Use the Tables tab in Manage Database to create, change, and delete tables in FileMaker Pro Advanced.



To create a new table, enter a name for the table at the bottom. Click Create, and your table will be created in the list of tables. An instance of the table will also be created automatically in the Relationships graph. To rename a table, highlight its name in the list of tables in the Tables tab, type in the new name at the bottom, and click Change.

To delete a table, highlight its name and click Delete. If you want to print out the fields in one or more tables, highlight them and click Print.

You learn how to import data along with the table definition using the Import command in Chapter 22, "Importing Data into FileMaker Pro."

note

The Manage Database dialog in FileMaker Pro Advanced has three additional buttons in the lower-right corner, as shown in Figure 3.10. You can copy a table and paste it in your database (thereby duplicating it), or copy and paste it into another database. You can also import a table definition, not the data, from another FileMaker Pro database.

Working with Fields

The heart of the database is the data within it—data that is stored in fields. This section provides some basics about working with fields.

Field Naming Conventions

The naming conventions for tables with regard to spaces, characters, capitalization, and so forth apply also to fields. There are some additional considerations when it comes to naming fields. Specifically, they have to do with the identification of field types and the naming of internally used fields.

Many developers use abbreviations for data types in field names. Often it's handy to know the data type of a given field when working with it without having to refer to the Manage Database dialog. Here we've used *t* for text, *n* for number, and *c* for calculation:

- ProductName_t
- Price_n
- TaxRate_n
- Tax_c

Some developers also note whether a field is indexed (*x* for indexed, *n* for unindexed):

- Location_Name_tx
- Location_Desc_tn
- Location_Size_nn

Some naming conventions also break out a division between data fields and what are commonly referred to as *developer fields*—those fields that you need only to make your FileMaker Pro solution work. If you ever went to import your database wholesale into another system, you would probably

leave behind the developer fields. Here, we have two abbreviations: *k* for key (or match field) and *zz* (so that it sorts to the bottom of an alphabetized list of fields) for developer utility fields. We also use underscores to ensure that keys sort to the top of our field list, with the primary key coming first.

*To understand how keys are used to identify records in tables and form relationships, **see** Chapter 5, "Relational Database Design."*

- __kp_primary_AlbumID
- _kf_foreign_ArtistID
- AlbumName
- Date
- zz_SelectedPortalRow
- zz_UserColor_Preference
- zz_UserGenre_Preference

Many developers use a minimal set of field-naming standards. It relies on leading lowercase characters to indicate the field type. If you choose to use that minimal set, here are the conventions used:

- **g**—Global.
- **c**—Calculation.
- **s**—Summary.
- **zz**—Internal use. (This causes the field name, when shown in an alphabetical list, to be at the bottom. If you use a single z, your internal fields will be interspersed with fields such as ZIP code.)

Another naming convention that you will see in some Starter Solutions uses capital letters for internally used fields. For example, in Research Notes you will find TYPE MATCH FIELD, which is a field that is used to match related records based on their type.

Descriptions of field types might or might not use this set of standards, which you can add to the end of the field name following an underscore:

- **t**—Text
- **n**—Number
- **d**—Date
- **ts**—Timestamp
- **tm**—Time
- **c**—Container

Putting these together, you could have field names such as these:

- creationDate_d
- gProcessingOffice_t
- gcNextInvoiceNumber_n

You can even go further by not bothering with a field type where the field name already includes it. creationDate_d really adds no information to creationDate.

Whatever you do, be consistent. The point is not to create a set of naming conventions that overshadows the database but, rather, to create naming conventions that help you and future developers build and maintain the solution.

tip

Don't imagine that all the fields on your Relationships Graph will adhere to these naming conventions. You control your own fields, but as you begin to use external data sources, you will be incorporating fields from other databases. You can have a field called payrollDate_d in your own table, but if you are relating it to a field called datePaid in the corporate database, chances are slim that the database administrator will want to rename the field to make it consistent. The ProjectID field in your database might be related to a field that is called JobNumber in another FileMaker database that you do not control. And, in a global world, the external data source names might just be in another language. Be as clear and consistent as you can, but do not assume that you can control the names of fields in other databases. (In general, the owner of the Payroll database wins out.)

*If you're planning on using FileMaker Pro as a web backend, **see** "Problematic Field Names" in the "Troubleshooting" section at the end of this chapter.*

*For more information on using databases on the Web, **see** "Designing for IWP Deployment," **p. 587**, as well as Chapter 25, "Custom Web Publishing with PHP and XML."*

Adding Field Comments

Notice also that you can add comments to your field definitions. Field comments are a clear indication that the database designer has thought through the database design. They also are a good example of the difference between the simple building of a database, as shown at the beginning of this chapter, and the deeper work that you can do with the Manage Database dialog: Comments can only be entered through the Manage Database dialog.

Commenting is a vital discipline to develop. Spending a few moments to add information to the Comment text box, below the field name, as you create a field will save time later in trying to figure out what you were thinking at the time. Don't bother repeating information that is in the field name. If the field represents pixels or pennies, adding that information to the field

tip

You can extract field comments using the `FieldComment` function so that you can use them in a tooltip or other dynamic documentation in your solution. If you use that design, remember that your comments will appear to end users, so word them accordingly.

name might be worthwhile (as in Width_In_Pixels). Use comments for in-progress remarks (such as "Added 4/1/2013 JF for task restructure" or "for reporting only").

Creating New Fields

To create fields in FileMaker Pro, you need to enter some text in the Field Name area of the Manage Database dialog and click Create. One important aspect of databases to keep in mind is that it's important to establish a discrete field for each bit of information you want to store. If you create a field called Contact Information and cram an entire address and a set of phone numbers into it, technically it will work fine. But if you ever need to export that information, sort by area code, or run a report by city, you won't be able to cull the information you want from the field without suffering a good headache.

To database wonks, the Contact Information example would be a violation of first normal form—or more colloquially, "one fact, one field."

*For information on relational data modeling and defining fields, **see** "Relationship Types," **p. 159**.*

As shown previously in Figure 3.1, the Manage Database dialog lets you create, change, duplicate, and delete fields. As is the case with the Tables tab, FileMaker Pro Advanced offers additional buttons: Copy and Paste. If you select a field or fields, you can click Copy and then paste the fields into the same or another table. Pasting them into the same table is the same as duplicating them.

Working with Field Types

Some of the most important aspects of understanding FileMaker Pro involve comprehending field types, realizing how they differ from one another, and knowing how to use them effectively. Simply stated, field types identify what kind of information each field of your database expects to hold. A person's name is text, the purchase amount for a transaction is a number, a birthday is a date, and so on. Generally, it should be quite clear to you what each needs to be.

Use the most specific field type you can. This allows you to use FileMaker Pro editing and formatting. Although FileMaker can convert a text field to a number where necessary, it can apply numeric formatting only to a number field. The same goes for dates and times.

Field types determine what types of operations can be performed on a given field, what information a field can accept, and the rules by which a field is sorted. The combination of a proper identifying field name and a data type definition is what gives a database its context and meaning. You can set field types in the Manage Database dialog, or in Table view you can use the Field, Field Type submenu in the shortcut menu.

Text

Text fields are the most free-form of the field types. Users can enter any range of information in them, including carriage returns, and there's no expectation of what form or sort of information a text field will hold. The only requirement is that it be character based; in other words, you can't place a picture in a text field. A text field can store up to 2GB of information (limited by RAM and hard drive space, of course) and indexes up to approximately 100 characters, depending on what language you're using. We cover indexing in more depth later in the chapter. For now, simply remember that each field type has different limits and approaches on indexing.

Number

Number fields can store values from 10^{-400} up to 10^{400}, and negative values in the same range. FileMaker Pro indexes the first 400 significant digits (numbers, decimal points, or signs) of a number field, ignoring letters and other symbols. Number fields can accept text (although not carriage returns), but any text in a numeric field is ignored. FileMaker interprets `12ax3` as `123` if you enter it into a numeric field, for example.

Something to keep in mind with FileMaker Pro: You can express a number field as a Boolean. A Boolean value is either true or false, and often used to test the condition of something. FileMaker Pro treats a zero or null value in a number field as false in the Boolean sense; it treats any other data as true. You will often run across the use of number fields to store Boolean values.

The primary distinction between a number field and a text field lies in how they sort: A text field sorts 1, 10, 2, 20, 3, 4, 5, whereas a number field sorts 1, 2, 3, 4, 5, 10, 20.

Date

Date fields accept only Gregorian calendar dates. FileMaker Pro honors whatever date formatting your country follows by taking the standard your operating system uses at the time you create a new file. Date formats—the order of year, month, and day—are common for a given file. Although it's possible to change the way FileMaker Pro displays dates, it fixes basic ordering at the time of file creation.

Dates in FileMaker Pro are internally stored as the number of days since 01/01/0001. January 1, 2013, for example, is 734869. If you need to compare dates or perform any functions on them, remember that behind the scenes they're really just numbers. This feature is actually quite handy. To switch a date to a week prior, all you need to do is subtract 7. Date fields can store values from January 1, 0001, to December 31, 4000.

➥ *If your fields are sorting or displaying oddly, **see** "Mismatched Data Types" in the "Troubleshooting" section at the end of this chapter.*

Time

Time fields hold HH:MM:SS.ddd information. Notice that you can add a decimal to the end. An additional useful fact: If a user enters **25:00**, FileMaker Pro rightly interprets this as 1:00 a.m. Similarly, 99:30 becomes 3:30 a.m. The clock simply keeps rolling over. This behavior is useful when you need to add, say, 30 hours to a time and don't want to be bothered with calculating what hour that becomes. Likewise, if you are doing data entry in a time-tracking system and don't want to create two entries for a case in which you worked from 2:00 p.m. until 2:00 a.m. on Monday (really Tuesday), entering **26:00** for the ending time in your system rightly calculates to 12 hours.

The maximum time value you can store in a FileMaker Pro time field is 2,147,483,647. That's a lot of time.

As in dates, FileMaker Pro stores time internally as the number of seconds from 12:00:00 on the current day: 1 is 12:00:01, and 43200 is 12:00 p.m. As it does with date formats, FileMaker Pro establishes your time format during the creation of the file, based on the operating system settings.

Timestamp

The timestamp data type combines date and time information. It appears as a field with both date and time values, separated by a space: 1/1/2013 12:00:00. As in date and time formats, timestamps are also stored as numbers: the count of seconds from 1/1/0001 00:00:00. Be prepared to work with large numbers when using this field type. Timestamps are an important aid to interoperability with other databases (such as those powered by the SQL language), which often store date and time information in a single timestamp field. The maximum value of a timestamp is 12/31/4000 11:59:59.999999 p.m. or 126,227,764,799.999999 seconds.

To extract just the date from timestamp data, simply use the `GetAsDate()` function. Likewise, use `GetAsTime()` to extract just the time. In a layout, you can format a timestamp as a date or as a time (as well as leaving it as a timestamp). If you format a timestamp as a date, the time value is not shown in the layout.

Container

Container fields are different from the ones already mentioned: They store binary information. Information is often inserted into container fields rather than being entered manually (you can copy and paste). You can place any sort of digital document in your database, limited again by the practical limits of your computer hardware, up to 4GB.

Beginning with FileMaker Pro 12, there are two sets of considerations for container fields. With a container field selected, you can use the Insert menu to insert a file in the field. At the bottom of the dialog, you have the option to insert the file itself or only a reference to it. If you insert a reference, FileMaker looks for that file every time it needs to display the data in the container. This keeps your database file smaller because the external files are located elsewhere on disk. However, the reference identifies the file, and if the file (or the FileMaker Pro database) is moved, that reference is likely to break.

FileMaker Pro 12 adds an extra option: FileMaker Pro can place the files for container fields into a directory in a known location. This means that you can have the advantage of keeping your database file small by not including container field files inside the database, but you do not need to store fragile references. (This is the design of many industrial-strength databases.) Users still have the option to store references or the complete file for every record in the database when they use the Insert menu, but if they choose to store the file instead of a reference, FileMaker (and you, the database designer) take care of where the file is placed.

*For additional information on displaying files in container fields, **see "Using the Inspector," p. 137**.*

When you set a field to be a container field, you have the ability to set options for it. The Options button, shown previously in the lower right of Figure 3.8, is available for a number of field types such as calculations as well as containers. Click it to open the Options dialog shown in Figure 3.11. Click the Storage tab at the top to continue.

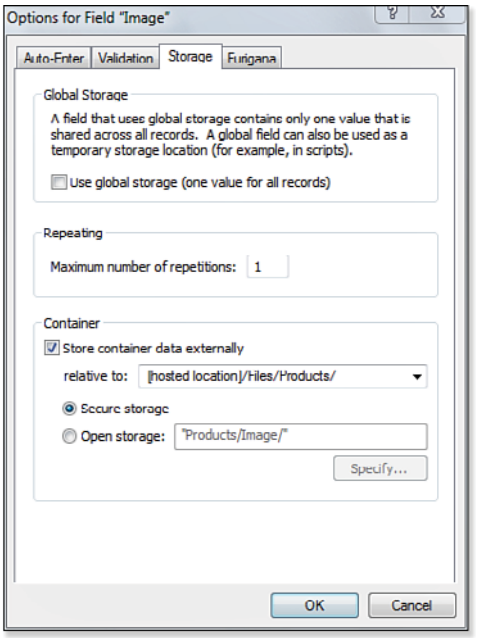
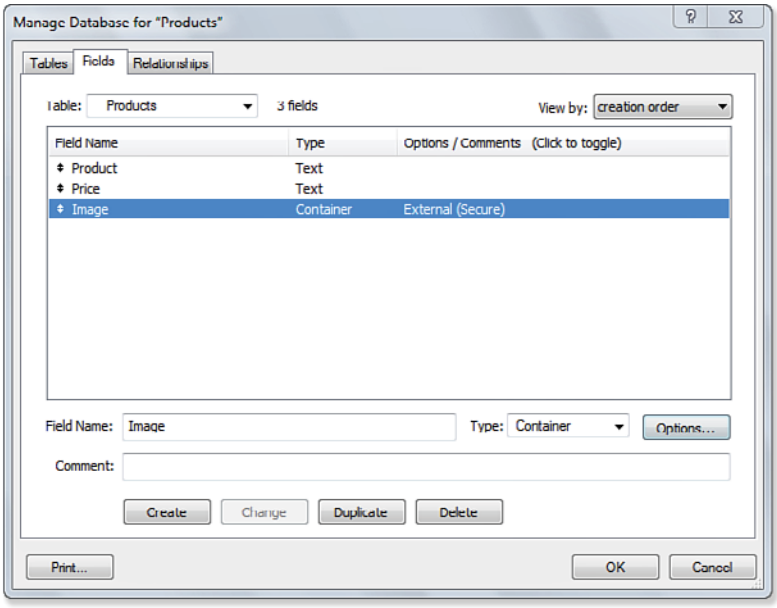


Figure 3.11
Choose storage options for a container field.

The check box at the bottom of the dialog lets you specify external storage for the contents of container fields. By default, that storage is in a folder called Files/<database filename>/. That folder is placed next to the database itself. Thus, if your database is named Products and is placed in a folder called Inventory, the container field files will be in a folder called Inventory/Files/Products/<filename>.

At least, that will be the case if you choose the option for open storage. This may give you a sense of confidence because you'll be able to see the files that are stored in your container fields, but so will other people. For that reason, the default is secure storage. Files are stored in encrypted versions. Once you have set the storage options, you will see them displayed in the Manage Database dialog for the relevant field, as you see in Figure 3.12.

Figure 3.12
You can choose open or secure storage for files.



If you make a change to the storage options for a container field, you are given the opportunity to transfer existing data to the new storage option, as you see in Figure 3.13.

On completion, you are informed of the results, as you see in Figure 3.14.

You can customize the storage locations for files in two ways. If you are using open storage, you can browse to the directory where you want the files to be placed; however, this can have the problem of broken references, as mentioned previously. In some production environments, a separate server is dedicated to storing files and, in those cases, you don't have to worry about the files being moved.

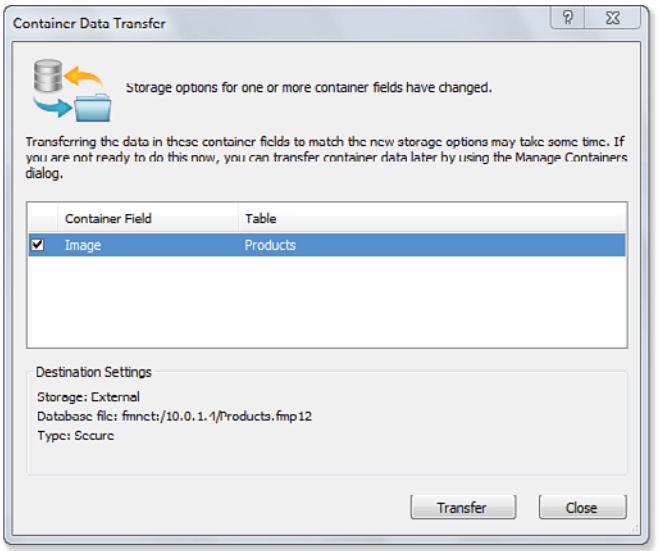


Figure 3.13
FileMaker can move your data to the new storage option.

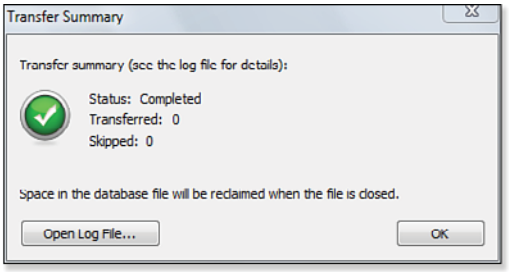


Figure 3.14
Review the results of the transfer.

You can also manage storage locations using File, Manage, Containers, as shown in Figure 3.15. You can add other directories to be shown in the pop-up menu shown previously in Figure 3.11. You still have the option to customize storage for each field, but this strategy is appropriate if you have a dedicated location for all (or at least many) external files.

The final piece of the puzzle with regard to container fields and file storage is a little bonus from FileMaker in FileMaker Pro 12. In the File, Manage, Container dialog, the Thumbnails tab shown in Figure 3.16 lets you store automatically generated thumbnails of images. This can dramatically improve performance.

Figure 3.15
Set container options for storage directories.

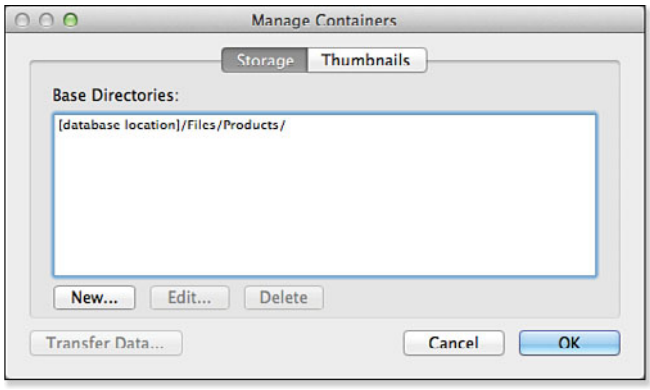
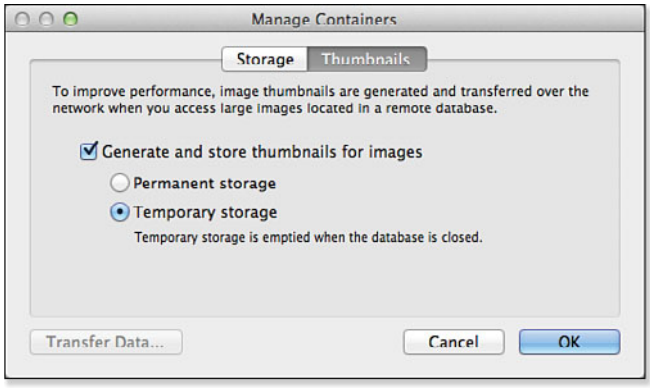


Figure 3.16
Manage automatically generated thumbnails.



Calculation

Calculations evaluate formulas and return the requisite results. When you create a calculation field, the Specify Calculation dialog opens, as shown in Figure 3.17. You use the same dialog to specify calculations used for script parameters, web viewers, security privileges, and other purposes in FileMaker Pro.

Features of the Specify Calculation dialog include the following:

- **Field list**—Select fields to include in your calculation from the list below the table menu. Use the drop-down menu to change from table to table. Note that double-clicking inserts a field into your calculation where your cursor currently sits.
- **Operators**—Use these buttons to insert math and special operators.

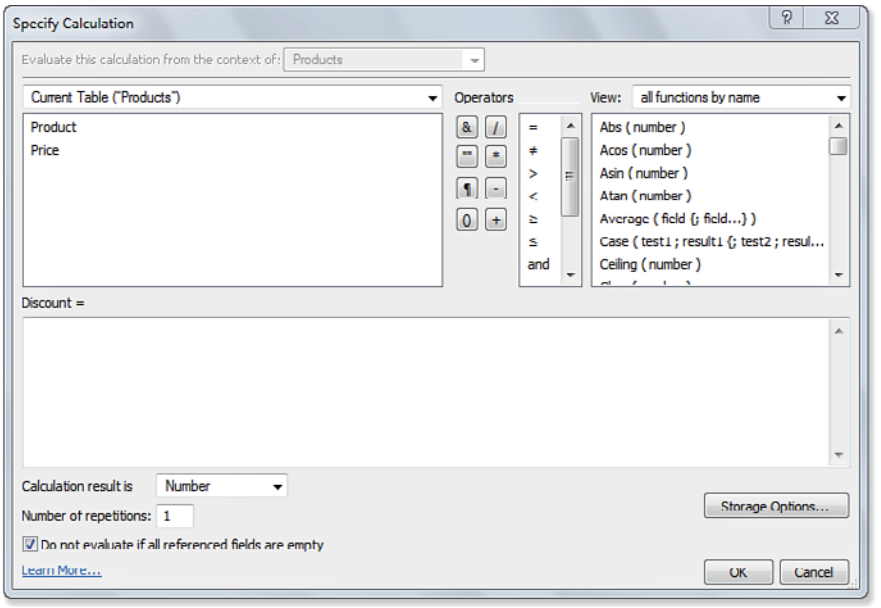


Figure 3.17
Calculations form an essential part of FileMaker Pro development.

- **Function list**—Just below the View drop-down menu is a list of functions. Here, you're able to scroll through all of FileMaker Pro's various functions and then double-click to insert. It's a good idea to start here to get your syntax correct. The menu above enables you to filter your list by category to show the functions you need.
- **Expression text box**—This is the place where you assemble your actual formula or expression. This is a simple text entry area: If you want, you can work in a text editor and paste the calculations here.
- **Calculation Result Is list**—Calculations return varying information, depending on what data/field type is required. If you want the field to be sortable by alphabet, set the return data type to Text. If you have a field returning, say, a price, set the type to Number.
- **Learn More link**—You can find more information by using this link in the lower left of the dialog.

Examples of calculations include the following:

- `3 + 4` always displays its result of 7.
- `Sale + Tax` displays the sum of two fields named Sale and Tax.
- `Personnel::EmployeeID` displays the value of a field in a related table. This type of calculation is sometimes utilized to create a field in a table that takes part in a sort or other routine where

you cannot use a related field. In old FileMaker Pro databases, relationships cannot be used more than one table away. Calculations designed simply to provide an in-table copy of a related value frequently litter such databases.

- `Position ( Notes; "a"; 1; 1 )` returns a numeric position, starting from the first character in the field Notes, for the first *a* found.
- `IsEmpty ( MyField )` returns a 0 or 1 (Boolean) depending on whether `MyField` has a value in it, including 0. If 0 is entered, the field is technically not empty. Only a null value is considered empty.
- `If ( MyDate > 900; "yes" ; "no" )` displays a `yes` for dates entered in `MyDate` greater than 6/19/0003; otherwise, it displays `no` (remember that you just tested for the number of days past 1/1/0001).

You can use the Specify Calculation dialog to create a calculation just by clicking fields, operators, and functions. However, you can also type directly into the expression text box. As you saw in Figure 3.17, you can spread out your calculation; spaces do not matter except within quotation marks. You can also use indentation to clarify the calculation. Comments can be inserted using two slashes (`//`), which mean that the remainder of the line is ignored. Multiline comments can be entered starting with `/*` and ending with `*/`.

*For more detail on calculations, **see** Chapter 8, "Getting Started with Calculations," and Chapter 15, "Advanced Calculation Techniques."*

*If your calculation formula looks correct but FileMaker is returning an odd result or `?`, **see** "Mismatched Calculation Results" in the "Troubleshooting" section at the end of this chapter.*

tip

You can use calculations to create calculation fields with data derived from other fields or constants. Calculations can also format data, just as fields in layouts can be used to format data.

In general, good database design separates the presentation of data from the content of data, and layouts are the primary tools to be used to format data. However, with FileMaker, the situation is now not so clear. Because you can access FileMaker Pro databases over the Web, with ODBC, and from remote copies of FileMaker Pro that use their own layouts, you might want to consider formatting data with calculations, rather than layouts. Calculation fields that round a number to two decimal places or that perform automatic formatting of dates and so forth produce formatted results visible to all potential users of the FileMaker Pro database, not just those using a layout in the database itself.

Summary Fields

Summary fields enable you to evaluate information across a found set of records. Sum, Average, Max, Min, and Count are among the summaries you can establish. Don't forget that they apply to found sets: Change your found set, and the result changes.

Summary fields can be placed in subsummaries, where they summarize data for a specific subset (perhaps individual clients or dates). FileMaker takes care of summarizing only the appropriate data, as you learn in Chapter 4, "Working with Layouts."

For example, say you have a table called Transaction, which contains Transaction_Date and Transaction_Amount fields. You can then define and place a summary field on a layout to total the Transaction_Amount field. The summary field adds the values of the Transaction_Amount fields for the currently active set of records. If you perform a find, by date, on 10/1/2013–10/31/2013, your found set will be all the transactions for the month of October, and the summary field will show just the aggregate monthly transaction amount. Perform a different find request and your total changes, reflecting the aggregate of the new found set. Table 3.1 contains a list of summary field functions.

Table 3.1 Summary Field Functions

Function	Summary Behavior
`Total of`	Adds values from the specified field in your found set. Think of it as a subtotal or grand total from a column of numbers. You can also enable the option to display a running total for your record set. This shows a running tally of your total if you place the summary field in the body area of a list.
`Average of`	Averages the values from the specified field in your found set. The weighted average option enables you to specify a second field to act as a weight factor for calculating the average. The field you choose must be a number or a calculation with a number result.
`Count of`	Counts the number of records in your found set that have data in the specified field. For example, if 18 of the 20 current found records have data, your summary field displays 18. A running count functions similarly to a running total: It displays the incremented count of each record in your record set.
`Minimum`	Returns the lowest number, date, time, or timestamp in a given found set from the referenced field.
`Maximum`	Returns the highest number, date, time, or timestamp in a given found set from the referenced field.
`Standard Deviation of`	Determines how widely the values in the referenced field differ. The function returns the standard deviation from the mean of the values in your found set. The standard deviation formula is $n-1$ weighted, following the normal standard deviation. Standard deviation comes in two flavors; to perform a biased or $n-0$ evaluation, select the By Population option.
`Fraction of Total of`	Returns the ratio of a total for which a given record (or set of records, when the field is placed in a subsummary part) is responsible. For example, you can track what percentage of sales is attributable to a given person. The subtotaled option enables you to specify a second field by which to group your data.

When you create a summary field, the Options for Summary Fields dialog opens, prompting you to choose the function you want to use and the field for which you want a summary (see Figure 3.18). Note that only number fields are shown: You cannot sum up text fields or images and the like.

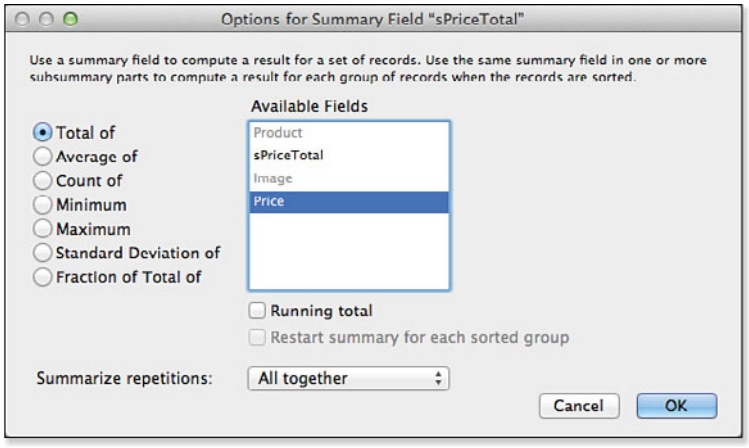


Figure 3.18
Summary fields are useful for performing functions across sets of records.

In Browse mode, a summary field evaluates your found set and displays a result when it is actually visible on a layout. For example, if a summary field is below the visible portion of a layout, it displays information only when the user scrolls to that portion of the window. Summary fields evaluate a found set for a given layout whenever you enter Preview mode, which is the logical behavior for printing—the primary use of Preview mode.

In FileMaker Pro 10 and later, subsummary fields are displayed and updated in Browse mode as well as Preview mode.

Working with Field Options

In addition to establishing fields and assigning data types, you can assign various options to your fields as well. These options range in function from managing auto-entry of default data to validation checks and internal storage settings. They can vary for each field type.

After you name a field and choose its type on the Fields tab of the Manage Database dialog, click Create to save it to your database. You can then opt to apply further behaviors via the Options button on the right. The first set of options is the auto-entry behaviors.

Auto-Entry Field Options

When defining noncalculation fields in FileMaker Pro, you can choose to have data automatically entered into a field as records are created and/or modified. The applications for this can range from

assigning default values to fields, to automatically reformatting data, to inserting values from other fields based on certain trigger events.

In some cases you might also want to prevent users from modifying these auto-generated values, such as when tracking a serial ID or applying a date you don't want adjusted afterward (see Figure 3.19).

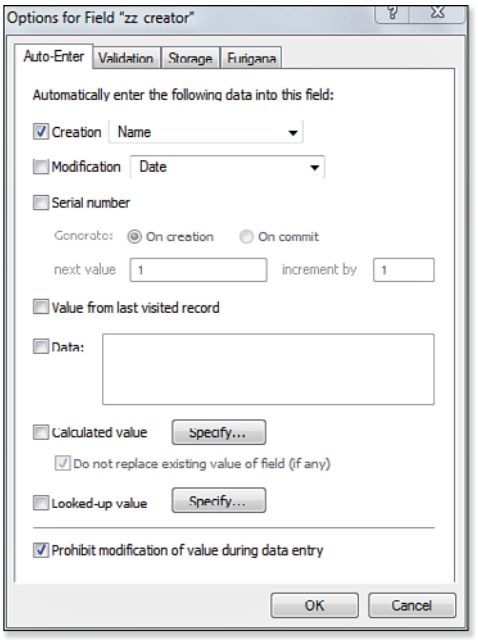


Figure 3.19
FileMaker's auto-entry options enable you to define rules for automatically populating data into fields in your database.

Based on some trigger event, FileMaker inserts auto-entry data into a field. The most common event is record creation: When a user clicks New Record, data can be prepopulated into the record and be accessible for making changes. Each auto-entry function has its own particular rules for what trigger event applies. In addition to new record creation, other trigger events include record modification and modification of a particular field. We cover both cases in the sections that follow.

Creation and Modification

The first two options on the Auto-Enter tab deal with tracking and applying certain values as a record is committed to your database. They behave essentially the same way, with Creation values being applied the first time a record is committed, and Modification values applied thereafter as it is subsequently modified (committed again).

Values that can be automatically entered include the current date, current time, current timestamp, current username (from the General tab of the Preferences dialog under the Edit menu), and current account name (the one entered by the user when logging in to the database).

Serial Number

Using the Serial Number option enables you to auto-enter a number that increments every time a new record is added to the table. Often this number uniquely identifies individual records in a table. The value can be generated either when the record is created or when it is committed. The difference is subtle: In the case of incrementing on creation, your number increments even if a user reverts and effectively cancels a record's creation. The next record will then have skipped a number in your sequence. This doesn't have much of an effect on your database unless your business requires strict tracking of each serial number, even those voided. In those cases, choosing On Commit helps avoid spaces in the sequence.

It is possible to include text characters in addition to a number as the starting value if you want. This enables you to create serial numbers that look something like "a1, a2, a3, a4..." Only the rightmost numeric portion of the value is incremented; the text portion remains unchanged. If you do this, you will want to use a Text field to allow for the alphanumeric combination.

One of the common uses of auto-entry options is in establishing serialized key values or IDs. This is a vital element of your database structure when you're working with more than one table, but we encourage you to adopt some best practices regardless of how complex or simple your plans.

For every table in your database, the first field you should create is a primary key or ID field. These IDs uniquely identify each record in your database. You could go about having the system establish unique IDs automatically in several ways; our recommendation in most cases is to use a serial number set to increment automatically.

To create a serial key field, use the following steps:

1. Define a number field. It is generally advisable to use number-based serial keys, but it is possible to use text as well; the important point is to make certain your keys are unique and users cannot modify them.
2. Go into the Options for that field and select the Serial Number option.
3. Click the Prohibit Modification of Value During Data Entry option at the bottom of the dialog. This is an important step: If you establish unique identifiers that your users can override, you're risking the chance that they'll introduce duplicate IDs.

caution

Both the name and account name can be problematic because users can change them. Knowing how your solution will be used can help you to decide what value to use. The name value is the name of the computer user—obviously not a good choice if your database will be used in a public library. The FileMaker environment typically controls account names, so they can be a better choice as long as people do not share them.

note

If you do not change any of the account settings of a new file, FileMaker establishes two default accounts for you: Guest and Admin. Admin begins with full access to the database; Guest begins with Read-Only access.

note

We can't stress this practice strongly enough. If you ever want to tackle relational data structures, these serial IDs are a vital element in doing so. Further, if you ever export your data to another system or need to interact with other databases, having a key field that uniquely identifies each record in your database guards against confusion or even possible loss of data integrity.

If you need an ID field for a business purpose (SKUs, student IDs, employee IDs from your organization, and so on), we recommend that you create separate fields for such cases. Generally, users should never need to access this serialized ID field, but you can opt to put it on a layout and allow entry in Find mode so that they can search if they choose.

For a full discussion of the use of keys (or match fields*), **see** the discussion in "Working with Keys and Match Fields," **p. 176**.*

Value from Last Visited Record

Used most often as a way to speed data entry when information repeats often for groups of records, the Value from Last Visited Record function copies the value from a prior record into a given new record. Bear in mind that *Visited* means the last record in which you entered data. If you enter data in a record and then view a second record without clicking into and activating a field, a new record obtains its value from the data in the first, edited record.

Data

In the Data field, you can specify literal text for auto-entry. This is frequently used to set default states for field entry. For instance, in an Invoice table, you might have a text field called Status where you want to enter `Not Paid` as a default. As a regular text field, the value is still fully modifiable by a user.

Calculated Value

In addition to establishing a field as a calculation field, where a defined formula determines its value, it is possible to use the Calculated Value option to insert the result of a calculation into a field of another type, including a container field, by using an auto-entry option. Furthermore, if you uncheck the Do Not Replace Existing Value for Field (If Any) option, the result of the calculation formula is entered into the field, overriding any existing value, any time a field referenced by the calculation changes.

Put differently, any field referenced in your calculation statement acts as a trigger: Any time that referenced field updates, the calculation retriggers and puts its result back into the auto-entry field.

*To learn more about advanced calculation functions, including custom functions, **see** Chapter 15, "Advanced Calculation Techniques."*

Looked-Up Value

The Looked-Up Value auto-entry option copies a value from a record in a related table into a field in the current table. Any time the field controlling your association to the related record changes, FileMaker Pro updates the value in the lookup field. For example, if a user enters a postal code into a given record, it's

tip

If there are multiple related records, the value from the first record will be copied. This means you might want to think twice about using looked-up values for relations in which there might be more than one related record.

possible you could have another table auto-populate your City and State fields with the appropriate information.

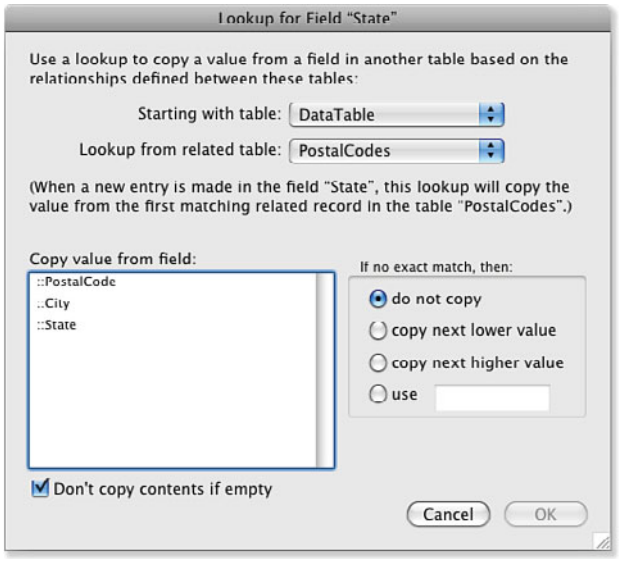
When a user enters a postal code in the record, the City and State fields trigger to pull values from the ZipCodes table. An important fact to keep in mind is that FileMaker *copies* the values from the ZipCodes table. If the source data changes or is deleted, this record remains unmodified until it is retriggered by someone editing the Zip Code field again.

Understanding this distinction is important, especially as we get into indexing later in this chapter. Consider an example for product prices: If you were to build an Orders database that tracks the prices of products, you would want to store the price of each Order line item or product within the order itself. That way if your prices change, your historical orders preserve their original prices. To see how to create a lookup field, refer to Figure 3.20.

note

Take special note that lookup auto-entry functions work just as all auto-entry functions do: They copy or insert information into a field. You are not displaying related information, nor are you controlling content by calculation. Thus, lookup values are not live links to related data. If you were to delete the records in the ZipCodes table in the preceding example, all your people records would remain untouched, thus preserving your city and state data.

Figure 3.20
Often you'll want only exact matches, but in some cases you can use the closest value based on a comparison of the trigger values in your related table.



Remember that any time your match field changes, your lookup refreshes. In this case, the auto-entry function does not act on record creation, but rather on committing.

When you're performing a lookup, it is possible to work with near matches in addition to exact matches. In the case of the postal codes example, obviously you would want only an exact match or you might end up with incorrect data. In a different case, however, you need not be so strict. Consider a scheduling system that automatically finds the closest available appointment: Enter a target date into a field, and the lookup function could return the closest match. Another application

might be a parts database with units of measurement. You may not be able to find a .78 wrench, but a .75 might work. This sort of requirement is easy to meet by using the Copy Next Lower Value setting (or its higher value companion).

How you set up your matching field values is important here. It's easy to compare numbers and come up with the next closest value. If your matching field is text, FileMaker Pro uses ASCII value rules to compare and determine order.

*For further discussion of lookups, **see** Chapter 6, "Working with Multiple Tables."*

Housekeeping Creation and Modification Fields

As a best practice, we also recommend that you create another set of fields in all tables that help track changes. Create a timestamp field and in the Auto-Enter options, choose Creation Timestamp. Define another timestamp field for Modification Timestamp, and text fields for Creation and Modification Account Names.

These four fields tell you exactly when a record was created or modified and by whom (assuming that you assign an account to each individual person using your database). If you ever need to identify problem records for a given day range, time, or account, these fields allow you to do this. We strongly recommend that you add them every time you create a new table. The only downside to following this practice is that additional storage space is required for this data; in this version of FileMaker Pro, this is unlikely to be a concern.

Field Validation

Storing correct and complete information is critical for generating accurate reports; establishing proper, expected conditions on which other functions and calculations are performed; and ensuring overall data integrity. Unfortunately, most data applications suffer from a chronic condition of having humans interacting with them; although some humans are worse than others, no one is perfect. We all make mistakes.

As a user enters data into FileMaker, you might opt to apply one or more validation checks to test that a record meets certain conditions before allowing the user to commit it to your system. This task can be as simple as ensuring that a field isn't empty or as complex as making sure that an invoice doesn't contain multiple entries for the same product. To review the various validation options available, see Figure 3.21.

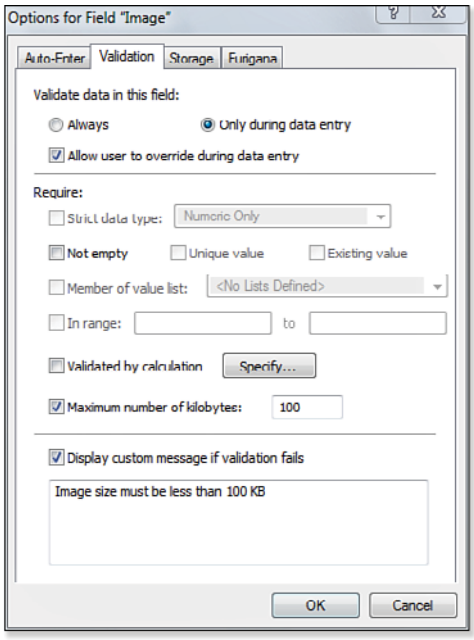
tip

Using FileMaker Pro's capability to import tables allows you to create a boilerplate new table, complete with a primary key serial ID, four housekeeping fields, and whatever other standard fields you want to define. Whenever you need to add a table to your database, import from the boilerplate rather than having to re-create these standard fields. If you are using FileMaker Pro Advanced, you can use the Copy and Paste commands for fields or a table containing these fields. If you are copying and pasting a serial number field, you should know that it starts at the next serial number beyond what it had used before. This means that after copying and pasting a serial number field, you might want to reset it to 1 as described previously in the Serial Numbers section.

note

Some of the validation rules shown in Figure 3.21 are not available for all types of fields; others vary. For example, as you see in Figure 3.21 you can specify the maximum size for an inserted file in a container field; for a text field, you can specify the maximum number of characters.

Figure 3.21
You can set validation rules for the database fields.



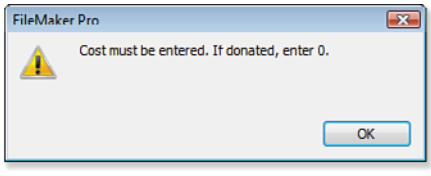
This example demonstrates a common approach to ensuring proper maintenance of your primary keys. This might be overkill if you've enabled the Prohibit Modification of Value During Data Entry option on the Auto-Enter tab, but on the chance that a developer turns that option off for some reason or that users import records into your database, this is a handy bit of insurance.

Importing records can circumvent your carefully designed field validation rules. For a full discussion, ***see*** *Chapter 22, "Importing Data into FileMaker Pro."*

Validation Conditions and Failure

Field validation simply tests whether one or more conditions, as defined in your Validation dialog, are false. If all validation tests are true, FileMaker does not interrupt or prompt the user for action. Figure 3.22 shows an example of what your users might see when validation fails.

Figure 3.22
The OK option appears only if a user has the option to override the validation warning.



In this case, the check box allowing users to override has been left enabled, so they have the option to ignore the warning. When that function is disabled, the field does not allow bad data to be committed, and the system forces users to deal with the problem. They can choose either to revert the field to its previous state or to clear it.

When Validation Occurs

Validation occurs when users manually enter data into the field being validated; some validations happen the moment the user leaves the field, whereas other validations are deferred until the user commits the record. Remember, however, direct entry is not the only way to get information into a field. You can also import records or use various script steps, such as `Set Field()`.

Simply clicking or tabbing into a field does not trigger validation; a change has to be attempted. Keep in mind that validation does not apply in cases in which users modify other, nonvalidated fields of a given record. A given field's validation check occurs only when data in that specific field changes.

At the top of the Validation tab of the Options dialog (refer to Figure 3.21), notice the Always and Only During Data Entry choices. The latter choice tests for validation conditions only when users modify the field in question. If you enable the Always option, validation occurs during scripts and imports as well as during data entry.

If an import process attempts to write invalid data to a field, FileMaker Pro simply ignores the improper entry. The field remains unchanged and does not import your data. You will see a note in the Import Records Summary dialog listing how many errors FileMaker Pro encountered. If you enable the Only During Data Entry option, FileMaker Pro would insert the improper data into your database.

*If you get trapped in a series of validation dialogs, **see** "Validation Traps" in the "Troubleshooting" section at the end of this chapter.*

tip

Some designers make a distinction between validation errors and quality errors. In general, *validation errors* must always be corrected and can never be ignored by users. They are hard-and-fast rules about the data: no nonnumeric data in a numeric field, no missing data, and so forth.

Quality errors (which FileMaker nevertheless implements through the Validation tab of Options) can be overridden. You can construct a quality edit based on a calculation that compares the entered value to the value from the previous record; a difference of more than a certain margin might result in a flag and require the user to confirm the value. This type of quality checking can catch many keying errors.

Storage and Indexing

Field storage and indexing options exist on the Storage tab in your Field Options dialog. These options control how FileMaker Pro indexes each field to speed up searches and sorts and form relationships.

Global Storage

A developer can designate a field to have global storage on the Storage tab of the Field Options dialog. Fields with this option are commonly referred to as *global fields*, and collectively they're usually referred to as *globals*. Global fields exist independently from any specific record in the database and hold one value per user session. Developers often use global fields to establish special relationships or to display unchanging information, such as interface graphics or field labels, across multiple records and layouts.

One vital element to learn is when data is committed and stored for globals: In a single-user environment, any change to a global field is permanent and saved across sessions. In other words, whatever value you last entered into a global will remain the next time you open your database. In the case of a multiuser environment, where a FileMaker Pro solution is hosted on FileMaker Server or via multiuser hosting, global values for each guest default to the value from the last time the database was in single-user mode; any change made to these defaults will be specific only to a given user's session. Other users continue to see the default values, and after the database session is closed, the database reverts to its original, default state.

Using globals is a great way to keep track of certain states of your database. For example, you could use a global field to store which row of a portal was last selected. This field could then be used in scripts or calculation formulas.

note

In the case of globals with values that can change, it is good to initialize them in a startup script. This could mean having pairs of globals. One of them can never change, and the other one can be changed by various users at various times, but you will always reset it in a startup script to the unchanging value. Now that local and global variables are available, it is often the case that they are better suited than global fields for values that might change.

Another common use of globals is for storing system graphics. Establish a container field, set it for global storage, and paste a favorite company logo, a custom button graphic, or any number of elements that you can then control globally in a field rather than having to paste discrete elements on each and every layout.

Beginning with FileMaker 8, a new feature was created in the form of variables defined within scripts (as well as similar variables defined by using the `Let()` function within calculations). These variables exist only in memory and are not permanent fields that you add to your database schema. In the past, developers had to content themselves with using a slew of global fields; starting in FileMaker 8, the need for global fields has dropped considerably. However, you will still encounter them in legacy databases.

*To learn more about variables in FileMaker, **see** Chapter 16, "Advanced Scripting Techniques."*

Repeating Fields

The second section of the Storage tab on the Field Options dialog lets developers allow a field to contain multiple values. Such fields are known as *repeating fields*. On a given layout, the developer can array repetitions either horizontally or vertically, and in scripts can refer to specific repetitions within the field.

Repeating fields can be problematic. They behave just as individual fields might and are really just a shortcut for having to define multiple instances of a given field. It's possible, for example, to have

no values in the first and second repetitions, but to have a value in the third. This sounds convenient and makes sense intuitively, but imagine having to write a script that references that field. How do you know which repetition of the field to reference? Unlike an array in other programming languages, you cannot manipulate a repeating field as a whole. You can reference only one specific repetition at a time.

FileMaker 8 extended the usefulness of repeating fields somewhat by allowing the script step `Set Field` to programmatically reference a repeating instance. You can now open a Specify Calculation dialog to point a script to a specific cell within a repeating field. Note that the same is true for setting variables.

Repeating fields do have their place, however. Sometimes a single data value does have several components. An RGB color, for example, has three values: one for red, one for green, and one for blue. Creating an RGBColor field with three repetitions makes a great deal of sense.

Indexing

Databases store data, of course, but they are also required to perform functions such as searches and sorts with that data. FileMaker Pro, like many databases, can index some of the data in a file to increase the speed at which it performs some of these functions and to enable it to relate data across tables.

An *index* is somewhat like a database within a database. FileMaker Pro can store, along with a specific value in a given field, a list of all the records in which that exact data is used. This enables FileMaker to recall those records quickly, without having to resort to a linear scan of your file. Aptly named, these indexes work just as a book index works: They facilitate finding all the locations in which a given item is used, without searching page by page through the entire book.

To familiarize yourself with the concept, look at a given field's index. Click into a field and select Insert, From Index. If the field is indexable, and has already been indexed, you see a dialog showing all the discrete values indexed for a given field. Just as when selecting from a value list, you can opt to choose from this list rather than type. As you can see in Figure 3.23, FileMaker Pro can create the index based on data values or individual words.

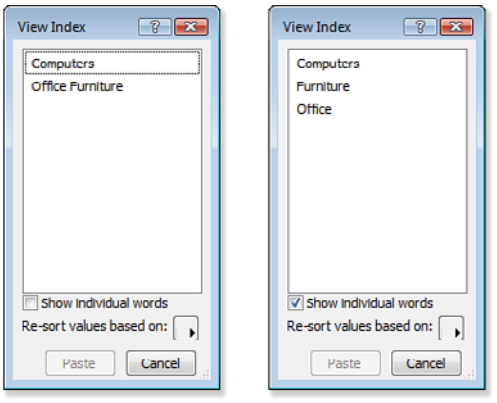


Figure 3.23
You can view index values using From Index in the Insert menu.

Allowing a user to select from an index is only one of the reasons to use indexes in FileMaker. Indexes enable FileMaker Pro to quickly perform find requests, sort records, and establish relationships.

There are two kinds of indexes in FileMaker: value indexes and word indexes. *Value indexes* apply to all field types, with the exception of container or summary fields. *Word indexes* apply only to text fields and are based on a given language or character set. The difference between the two index types, and when either is specifically enabled, lies in their applications.

FileMaker Pro's default indexing setting (found on the Storage tab of the Field Options dialog, displayed in Figure 3.24) is None, with the check box for Automatically Create Indexes As Needed enabled. Most developers, even the more advanced, should find that this setting serves most of their needs.

tip

The Minimal setting produces a value index for text fields or calculation fields returning text. For all indexable fields, the All setting creates a value index, and for text fields or calculation fields returning text, it also produces a word index.

Figure 3.24
FileMaker creates either one type of index or both, depending on how users define and use a field.

A database's schema definition establishes value indexes, as a developer defines fields and builds relationships. In addition, value indexes allow for relationship matches and value lists. If a developer creates a serial ID and joins a relationship via such a field, FileMaker Pro creates a value index for the serial ID field.

Unless a developer explicitly sets a field to generate an index, FileMaker Pro creates word indexes as users are interacting with and using a given database. Word indexes are utilized in text fields for find requests; they are created when a user explicitly chooses Insert, From Index. If a user enters

data in a find request for a field that lacks a word index, FileMaker Pro enables indexing for that field and builds one (unless it's explicitly unindexed or an unindexable calculation).

At this point you might be wondering what all the fuss is about. Why not index every field in a database and be done with it? The downside to indexes is increased file size and the time it takes FileMaker to maintain the indexes. Creating new records, and deleting, importing, and modifying them, all take more time, in addition to the fact that the indexes themselves take up more file space.

Notice that FileMaker doesn't allow you to explicitly control word and value indexes. Value indexes are possible for all field types; word indexes apply only to text fields. The Minimal setting is an available option only for text fields, and when you see it marked, it indicates that at least one of the two indexes exists for the field. There's no straightforward way of determining which index exists. If you explicitly set the field to Minimal, FileMaker creates, on demand, either of the two indexes based on how the field is used. When a user creates a find request including that field, FileMaker creates a word index; if a developer uses the field in a relationship, FileMaker creates a value index.

Only a subset of the fields in your database will ever need to be indexed, and FileMaker's "on demand" approach makes things simple for developers. In general, it's best if a field is indexed only when necessary.

To explore the vagaries of storage and indexing considerations for calculation fields, ***see "Options," p. 223.***

An important point to remember is that some fields are not indexable. This means that they will be slow when used in sorts and find requests, but, most important, you cannot use them to establish relationships. A field is unindexable if it is a calculation based on a related field, a summary field, or a global field, or if it references another unindexed, unstored calculation field.

You can also explicitly make a field unindexable by turning indexing options to None and unchecking the Automatically Create Indexes As Needed setting. In the case of a calculation field, an additional radio button option is available: Do Not Store Calculation Results—Recalculate When Needed. These settings are important to remember; they allow you to force FileMaker to reevaluate and display dynamic information. The `Get (CurrentDate)` function, for example, displays the current date if you have indexing turned off but displays whatever date was last stored with the record if you leave indexing (and storage) turned on.

Furigana

The fourth tab in the Field Options dialog is one that many English-speaking developers will have trouble properly pronouncing, let alone using. Because of the adoption of Unicode support in FileMaker Pro 7, it is now possible to offer Asian-language double-byte language support. As a result, you can now manage Japanese.

Japanese is written using a combination of kanji, complex glyphs borrowed from Chinese that represent complete concepts, and hiragana, a simpler alphabet that represents the phonetic syllables of the language. Furigana is a smaller version of hiragana that acts as a cheat sheet for readers who aren't familiar with a kanji character's reading. The Furigana feature in FileMaker makes it possible to render a kanji-based block of text into its phonetic hiragana equivalent—quite useful when you don't know how to read one of the more than 20,000 kanji characters.

Troubleshooting

Mismatched Data Types

My data isn't sorting properly. Where should I look first to diagnose the problem?

One of the most common bugs you'll run into in FileMaker Pro is confusion stemming from mismatched data types. If your users are entering text data into a field you have defined as numeric, you're bound to get unexpected results, and sorting will be unpredictable. Check your field types when your data appears to be misbehaving.

Mismatched Calculation Results

One of my date calculations looks like an integer. What's going on?

Some of the more subtle extensions of the data type problem are calculation fields. Note that their result is both the determination of their formula and a data type that you set at the bottom of the Specify Calculation dialog. If you're working with dates and return a number, for example, you'll get an entirely valid calculation that will look nothing like "12/25/2013."

Problematic Field Names

My web programmers are complaining about my field names in FileMaker Pro and that I keep changing them. What should I consider when naming fields?

Some other systems are not as flexible as FileMaker Pro; this is especially true for URLs and the Web. Spend some time with Chapters 24, 25, and 26 if you ever plan to publish your database to the Web. FileMaker Pro breeds a certain freedom when it comes to changing field names as the need arises, but you'll send your XSLT programmer into fits every time you do.

Also be sure to check the restrictions of various SQL databases in your organization. If you need to interoperate with them, your field names might have to conform to stricter naming standards.

You'll be safe if you never use spaces or special characters and start each field with a letter of the alphabet or an underscore.

Validation Traps

My field validation seems to have gone haywire. I defined a field that now simply throws up one error message after another. What's the problem?

At the end of the day, field validation is only a helpful bank of sandbags against the storm of human interaction your database will suffer. And as in all aspects of your database, the first and worst human in the mix is the developer. Just as with any programming logic, carefully test your validation conditions. FileMaker Pro can't totally prevent you from illogically conflicting restrictions. For example, if you set a field to be unique and nonempty but also prohibit modification in the auto-entry options, the first record you create will trap your system in an irresolvable conflict.

It's a good idea to leave the Allow User to Override During Data Entry option enabled while you're building a solution and turn it off only after you have completely tested the field in question.

Re-creating Indexes

I am getting find errors returned for valid requests. What has happened? What should I do?

These errors can be a symptom of a corrupted index. In Manage Databases, go to the Fields tab and select Options, then the Storage tab. This is the place where you manage indexes. Note the settings; then click the None check box and turn off Automatically Create Indexes as Needed. Close the various dialogs until you are back in FileMaker itself. If you want to be absolutely safe, quit FileMaker, restart it, and then reopen the database. You will then have no indexes on the field in question. Go back to Manage Databases, through the Fields tab, Options button, and the Storage tab. Turn indexing back on using the settings that you noted. The index is re-created and should be correct.

FileMaker Extra: Indexing in FileMaker

One of the more significant changes beginning in FileMaker 7 revolves around indexing. In prior versions, indexing was restricted to 60 characters total, broken into blocks of up to 20-character words. Relationships had to be built around match fields or keys that were relatively short and generally nondescriptive. This fact is one reason we generally advocate using simple serial numbers for indexing purposes. It's rare that you'd need more than 20 digits to serialize the records in a data table.

FileMaker can index words up to approximately 100 characters. It can index text fields to a total of 800 characters, and numbers up to 400 digits. The limits to indexing have been effectively removed.

What this means to developers is that you can now use far more complex concatenated key combinations (ironically, there will be less of that in FileMaker, given that data can be related across multiple tables), use longer alphanumeric keys, or, as suggested earlier, introduce descriptive elements to keys.

This bit of history is particularly useful if you are working with older FileMaker solutions. Even though you are looking at a database in FileMaker 12, it may have had its roots in a much earlier version. Workarounds with indexes often survive, leaving mysterious remnants for you to try to figure out.

4

WORKING WITH LAYOUTS

What's a Layout?

In the preceding chapter, we discussed how to define fields for holding the data you want to store in your database. In this chapter, we discuss the tools at your disposal for creating user interfaces to manage that data.

You use layouts to create user interfaces in FileMaker Pro. A *layout* is a collection of graphical objects that a user interacts with to view and modify data. These objects include fields, buttons, static text blocks, graphic elements (such as lines or rectangles), images, and even a Web Viewer object that can dynamically display a web page. FileMaker Pro contains a rich set of tools for manipulating these objects, allowing you to create attractive and functional interfaces for your users easily.

note

Beginning with FileMaker Pro 12, *themes* are part of the developer's toolkit. They consist of coordinated settings for graphical elements so that your default background, buttons, and other interface elements all work together. Themes are discussed in Chapter 14, "Advanced Interface Techniques."

You can create many kinds of layouts in FileMaker. Form layouts are useful for data entry; often form layouts are shown as forms using the View as Form command from the View menu with a single record's form shown at a time. List layouts are often used for reports and can contain summary parts; such layouts are often viewed in Preview mode, from which they are printed. List layouts are also created to be viewed as a list so that many records can be viewed at a time. Such list layouts generally have navigation tools to allow you to switch with a single mouse click to a Form view with more details for an individual record, and Form views often contain a button to let you

switch to a multirecord list view. In addition, Table layouts display multiple records in a spreadsheet-like view.

Some layouts might be designed for system administrators to clean up data quickly with a minimum of interface elements and a maximum of data. Such layouts might also allow access to fields otherwise not shown on a layout or that do not allow entry on standard layouts. Still others can serve as user navigation tools and contain no data at all.

One of the aspects that makes FileMaker different from database products such as MySQL, SQLite, and databases such as Oracle, DB2, and SQL Server is that the layouts themselves are stored in the database file, along with data, scripts, access privileges, and other elements of application logic. Every FileMaker Pro file must have at least one layout; there is no practical limit to the number of layouts a file can contain. It's neither unheard of nor undesirable to have anywhere from a dozen to a hundred or more layouts in a file.

Layouts are created and managed in Layout mode. Almost all the material in this chapter deals with tools and functions that require you to be in Layout mode to access them, but for simplicity and brevity, we do not specifically mention that fact in conjunction with every tool and tip. When you are in Layout mode, you can manipulate the objects in the layout. In addition, you can show the *Inspector*—a floating window that lets you view and change settings for the currently selected layout element. The layout itself in Layout mode and the Inspector together are sometimes referred to as the *design surface*.

The introduction of triggers in FileMaker Pro 10 dramatically increased the options available to you as an interface designer. For more information, ***see*** *Chapter 17, "Working with FileMaker Triggers."*

There are two ways to enter and exit Layout mode:

- Choose View, Layout Mode, or simply press (Command-L) [Ctrl+L]. To leave Layout mode, use the similar commands to choose another mode.
- At the right of the Status toolbar, click the Edit Layout button. That button turns to Exit Layout when you are in Layout mode; clicking it returns you to where you were.

Layouts provide the interface tools with which people interact to use your FileMaker solution, but the user interface is more than fields, buttons, and graphical elements. The user interface includes scripts that run in response to user actions as well as the triggers that cause other scripts to run in response to events that might or might not be caused by user actions. The user interface also includes validations and messages that you specify when you create and manage fields in your database. Together, all these make up the user interface. However, layouts are the most basic part of that interface. This chapter provides an introduction to the design and creation of FileMaker Pro layouts.

When you create a database and its fields, FileMaker Pro creates a default layout for you; each field is added to the layout as you create it, along with a label providing the field's name. If you update the fields in the database while a layout based on the table you are updating is open, the fields you create will be added to that layout. If you are updating the database while the current layout

is based on another table, no fields are added. Similarly, if you remove fields from tables, they are removed from all layouts on which they appear.

Because FileMaker Pro is creating a default layout for you and managing it as you add and delete fields, you can use that default layout without any customization or design features. For example, consider creating a new database called Tiny Task Management.

Tiny Task Management is shown in Figure 4.1. It has three fields:

- **DueDate** is a date field.
- **Name** is the task name.
- **zzID** is a unique number that is auto-entered using a field option.

In Figure 4.1, a single record has been created in the database, and values have been entered for those fields. Figure 4.1 shows the new layout in Table view.

note

This is a companion to Small Task Management, which you will build in Chapter 6, "Working with Multiple Tables," and Chapter 7, "Working with Relationships."

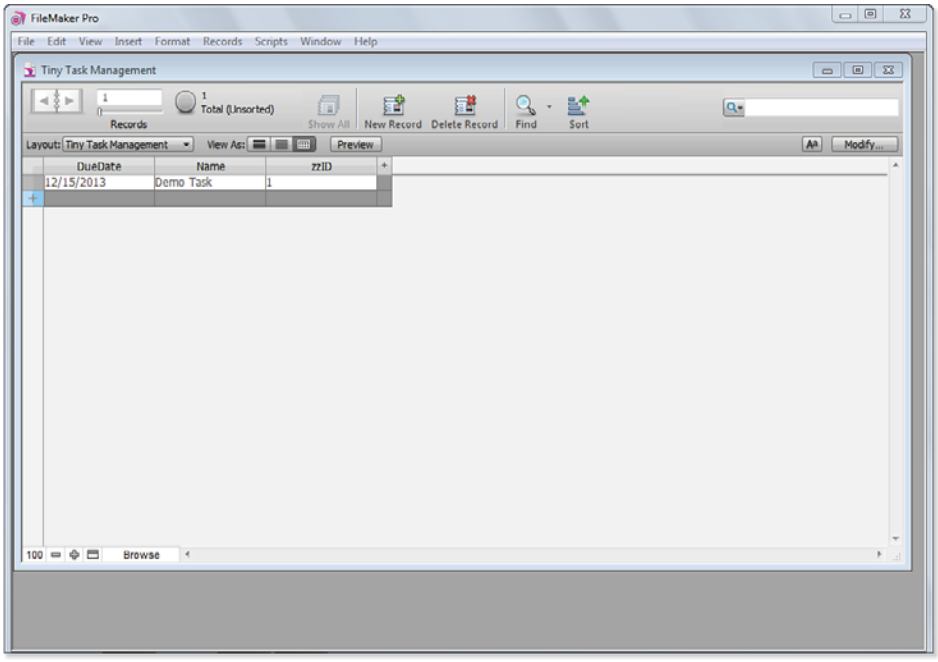


Figure 4.1 FileMaker Pro creates a default layout as you build your database.

In Figure 4.2, you see the same default layout with data entered as it appears in Form view.

In Figure 4.3, you see a second record added to the database and the layout is shown in List view. (Because List view shows multiple records at the same time, you don't get the full effect until you add a second record.)

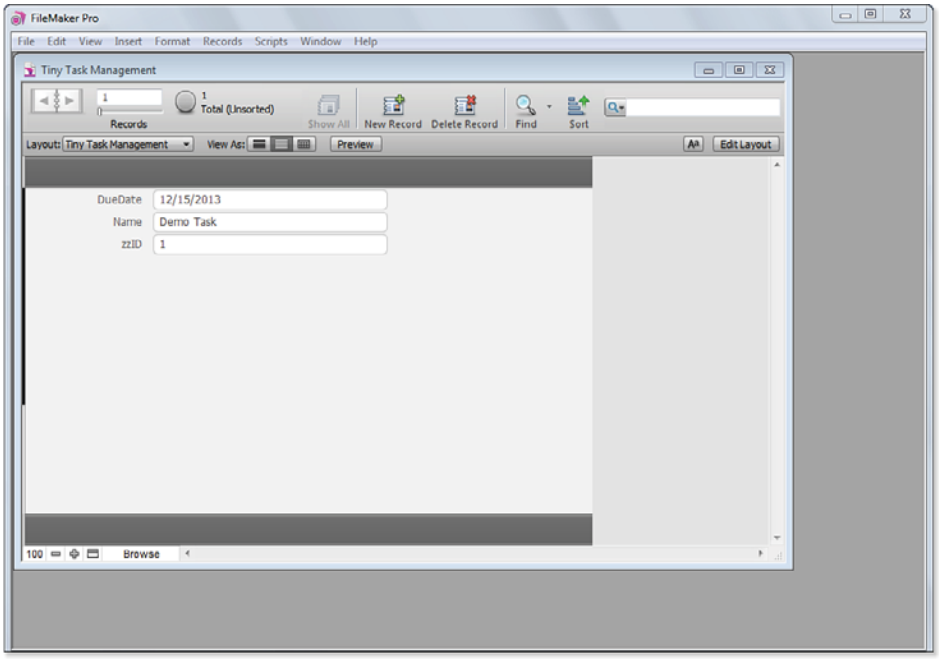


Figure 4.2
The default layout can be shown in Form view.

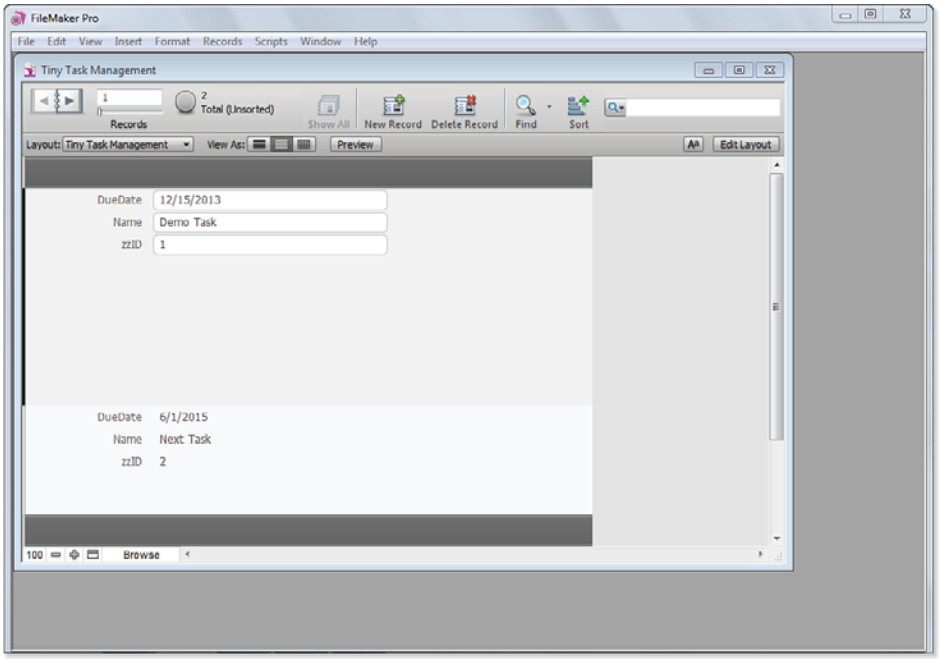


Figure 4.3
With a second record added, you can view the layout in List view.

The default layouts that FileMaker Pro builds for you are functional without any further work on your part, so you can get started doing whatever it is you want to do with your database. However, by building your own layouts, you can make your solution much more attractive and easy to use. In addition, without a great deal of trouble, you can make your solution usable on desktops, iPhones, and iPads as well as with Instant Web Publishing. Figure 4.4 shows the Contacts Starter Solution, which comes with an attractive and very usable interface. This chapter shows you the basics of moving from the layout shown in Figures 4.1, 4.2, and 4.3 to your own version of the layout shown in Figure 4.4.

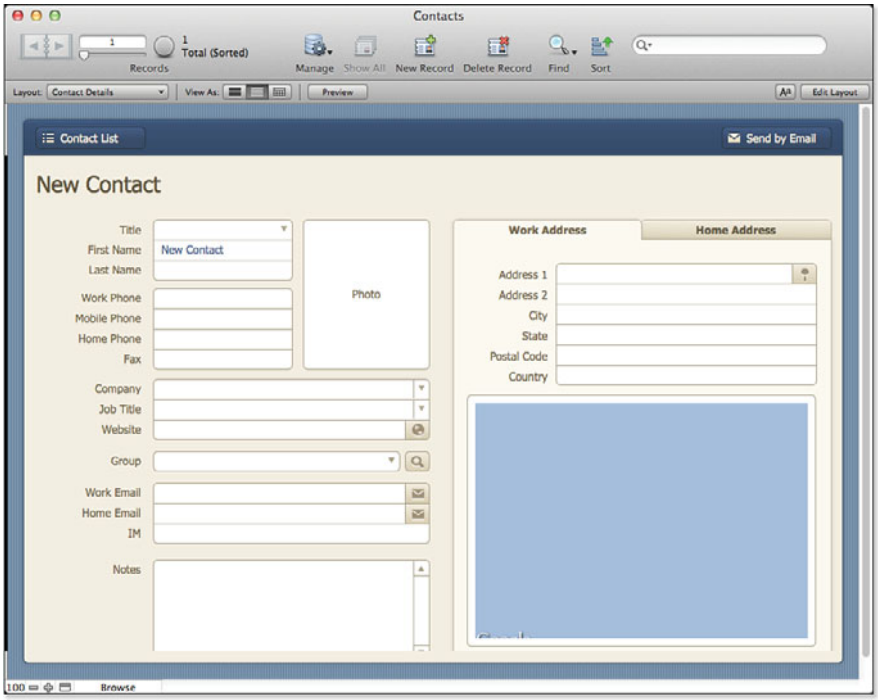


Figure 4.4
The Contacts Starter Solution has an attractive layout.

One of the major tools for improving the look of your layouts is themes. You can change the look of the interface consistently just by selecting one theme or another. You can customize the themes, but if you start by leaving them "as is," you can switch easily from one theme to another. If you then want to customize a theme, you can add specific refinements.

You can learn more about themes in Chapter 14, "Advanced Interface Techniques."

Using Multiple Layouts Automatically

As noted previously, layouts are just one part of the user interface: Scripts and triggers are other important components, along with options for your database field. This section provides a quick overview of how those tools can work together to help implement your user interface.

> *This is only an overview. Details of scripts are provided in Chapter 9, "Getting Started with Scripting," and Chapter 16, "Advanced Scripting Techniques." Triggers are discussed in Chapter 17, "Working with FileMaker Triggers." Field options were discussed in Chapter 3, "Defining and Working with Fields and Tables." Finally, Chapter 14 continues the discussion in this chapter in more detail.*

The Contacts Starter Solution comes with three sets of layouts, as you can see in Figure 4.5.

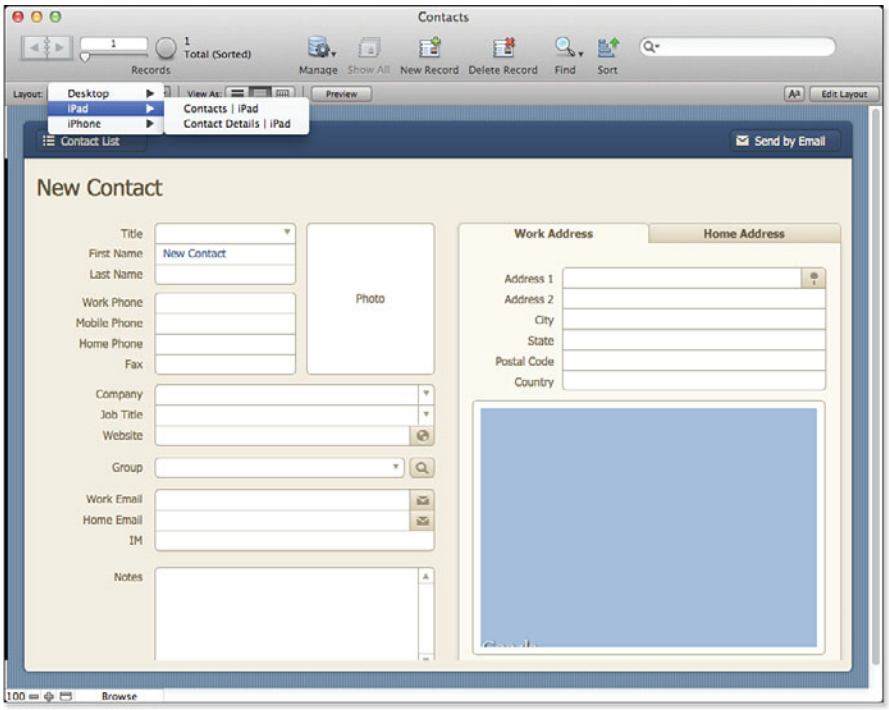


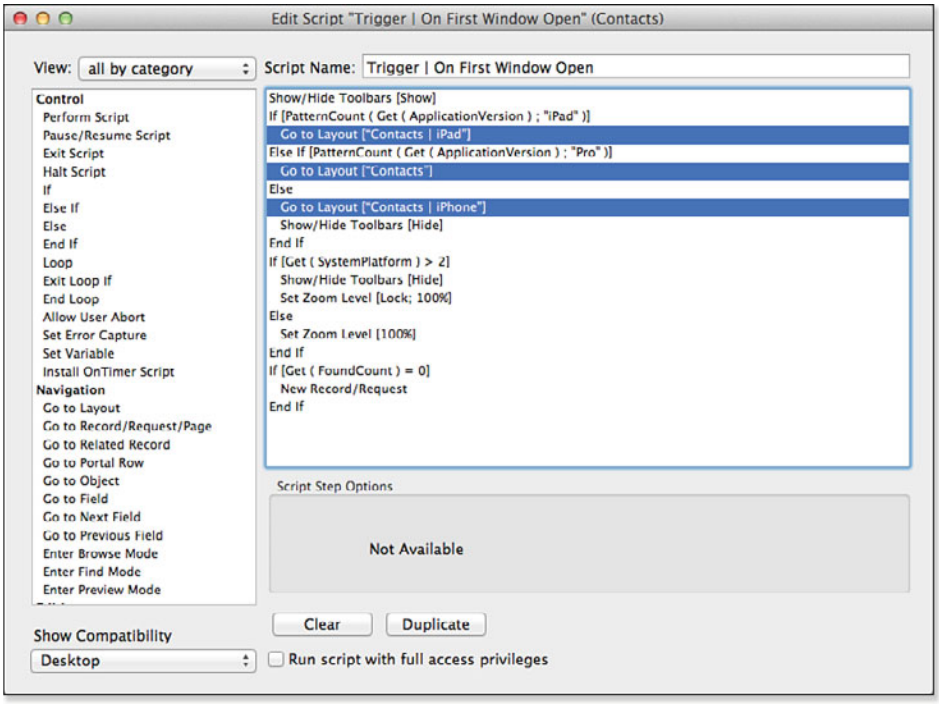
Figure 4.5
Contacts has multiple layouts.

There are layouts designed to be seen in Form view as well as layouts designed to show multiple records in List view; each pair of layouts is implemented for the desktop, iPhone, and iPad, making a grand total of six layouts. The user might not actually ever see the multitude of layouts because Contacts—like most Starter Solutions, including those that you create—automatically manages them.

Here is how you set up automatic layout management. Don't worry that the details of layouts, triggers, and scripting haven't been discussed yet: This process is used in almost the same form over and over again.

1. The first step is to design and create your layouts (which is the purpose of this chapter).
2. Next, create a script that tests to see on which device your solution is running. This script is used almost without change for many solutions, as you see in Figure 4.6. The only changes you normally make involve the highlighted lines: You identify the layouts you want to go to for each device.

Figure 4.6
Create the startup script.



3. Select File, File Options to open the dialog shown in Figure 4.7. In the Open tab, you may specify an automatic login (Admin is the default for many databases including the Starter Solutions). Then, automatically switch to the opening layout.
4. In the same dialog, select the Script Triggers tab, as shown in Figure 4.8. Set the OnFirstWindowOpen trigger to your startup script.

Having selected the right layout, you have made everything ready for people to use the solution. The buttons on each layout let users move from the Detail layout to the List layout, and vice versa. For the iPad Detail layout, the button moves to the iPad List layout; the same process occurs for iPhone and desktop. Once you have set the initial layout, you are home free (that is, provided you implement the layouts!).

note

If you have a sharp eye, you might notice that the Startup Screen layout shown in Figure 4.7 was not shown in the pop-up menu of layouts shown in Figure 4.5. You can control whether layouts appear in that menu.

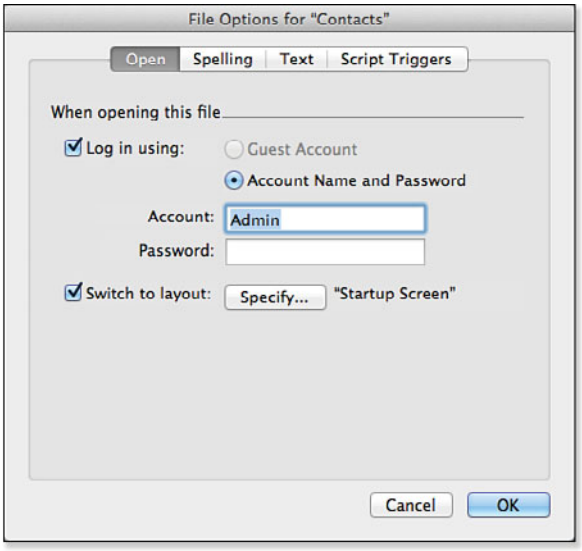


Figure 4.7
Set the opening layout.

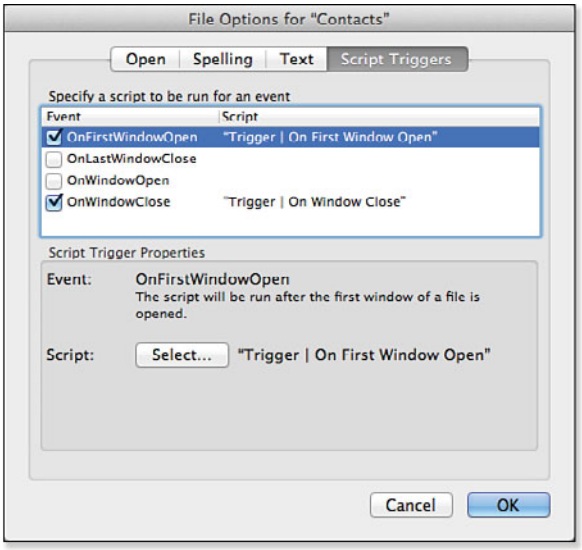


Figure 4.8
Run the startup script.

Creating and Managing Layouts

Creating and managing layouts are among the most important tasks required of a FileMaker developer. They are also among the most intuitive. Nonetheless, you need to know numerous subtle facts and details. We encourage you to have a test file open as you go through the following sections so that you can try things first-hand.

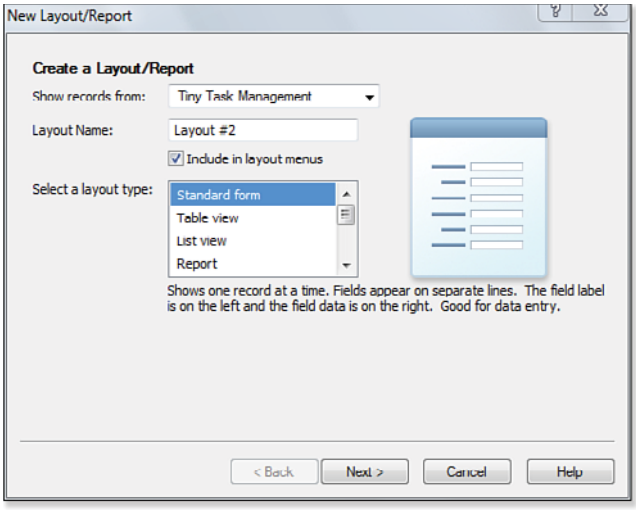
tip

Beginning in FileMaker Pro 12, you can select a theme for your layout. As you build it—either by hand or by using an assistant—the theme attributes are used for fields and other graphical elements. Often, that is sufficient to produce very attractive and usable layouts. If you want to further customize your layouts, consider copying them and customizing the copy so that you can easily revert to the basic theme.

Creating a New Layout

You can create new layouts anytime you want while in Layout mode simply by choosing Layouts, New Layout/Report, or by pressing (Command-N) [Ctrl+N]. You are taken to a setup assistant that can help you configure a layout according to one of a handful of types of common layout designs. Figure 4.9 shows the first screen of the New Layout/Report assistant, where you specify a name for the layout and choose a layout type. You also specify a layout's context here; the next section covers that topic.

Figure 4.9
This is the first screen of the assistant for creating new layouts.



You can create the following seven types of layouts. As you make your selection on the first screen, schematic diagrams of the various layouts appear at the right of the dialog.

- **Standard Form**—Useful for data-entry layouts, Standard Form generates a basic form view layout with a set of fields you specify. You can select a theme for the layout as well; themes specify the default background color and text styles that will be applied to the layout.

- **List View Report**—As its name implies, this type is used for creating basic reports. It has the data organized in columns, with the name of each field at the top of the appropriate column (you can change these if you want). Each row of the layout is a single record's data. Optionally you can constrain the layout to the width of a page. You also can select which fields/columns appear in the layout. If you don't already have the necessary summary fields in your database, you can create them right from within the assistant.
- **Report**—This layout presents data in rows and columns, but it adds the capability to group and summarize the data for the entire report as well as for subsummaries within it. For example, you can group data based on an individual's postal code in a subsummary, by state or province as a larger subsummary containing multiple postal codes, and then with a grand total across all the data. If you have declared summary fields in the table, not only will the data be grouped, but the relevant summary fields can be calculated at each subsummary level and for the grand total.
- **Table View**—Table View gives you a spreadsheet-like view of your data. When you select Table View as your layout type, you can select the fields you want to appear on your new layout. They are displayed in Table view according to your selected theme. Table view is quite useful for behind-the-scenes data manipulation, but it might not be suitable as an end-user interface.
- **Labels**—This type of layout is used for printing sheets of labels in standard or custom sizes. The Layout assistant prompts you to specify the type of label you will be using; Avery 5160/5260 are the labels used most commonly. If you don't see your label type listed, you can specify custom measurements.

*For some tips that come in handy for working with label layouts, **see "Multicolumn Layouts," p. 126.***

- **Vertical Labels**—This layout is used for Asian and full-width characters that will be rotated so that the labels can be used vertically.
- **Envelope**—You are prompted to select fields you want to use for the address portion of the envelope. The default layout is sized for standard business envelopes. You might have to do some testing and tweaking of the layout to get things just right for your envelopes and printer.
- **Blank Layout**—Choosing Blank Layout gives you just that: a completely blank layout that you can manipulate any way you want, free of assistants.

We do not discuss all the screens of the New Layout/Report assistant here; they're quite intuitive, even for new developers. Besides, if you are new to FileMaker, nothing beats spending an hour just playing around with the assistant to see firsthand what the various configuration options do for you. You won't cause harm to any existing layouts by doing so, nor can you hurt the database even if you mess up the creation of a new layout.

tip

No tool is available for importing layouts from one file to another. If you ever need to do this, the best method is to set up a new, blank layout with layout parts sized the same as the source layout. Then copy all the objects from the source file and paste them into the new file. Fields, buttons, and portals must be re-specified to point to their correct referents, but at least all your formatting will be retained.

After you create a layout, you can modify it and turn it into whatever you need it to be. Much of the remainder of this chapter is devoted to the tools at your disposal to do just that.

Within a file, you can duplicate layouts by choosing Layouts, Duplicate Layout. Often, this is a preferred method for creating new layouts, even if they end up looking significantly different from the original. All part sizes, graphic elements, and formatting options are retained, so modifying as necessary with these as a starting point is usually much faster than creating new layouts from scratch.

tip

Create a template layout for yourself that has examples of all the necessary bits and pieces specified (portals, fields, field labels), along with color squares and grid lines. Then you can simply duplicate your template when you need to create a new layout, and you'll be well on your way to a finished product. In a large project, you might create several template layouts: one for form views, another for layout views, and so forth. If you base your templates on themes (beginning in FileMaker Pro 12), you will dramatically streamline your design process.

Layout Context

Every layout is linked to a *table occurrence* from the Relationships Graph. You specify this on the first screen of the Layout assistant in the Show Fields From area; a similar area exists in the Layout Setup dialog, described later. Many layouts can be linked to a particular table occurrence, but each layout must be tied to one, and only one, table occurrence.

*For more information on table occurrences, **see** "Adding a Table Occurrence to the Relationships Graph," **p. 195**.*

The reason layouts need to be associated with table occurrences is that, in a multitable file, FileMaker needs some way to know which records to display in a given layout. In the old days, when FileMaker allowed only one table per file, it was always clear that layouts in file X should display records from table X. Now, layouts in file X can be configured to display records from table A, B, or C. The context of a layout is determined by the table occurrence to which it is tied. Context, in turn, determines the table from which the layout will show records, and establishes the reference point for other types of operations, such as displaying data from related tables and evaluating calculations that reference related tables.

note

The concept of layouts being tied to table occurrences can be a bit confusing. See "Determining Which Records Will Be Displayed on a Layout" in the "Troubleshooting" section at the end of this chapter.

You might wonder why layouts need to be associated with table occurrences and not source tables themselves. If you were only concerned with displaying records from the source table, you wouldn't have to worry about table occurrences. But layouts must also be able to contain records from related tables (that is, portals), and relationships are built between table occurrences, not between source tables. Having a layout linked to a table occurrence makes it unambiguous which context FileMaker should use to access related records.

Consider this situation in terms of perspective. To view any data within your solution, your user needs a starting point, or perspective, and an endpoint. For example, you might be looking from Company Detail through a portal to Employees related to that company

record. The associated table occurrence tied to a given layout serves as a user's starting point, and any related data is viewed from that table occurrence's perspective on the Relationships Graph.

If you're unfamiliar with relational data modeling or how to display related data in FileMaker, ***see*** *Chapter 5, "Relational Database Design," and Chapter 6, "Working with Multiple Tables."*

When you define a new layout, the first prompt of the New Layout/Report assistant lets you specify where to show records from. The options in the pick list are all the table occurrences from the current file's Relationships Graph. If you want to use a table from another file, just add that table to the Relationships Graph (you may have to also add an external data source for the file if you have not done so already).

Working with relationships is the subject of the first three chapters of Part II, "Developing Solutions with FileMaker." However, looking at a preview of the general issue is worthwhile because it illustrates the key role that layouts play. Figure 4.10 shows a FileMaker Relationships Graph for an ordering and quoting system. (Don't worry about the details; they will be explained in Part II.)

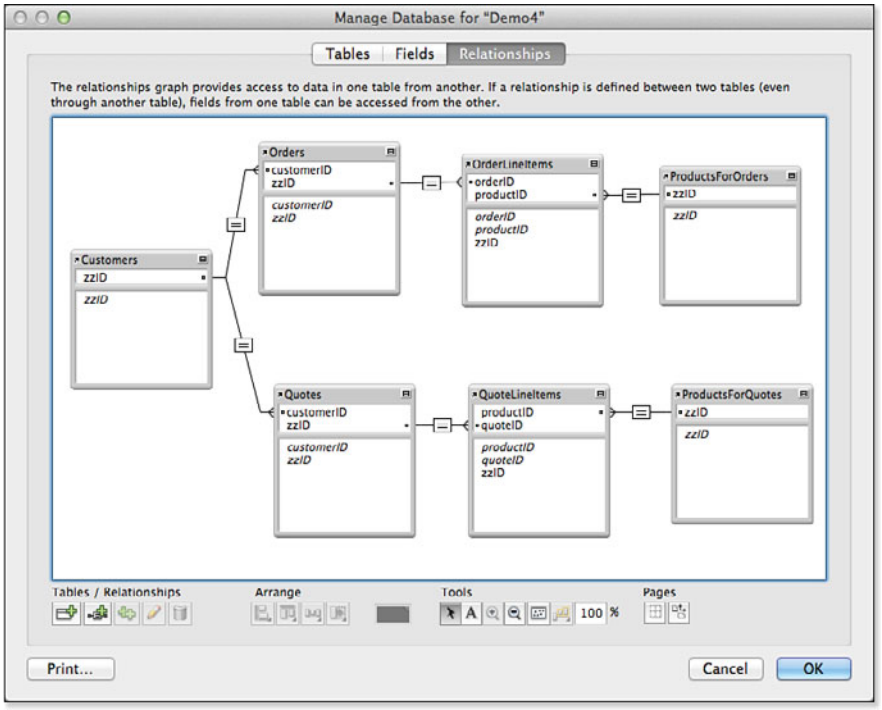


Figure 4.10
Create an ordering and quoting system.

Customers can be related to quotes or orders. Each quote or order can have multiple line items, and each line item is related to a given product. As will be explained in Part II, FileMaker requires that there be a unique path between any two related table occurrences in the Relationships Graph. Thus, the Products table appears as two different table occurrences: ProductsForOrders and

ProductsForQuotes. It is the same table, but there must be two separate occurrences because otherwise there would be two paths between Customers and Products, and that is not allowed.

You can solve this problem in another way, as shown in Figure 4.11. Here, the duplicated table occurrences are for the orders and quotes. Note that there is no direct link from customers to any product table occurrence. For this structure to work, it must be tied to layouts. One layout can display customers; in a portal within that layout or in another layout, it can display orders and quotes for that customer. But when you want to look inside an order or quote, a button or some other interface element takes you to a layout that is based on either the Orders or Quotes table occurrence that is related to its own line items and thence to a single Products table occurrence. Many people would agree that this structure—although it contains nine table occurrences rather than seven—is simpler than the one shown in Figure 4.10. The layouts supporting both structures are going to be similar; they just are based on different table occurrences. The moral of this story is that layouts help you structure your database; they are not just for the interface.

For the implications of context for scripting, ***see*** *"Script Context and Internal Navigation,"* ***p. 268.***

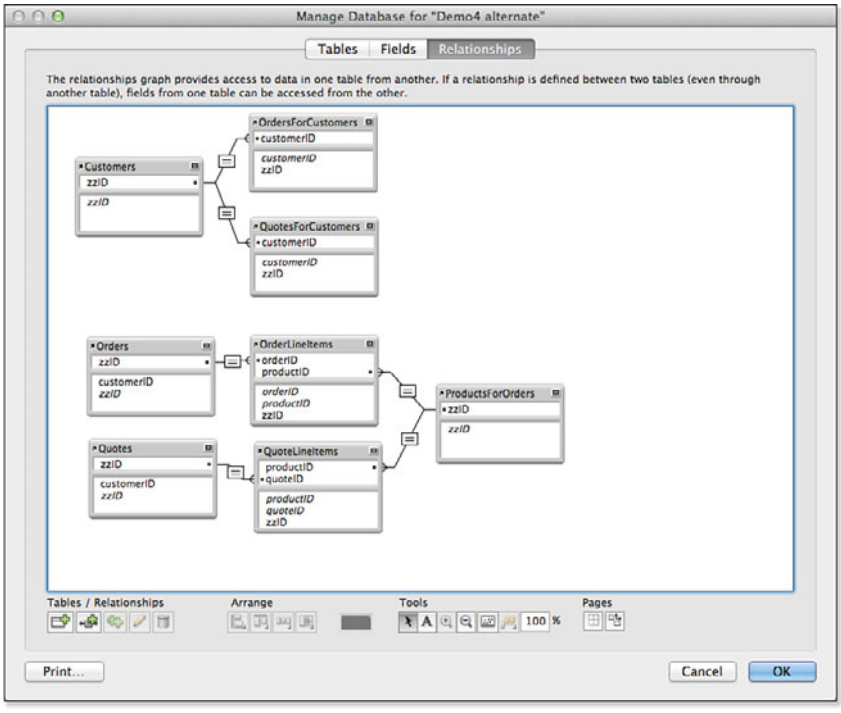


Figure 4.11
Create an alternative structure.

Layout Setup

The Layout Setup dialog, accessed from the Layouts menu, allows you to edit many of the fundamental characteristics of a layout, such as the name of the layout, its context, and how it can be viewed (see Figure 4.12).

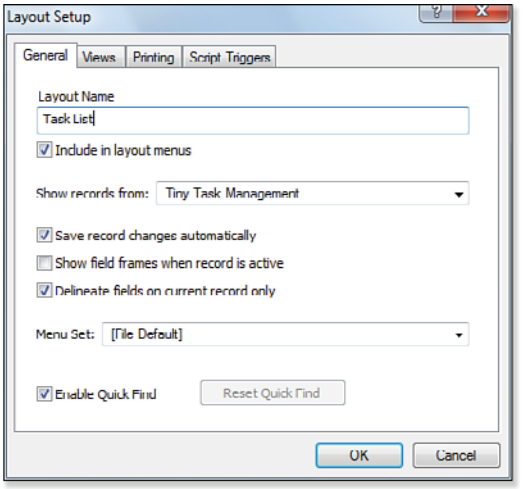


Figure 4.12
The Layout Setup dialog allows you to change the name of a layout and its context, for example.

In the Layout Setup dialog, you name the layout and select the table (actually the table occurrence from the Relationships Graph) that is the base table for the layout. Data from other tables can appear in the layout, but there is always one, and only one, base table for each layout.

Check boxes also let you save record changes automatically if you want to. These changes are the changes to the layout itself (not to its data). If you are doing a lot of layout modification, you can choose to have your changes automatically saved without a prompt. When major development is over, you might want to turn this option back on so that the relatively rare updates to layouts are reviewed more carefully.

The Show Field Frames When Record Is Active check box influences the layout's behavior in Browse mode. It is a good idea to select the same setting for this check box for all layouts in a given solution so that users know what to expect.

note

Naming your layout as well as the objects within it is just as important as naming fields and tables. The "Layout Naming Conventions" section, later in this chapter, provides guidance in these areas. The "Hiding and Reordering Layouts" section, also later in this chapter, gives you some tips on how to handle the Include in Layout Menus check box.

*You can provide custom menu sets for each layout. To do so, **see** Chapter 14, "Advanced Interface Techniques."*

View Options

Every layout you develop could potentially be viewed in three ways: as a form, as a list, or as a table. A user with access to standard menu commands can use the View menu in Browse mode to switch among them. When you navigate to a layout, you will see it in whatever state it was last saved, so bear in mind that switching from layout to layout might change the view setting as well.

The differences among the three view types are quite straightforward:

- **View as Form**—This view type always shows one record at a time. Any header and footer parts are not fixed on the layout; if the layout has a long body, a user might have to scroll to see the footer. If the body part is short, the last part on the layout expands to fill the empty space in the window. If you are using scripts to navigate among layouts, you can adjust the window automatically to fit the form so that there is no empty space. For some long forms, such as legal contracts, you might have to split the form into two separate layouts.
- **View as List**—With View as List, the height of the layout body part and the height of the window determine the number of records displayed. If more records are present in the found set than can be displayed onscreen, the vertical scroll bar enables users to see additional records. Any header and footer parts are fixed onscreen at all times, even when a user scrolls to see additional records. Subsummary parts are never visible in Browse mode with View as List. If fields are placed in the header or footer parts, they take their values from the currently active record. Any modification to a field in the header or footer part likewise affects the currently active record.
- **View as Table**—In Table view, all the fields placed in the layout's body are presented in a spreadsheet-like grid. The fields' top-to-bottom position on the layout determines their initial order. That is, the first column is the topmost field on the layout. No nonfield elements (for example, buttons, text, or graphics) from the body of the layout are rendered in Table view. Field formatting (for example, color, font, and font size) is honored, however. The column headers conform to the format of the first field. Other properties of the Table view can be specified under the Views tab of the Layout Setup dialog. As you can see in Figure 4.13, you can specify whether header and footer parts should be visible and whether columns can be sorted, reordered, and resized. You can also specify row heights here.

Using the Views tab of the Layout Setup dialog, you can disable user access to certain view types. Although usually not necessary, this can be a good precaution to take to keep adventurous users on the right track. Accessing an inappropriate view type is likely not going to cause much harm, but it certainly can confuse users.

tip

In developing a custom solution, you can provide scripted buttons to switch from layout view to layout view. If you do so, you can use the scripts to enforce certain standards, such as always viewing a certain layout with a certain view (at least to start). By using a combination of hiding the Status Toolbar and a custom menu set, you can even hide the standard FileMaker Pro commands that allow users to switch from view to view without using the controlled scripts that will limit the experience and flexibility but provide more consistency. This capability is particularly useful when your users are not adept at using FileMaker Pro.

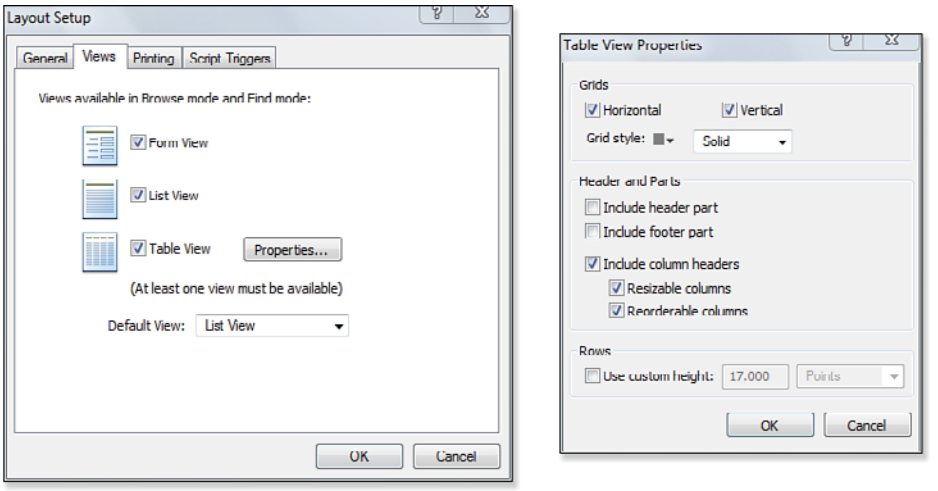


Figure 4.13
You can alter the look and functionality of the Table view by using the Table View Properties dialog.

Multicolumn Layouts

When printing labels and certain types of reports, you might want to present your data in multiple columns. You can specify the number of columns to display on the Printing tab of the Layout Setup dialog; this is shown in Figure 4.14.

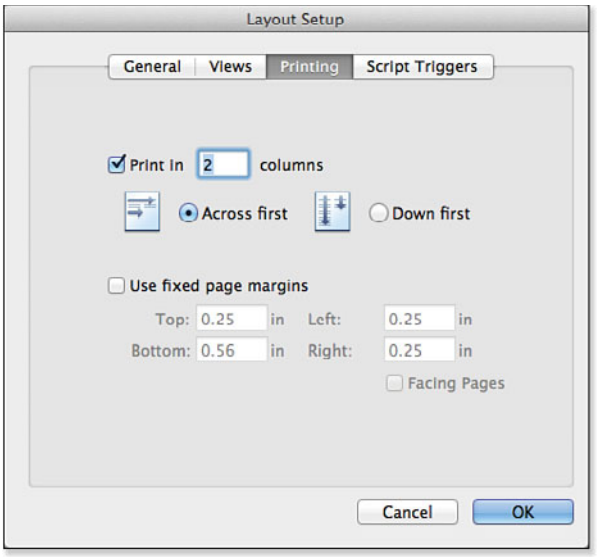


Figure 4.14
You can customize the print settings for a particular layout on the Printing tab of the Layout Setup dialog.

In Layout mode, dashed vertical lines represent the boundaries between columns. FileMaker grays out columns other than the first; the idea is that you need to place any objects you want displayed in the first column, and these objects replicate to the other columns as necessary. Figure 4.15 shows an example of a three-column layout: Labels layout from Contacts. (These happen to be Avery 5160 labels: the New Layout/Report label section provides dozens of standard labels from Avery as well as DYMO.) Notice that the header and footer part are not divided into columns. This means that if you want headers to appear above the second and third columns, you have to add them explicitly.

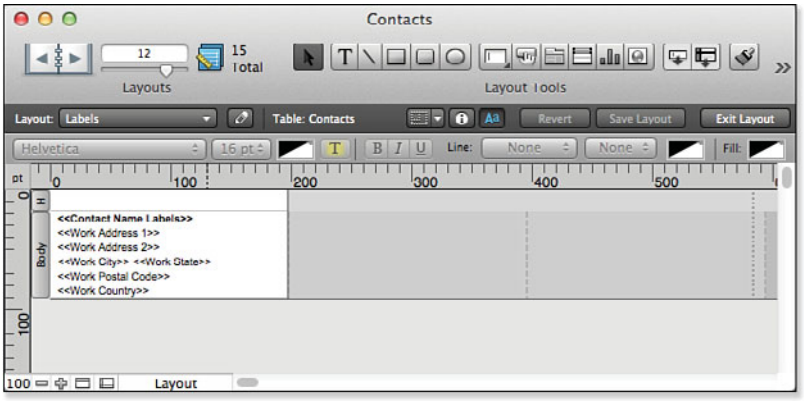


Figure 4.15
This example shows a layout for the three-column label layout from the Contacts Starter Solution.

You can use subsummary parts and leading and trailing grand summaries on multicolumn layouts, but they behave slightly differently depending on whether you choose to display data using the Across First or Down First option. If you choose the Down First option, any summary parts are also columnar. On the other hand, if you choose the Across First option, summary parts span the full width of the layout, just as the header and footer parts do.

Subsummary parts are covered in depth in "Working with Parts," p. 130.

The effects of a multicolumn layout can be viewed only in Preview mode. In Browse mode, the user sees only a single column of data.

tip

It's not possible to have columns of differing widths; every column is the same width as the first one. You can manually adjust the column width by clicking the dashed divider between the first and second columns and dragging left or right as appropriate.

Hiding and Reordering Layouts

In Browse mode, layouts can be designated to be either accessible or inaccessible via the layout pull-down menu in the Status Area. If a layout is accessible, users can see it and navigate to it at will, assuming that the Status Area is visible and/or accessible. If the layout is inaccessible, users can navigate to it only by running a script that takes them there. In Layout mode, all layouts are accessible.

Typically, layouts are set to be inaccessible when you need to prevent users from manually navigating to a layout. For instance, you might have report layouts or find screens that require certain preparation before they become useful. There might be unanticipated and/or undesired results if a user is able to bypass the scripts you created and navigate directly to a layout.

The option to have a layout be accessible or not is the Include in Layout Menus check box on the first screen of the New Layout/Report assistant; it can also be set through the Layout Setup dialog. The Manage Layouts dialog, shown in Figure 4.16, also has a check box on each line that can be toggled to change a layout from visible to hidden, and vice versa. Using this method is the quickest way to hide or show a number of layouts at once. Open the Manage Layouts dialog from File, Manage, Layouts, or as shown in Figure 4.17.

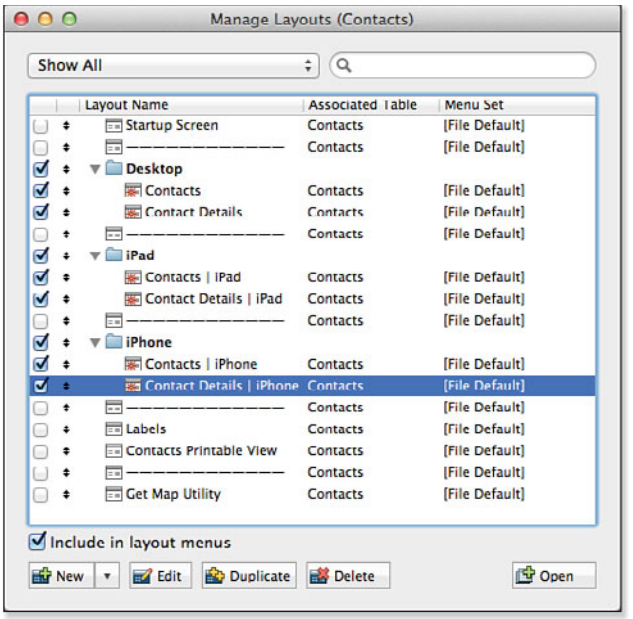


Figure 4.16
Use the Manage Layouts dialog to set the accessibility and order of layouts.

The Manage Layouts dialog also enables you to change the order in which layouts appear in the Layout pop-up list. You can use the double-arrowed selection tool to move a layout up or down in the order. On OS X, you can accomplish the same thing by selecting a line and pressing (Command) [Ctrl] and the up or down arrow.

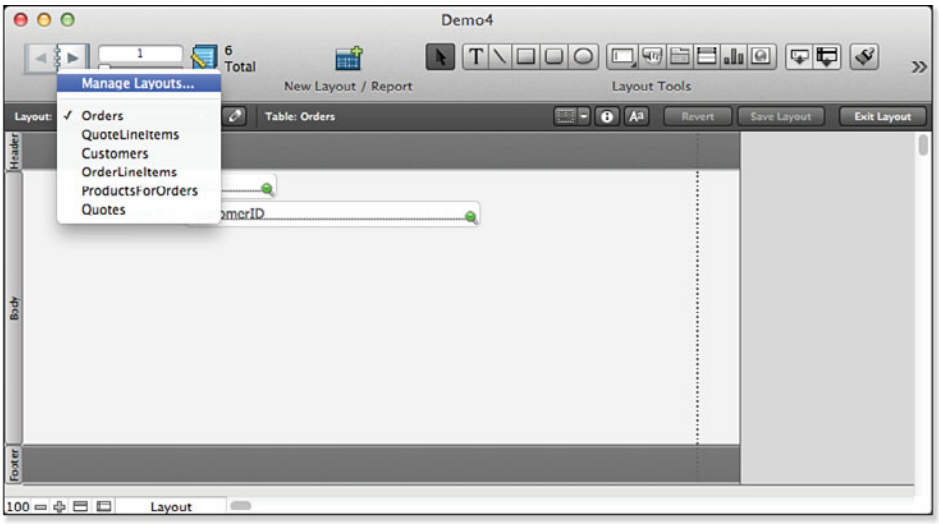


Figure 4.17 Open Manage Layouts from the Layouts pop-up menu in the Status toolbar in Layout mode.

Layout Naming Conventions

You have a great deal of flexibility in how you name layouts. Layout names do not have to be unique and can be up to 100 characters long. They can include numbers, symbols, spaces, and pretty much anything else you want to use. Although flexibility is a good thing, we suggest that you follow a few guidelines:

- If a layout has the potential for access via ODBC or Custom Web Publishing, you should avoid symbols, punctuation, and spaces in its name.
- Only the first characters of a layout name are visible in the layout selection pop-up menu in the Status toolbar. The full name is visible when a user clicks the pop-up, but it can be helpful to use short, unique names for easy identification.
- Try to use names that are somewhat descriptive of the purpose of the layout. Names such as List and Layout #3 might not convey much meaning to users, or future developers, for that matter.
- In a multitable file, consider having the base table name as part of the layout name. For instance, Customer:Data Entry and Data Entry (Customer) might be good names if you need to differentiate among multiple data entry layouts. Note that this may conflict with the first guideline about not having symbols, punctuation, or spaces in the layout name.
- You can use a z or zz prefix for internally used layouts. You can omit them from the Layout menu or place them at the bottom (using the Set Layout Order command from the Layouts menu in Layout mode). You can also use a distinctive theme in the Layout assistant. Any or all of these will make it clear when you are working in a layout that users will not normally see.

- Finally, if you use a single hyphen (-) as a layout name, this appears in the Layout pop-up list as a divider. Users can't select divider layouts, which merely serve to help organize what might otherwise be an unwieldy list. Typically, such layouts would be left completely blank, but this isn't a requirement.

As you see in Figure 4.16, you can construct a sophisticated naming guideline—many of the Starter Solutions use this type of naming convention so that the base table as well as the device are included in the name. In practice, the issues involved in naming layouts might work themselves out quite easily. If you are building a solution with multiple layouts, you might very well build navigation tools into the layouts (commonly at the top of a layout). This means that when you go to a layout using a script, you automatically adjust the window, or in the case of a layout designed for printing, you automatically go into Preview mode. In that case, you can name layouts from the developer's perspective and hide them from users in the Layouts menu.

The single-hyphen naming trick works in other areas of FileMaker as well, such as within value lists.

Working with Parts

Parts make up layouts. Depending on your objectives, your layout might contain header and footer parts, a body part, one or more subsummary parts, and maybe even a leading or trailing grand summary. Every layout must contain at least one part. Briefly, the purpose and some characteristics of each type of part are as listed here:

- **Title Header**—Title headers are used when you need a header on the first page that differs from the header on subsequent pages of a multipage report. In Form view, a user can view a title header while in Browse mode, but not in List or Table view.
- **Header**—Objects in the header part appear at the top of each page of a multipage report, except the first page when a title header is present. A header part remains fixed onscreen in List and Table views, even when a user scrolls to see additional records. Data in fields placed in a header part can be edited; fields in a header part always display data from the currently active record.
- **Leading Grand Summary**—Typically used on report layouts, a leading grand summary appears between the header and any subsummary or body parts. Summary fields placed in this part aggregate across the entire found set.

*For more information about using summary fields and summary parts to create reports, **see "Using Summarized Reports," p. 295**.*

- **Body**—The body part is used to display data from a single record. A data-entry layout often consists of nothing other than a body part. Almost every layout you create will have a body part.
- **Subsummary**—Subsummary parts are used primarily for displaying subtotals on reports. For a subsummary to display properly, the found set must be sorted by the same field as that on which the subsummary is based. Subsummaries can be placed either above or below the body part, depending on whether you want the subtotals displayed before or after the data they summarize.

- **Trailing Grand Summary**—Similar to a leading grand summary, a trailing grand summary is typically found on report layouts and is used to display aggregate summaries. When printed, the trailing grand summary report appears directly following the body part and any trailing subsummaries.
- **Footer**—Objects in the footer appear on every page of a multipage printout, except on the first page when a title footer is present. In List view, the footer remains fixed on the layout when a user scrolls through records.
- **Title Footer**—A title footer part is used when you want to display a different footer on the first page of a multipage printout.

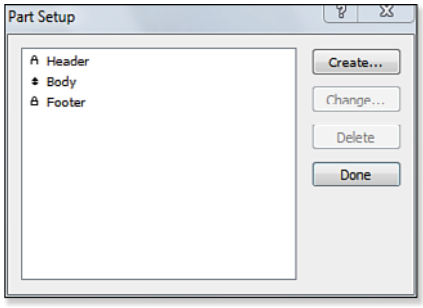
You can save time creating complex layouts by beginning with a layout that might never see the light of day. Create a layout with a body part and as many subsummaries as you might ever need. Then duplicate the layout and remove parts that you don't need. For example, you can delete the body part from a duplicate and—voilà—you have a summary layout. Likewise, you can delete some of or all the subsummaries to produce a detail report. To finish up, you can even delete the original layout.

Adding and Ordering Parts

There are two ways to add parts to a layout. The first is by clicking and dragging the Part button in the Status toolbar to the point where you want the new part to appear. FileMaker prompts you to select a part type when you release the mouse. Although this method is convenient, we discourage using it to add new parts. New parts, except when added to the bottom of the layout, always come at the expense of existing parts. That is, if you have a 50-pixel header followed by a 200-pixel body, and you attempt to add a subsummary between these parts, the body part shrinks by the size of the subsummary part. Moreover, fields that were in the body part might now be part of the subsummary part.

The other option for adding new parts, which we prefer in almost every circumstance, is to use the Part Setup dialog (shown in Figure 4.18), which can be found under the Layouts menu. When you add parts with this tool, it's not at the expense of any existing part; the total height of the layout increases.

Figure 4.18
You can add, edit, delete, and reorder the parts on a layout from the Part Setup dialog.



The Part Setup dialog can also be used to reorder, edit, and delete parts. The only types of parts that can be reordered are the body and subsummary parts. To reorder them, click the arrow in front of the part name and drag it to the desired position. Other part types appear with a lock in front of them, indicating that they are fixed in a certain order by definition.

You can delete a part from a layout either by selecting it from the Part Setup dialog and clicking Delete or by clicking the part label while in Layout mode and pressing the Backspace or Delete key on your keyboard. Either way, when you delete a part, you also delete any objects contained in that part.

Formatting a Part

You can configure a few attributes of parts directly from Layout mode itself. First, you can set a background color and/or fill pattern for a part by clicking the part label and then selecting a color and/or fill pattern. (Control-clicking) [right-clicking] the part label similarly pulls up a contextual menu with access to these attributes.

You can achieve much the same effect simply by drawing a large rectangle on the layout, sending it to the back, and locking it. Setting a background color for the part is preferred because the color extends to the right and downward if the user expands the window beyond the boundaries of your rectangle.

You can also change a part's size. To do this, simply click the dividing line between two parts and drag either up or down. When making a part smaller, you can remove whitespace from the part, but you are prevented from dragging through any objects in the part. Any expansion of a part increases the overall size of the part.

Holding down (Option) [Alt] as you resize a part changes the rules slightly. First, any expansion or contraction comes at the benefit or expense of the neighboring part; the overall height of the layout remains the same (except, of course, when enlarging the last part on the layout). Also, you can "run over" objects this way; an object that was in one part might end up belonging to another part after you resize things. An object that ends up straddling two (or more) parts belongs to the part that contains its upper-left corner.

tip

For users with monitors set to higher resolutions than your database was designed for, consider adding a footer with a background color different from the body part so that users can visually see where the layout ends and size their windows appropriately. Alternatively, use buttons and scripts for layout navigation so that you can automatically adjust the window to the layout.

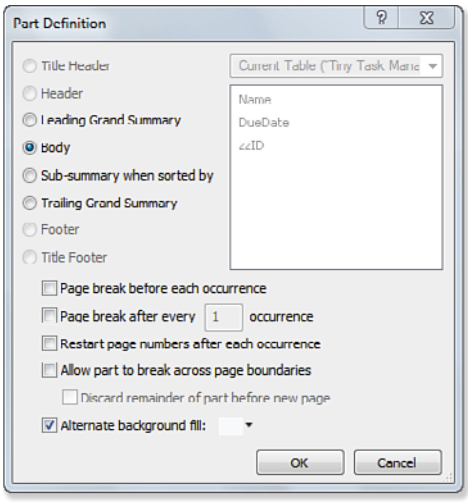
The Object Info palette can also be used to see and set a part's length. This is the best way to set part lengths precisely, especially when you're trying to duplicate complex layouts from one file to another. Click the part label to display that part's data in the Size palette.

*For more information about the Object Info palette, **see** "Positioning Objects on a Layout," **p. 145**.*

Part Definition

Beyond the size and background color of a part, some part attributes can be set only in the Part Definition dialog, shown in Figure 4.19. You can get to this dialog either by using the Part Setup dialog (by clicking Create or Change) or by double-clicking the part label itself.

Figure 4.19
The Part Definition dialog specifies a part's type and attributes.



The radio buttons on the left side of this dialog indicate the type of part. You can change the type of a part simply by selecting a different radio button. If a type is grayed out, it means you already have a part of that type. The only part type for which you can have multiples is subsummary.

The fields on the right side of the dialog apply only to subsummary parts. When you make a subsummary part, you must specify which field will act as the break field for the summary. The break field doesn't actually have to appear in that part, but the found set must be sorted by the break field for the subsummary part to appear on a report.

For more information on break fields and subsummary reports, ***see*** *"Using Summarized Reports,"* ***p. 295****.*

At the bottom of the dialog are some options for configuring page breaks and page numbers. In subsummary reports, you'll often want each new subsection to start on a new page. To do this, you edit the part definition of the subsummary part to include the Page Break Before Each Occurrence option. As you would expect, a page break precedes only each occurrence after the first one.

You can also opt to use the Alternate Background Fill feature. This option is available only on body parts. Any color and/or fill that you

tip

If there is a script to display the layout, as is the case if you have used the assistant, that script contains a Sort step. When you add or change a subsummary part, make it a habit to immediately go to the script and change the sort so that it reflects the new or changed sorting order.

specify is used as the background for every other record. It alternates with any background color specified for the part itself. A slight shading of alternate rows on a report often makes it easier to read.

Working with the Layout Status Toolbar

You have seen how the Status toolbar works and can be customized in Browse and Find modes; now you will see how it can be used in Layout mode. As noted previously, the Status toolbar has a slightly different appearance in Windows and OS X. Furthermore, the methods of customizing it differ across the two platforms. Nevertheless, its buttons and other interface objects behave the same way on the two platforms.

The Status toolbar can be customized in many ways. This section describes the components of the Status toolbar; they may or may not be in the Status toolbar you are looking at, and even if they are, they may be in different places. Figure 4.20 shows the Status toolbar with a minimal configuration.

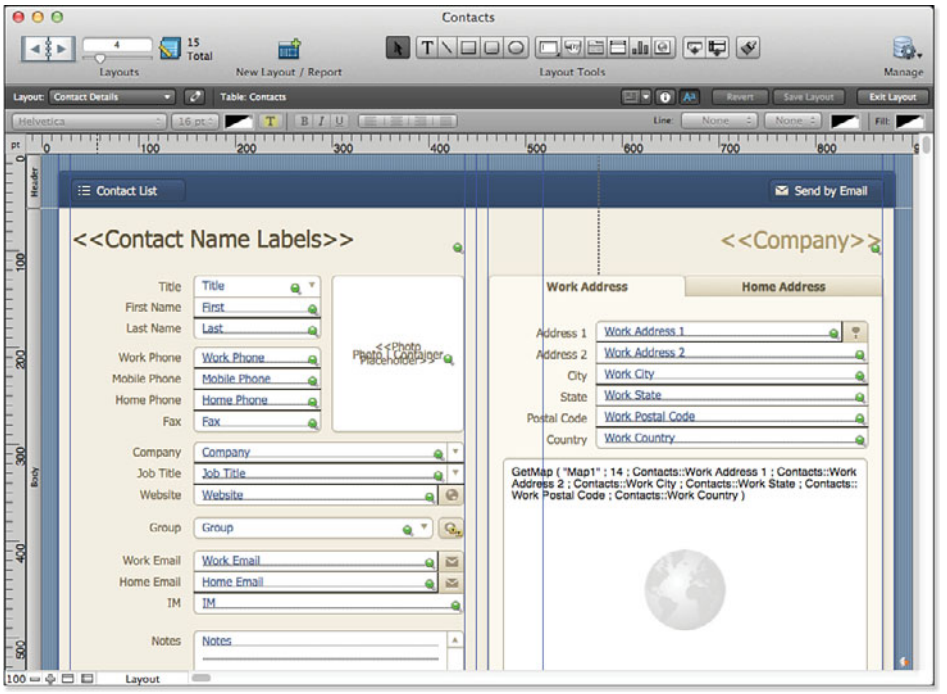


Figure 4.20 The Status toolbar varies in its appearance.

Using the Layout Bar

Below the main area of the Status toolbar is an uncustomizable Layout bar. From left to right, here are the tools available:

- The Layout pop-up menu lets you select a layout by name. The first item in the pop-up menu is Manage Layouts. It opens the Manage Layouts dialog shown previously in Figure 4.16.

- The pencil icon opens the Layout Setup dialog shown previously in Figure 4.12.
- Next, you see the base table occurrence for the layout. (Set it in the General Pane of Layout Setup.)
- Three buttons show or hide palettes and the formatting bar. Further details about these palettes and the formatting bar are provided later in this chapter. The first shows or hides the device dimension stencils, the next shows or hides the Inspector, and the third one shows or hides the formatting bar.
- The last three buttons let you save the layout. The first lets you revert to the last saved version. The second lets you explicitly save your layout changes. The last one lets you exit Layout mode.

Using the Customizable Status Toolbar Tool Groups

On OS X, the View, Customize Status Toolbar command opens the sheet shown in Figure 4.21. You can drag items into or out of the toolbar; you can also rearrange them. In Windows, the same command opens the text-based dialog shown in Chapter 2, "Using FileMaker Pro." Also in Windows, you can use the small arrow at the right of the Status toolbar to open a hierarchical set of menus in which you can select what tools are used in the Status toolbar. This menu hierarchy is shown in Figure 4.22.

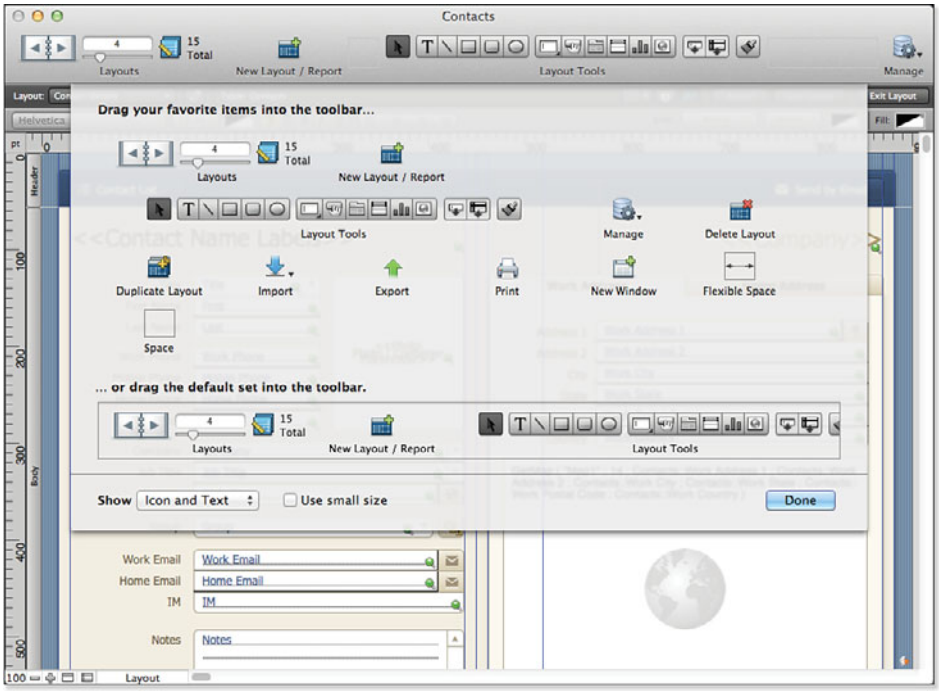


Figure 4.21 Customize the Status toolbar on OS X.

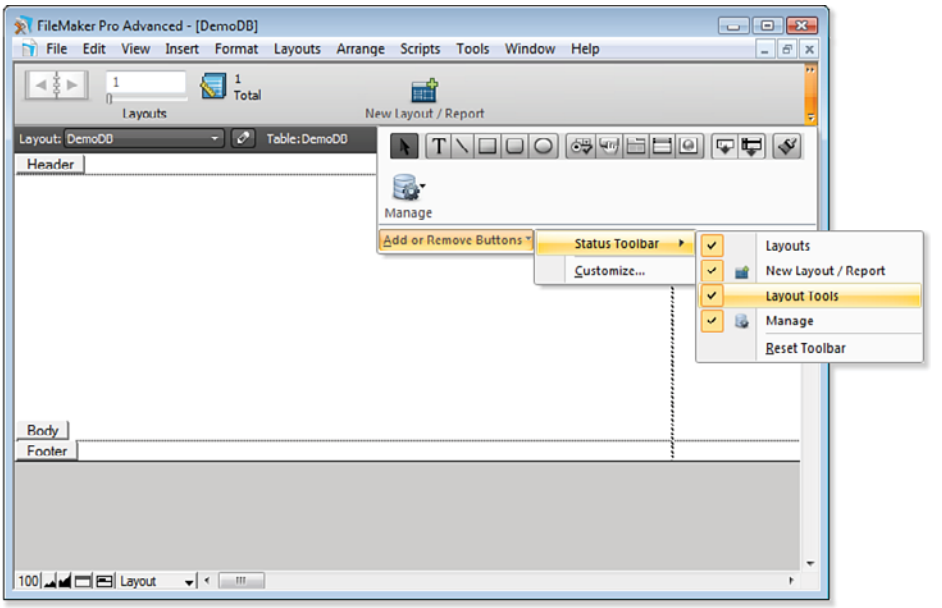


Figure 4.22 Customize the Status toolbar in Windows.

The tools described here are actually groups of tools: You cannot split them apart. They move into and out of the Status toolbar as a group. Many of the tools correspond to menu commands.

*For more information on how to customize the Status toolbar, **see** Chapter 2, "Using FileMaker Pro."*

tip

Because all the tools are shown in the Customize Status Toolbar sheet shown on OS X in Figure 4.21, earlier in this chapter, you might want to refer to that figure throughout this section.

Layouts Group

The Layouts tools are shown installed in the Status toolbar at the top left of Figure 4.20. At the left, the book icon lets you move to the next and previous layouts (provided that there is a next or previous). The slider lets you quickly drag through your layouts, and the text field above the slider shows the current layout number; you also can type in a number to go to that layout. Finally, you will see a layout-editing icon that displays the number of layouts in the file.

Layout Tools Group

The Layout tools collection consists of five groups of tools. Their use is described in the following section.

- The pointer lets you select an item in the layout. Holding down the Shift key lets you select multiple items. You can then drag the items around the layout and resize or reshape them by using the handles at their corners and sides.

- Next come five tools that let you draw a text box, a line, a rectangle, a rounded rectangle, or an oval. Holding down the Shift key while you draw constrains the object vertically and horizontally; holding down the Option key makes it regular (that is, a rectangle becomes a square, and an oval becomes a circle).
- The next six tools let you draw specific FileMaker objects. From left to right, they let you draw fields or controls, buttons, tab controls, portals, charts, and Web Viewers. Select a tool and draw the object on the layout. After you have finished drawing, a setup dialog opens for you to specify further details. These setup dialogs are described in the sections related to using fields, buttons, tab controls, portals, and Web Viewers.

For more information on adding fields to a layout, see "Working with Fields," p. 149.

- The next pair of tools lets you add a field or part to a layout. Drag a tool onto the layout and select its field or its part. Note that these tools are dragged into the layout; the previous five let you draw on the layout.
- Finally, the Format Painter tool lets you copy formatting attributes from one object to another.

Using the Status Toolbar Items

Many of the Status toolbar's customizable items correspond to individual commands or to a group of commands:

- New Layout/Report (Layouts, New Layout/Report).
- Delete Layout (Layouts, Delete Layout).
- Duplicate Layout (Layouts, Duplicate Layout).
- Import (File, Import command and its subcommands).
- Export (File, Export command and its subcommands)
- New Window (Windows, New Window).
- Manage. This is the File, Manage set of submenus: Database, Accounts & Privileges, Value Lists, Layouts, Scripts, External Data Sources, Custom Functions (FileMaker Pro Advanced), and Custom Menus (FileMaker Pro Advanced).

Using the Inspector

Beginning with FileMaker Pro 11, the Inspector was added to Layout mode. You find inspectors in a number of modern programs like FileMaker. In the initial versions of graphical user interfaces, there was a routine process for users: Select something, and then choose a menu command to act on it. Over time, the menu commands multiplied, and inspectors were devised to manage the multitude of commands for selected objects. The Inspector is shown in Figure 4.23.

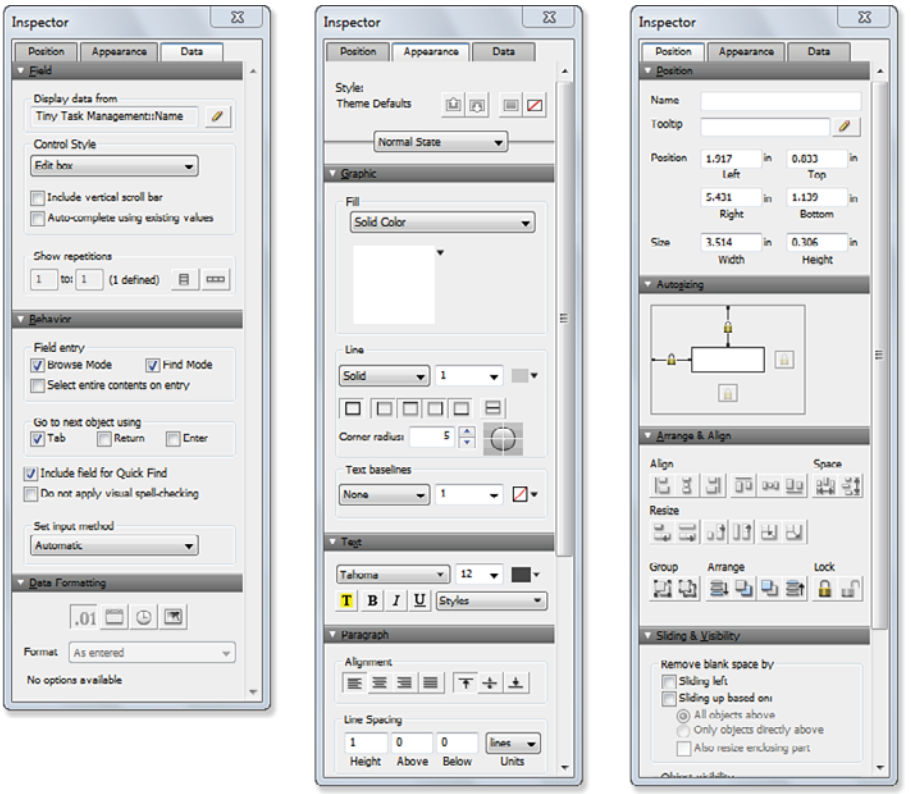


Figure 4.23
The Inspector lets you work with layout objects.

The three tabs on the Inspector let you work with the data for a selected object, its position, and its appearance. You open the Inspector with View, Inspector, (Command-I) [Ctrl+I], or by using the Inspector button on the Layout bar. Remember that the data displayed in the Inspector varies depending on what is currently selected.

tip

You can open up to three inspectors at the same time. If your screen is large enough, that means that you can see all three of the tabs at the same time.

Inspecting Data Settings

The data settings are composed of three sections: Field, Behavior, and Data Formatting. All of these are related to the data itself rather than its appearance or position on the layout. It is important to note that you can have more than one field showing the same data element from the database. By judiciously combining the various attributes of fields in the Inspector, you can display the same data in various formats (different numbers of decimals on numeric fields, for example), and in various types of controls (radio buttons as well as edit boxes, for example).

Using Field Settings

At the top of the Inspector, you can specify data settings for a selected field. These settings include the field name and the style of the control that is used to display or edit it. Depending on which type of control you select, some additional settings may be displayed. For example, if you select a radio button or check box, you can specify a value list that is used to support it.

For standard fields where a user will be manually entering and editing data, the Edit Box format is appropriate. The option to include a vertical scroll bar is normally used only when a user is able and/or expected to type multiple lines of text.

The options to format a field as a Drop-Down List, Pop-Up Menu, Checkbox Set, or Radio Button Set require you to specify a value list that provides the content for the selection values.

You can apply a Drop-Down Calendar to a field to help with entering dates, and you can toggle either a drop-down indicator icon or a calendar picker icon for the fields you're working with. The two icons—drop-down indicator and calendar icon—appear only if a field has its right border turned on.

The lower left of the field settings is relevant only for fields defined to allow multiple repetitions. You can hard-code the starting and ending repetitions and specify whether a vertical or horizontal orientation should be used.

tip

As you choose the control style of a field, bear in mind that on mobile devices tappable inputs (radio buttons, checkboxes, drop-down lists, and drop-down calendars) are particularly useful.

Controlling Field Behavior

The behavior settings contain controls for setting when a field is enterable and how a user can exit it.

In this section, you can control whether a user is able to enter a particular field while in Browse or Find mode. Typically, a user should be able to enter a field in both Browse and Find mode. Sometimes, though, you'll want a field to be enterable in only one of these modes. For instance, you might have a field that you don't want users to manually edit but that they might have to use as part of a query. On the other hand, there might be unindexed fields on your layout that, for performance reasons, you don't want users to search on.

You can also use a check box to determine whether a field is used in Quick Find. As described in Chapter 2, "Using FileMaker Pro," Quick Find lets users search from Browse mode. The search term is entered at the right of the Status toolbar, and FileMaker Pro searches all Quick Find fields for the value.

You can also specify visual spell-checking for each individual field in this section. You set the file-wide spell-checking option in the File Options dialog in the File menu; this setting overrides that value on a field-by-field basis.

The other setting in this section is the Go to Next Object Using option. By default, in FileMaker Pro, pressing the Tab key lets users move to the next field on the layout. Developers can also specify the option to allow the Return and/or Enter key to perform this function. This is desirable in some cases to allow rapid data entry and to prevent data-entry mistakes. For instance, by setting a text field to

use the Return key to go to the next field, you prevent users from accidentally adding stray returns at the ends of fields. Obviously, if a user needs to be able to enter carriage returns in a text field—say in a Comments field—you wouldn't set the Return key to go to the next field.

Note that you can also choose the input method that FileMaker uses for this field in Japanese.

caution

Normally, the Enter key serves to commit a record and exit all fields. If you change all your field behavior to have a press of the Enter key move focus to the next field, be aware that users must explicitly click the background of a layout or perform some script or navigation routine to commit record changes.

Managing Data Formatting

At the bottom of the Inspector, you can set the formatting for numbers, dates, times, and images. Note that for container fields beginning in FileMaker 12, you can set optimization. Images such as PNG, JPEG, and BMP, are simply downloaded. If you choose the interactive radio button (PDF, MP3, and the like), you can allow interaction and set an option the start playback automatically.

Inspecting Appearance Settings

These are the standard settings you expect for text and graphics: fonts, alignments, colors, and the like. You can also set styles for objects.

Styles are discussed in Chapter 14.

Inspecting Position Settings

These settings let you size and position an object as well as arrange and align several selected objects. Autosizing settings let you anchor the edge of an object to its container so that as the container changes size or shape, the object does so, too. This is particularly important if you are developing layouts that will be used on iOS devices that can be rotated.

Naming Objects on a Layout

You can name objects on a layout. Object names must be unique on a given layout. They most often work in conjunction with the new `Go To Object` script step. When you go to a layout, you can immediately select the specific field, portal, tab, or any other layout object that you want to select. You can even determine this dynamically in a script that goes to a given layout—yet another reason for using scripts for layout navigation. The combination of object names and the `Go To Object` script step makes FileMaker Pro a much more powerful interface development system.

tip

The name of an object as specified on the position tab is used for scripting. In the case of text that is formatted as a button using Format, Button Setup, that text is totally separate from the name. You might even want to establish a naming convention for names that makes it clear what they are. For example, a button that appears on the layout as Print might have a name of btnPrint or PrintBtn.

Using Tooltips

A *tooltip* is a small snippet of text that appears when a user hovers the mouse pointer over a layout object (a data field, field label, button, or any other item that can be placed on a layout in Layout mode). The tooltip can provide information about the item beneath it, or it can provide additional information about data in the database, among many possible uses.

note

Remember that there is no such thing as hovering on an iOS device. This means that tooltips never appear on iPads and iPhones.

You can supply a simple text string as the tooltip. Note, though, that you can also use the edit button (the pencil next to the tooltip field), which enables you to define a calculation that is used to generate the tooltip text. The capability to have tooltip text derived from a calculation can lead to some elegant applications. Here are a few examples:

- In a list or portal view of data, in which some data might be too wide for its column, add a tooltip to display the entire data value of the field.
- Create contextual help. If you have Next Record and Previous Records links on a layout, create tooltips that will warn the user if there is no next or previous record (that is, they're at the beginning or the end of the found set).
- Create "smart" Next and Previous buttons that use the `GetNthRecord()` function to display some information about the next or previous record in the set.

For more information on FileMaker's `GetNthRecord()` *function, see "*`GetNthRecord`*," p. 438.*

Tooltips are a standard feature of modern user interfaces. Because they are a relatively new addition to FileMaker Pro, you might want to add them whenever you work on an older solution.

Automatically Resizing Objects on a Layout

The autosizing section of the Position tab lets you resize and relocate an object. For a selected object, you can choose to bind it to the edge of the window or its container (such as a portal) by clicking the appropriate box by the anchor in the direction you want to bind it. You can check from zero to four boxes.

As the layout window is resized, any bound edge of an object moves to keep its same distance from the edge to which it is bound. Thus, you can create a text field that expands horizontally with the window, but whose vertical height is fixed (that is, unbound).

Objects that contain data or that are backgrounds are prime candidates for auto-resizing. Objects such as buttons are not normally expected to change size as a window resizes, but you may choose to anchor them to a constant location relative to the window.

The default behavior is what has happened until now: Objects are placed in a layout, and, as the window is resized, they remain where they are relative to the top and left. Thus, if you widen or lengthen a window, you might have unused space at the right or bottom. Now that you can see what the bindings are, you will see that this is implemented by default with the top and left sides of objects bound to the window.

If you bind an object to the right and bottom, it will stay at the bottom right of the window as you change its size. If you bind an object to the right and left, it will stretch as the window is resized horizontally; likewise, an object bound to the top and bottom will stretch vertically. If you have a large object such as a Web Viewer, you may choose to bind it to all four edges. It will resize as the window is resized in any direction and will retain its distance from each of the new edges.

note

In Preview mode (and, thus, in printing), there is no vertical resizing. Horizontal resizing occurs if the page size is wider than the layout. Furthermore, remember that autosizing happens on the desktop when a user resizes the window. On mobile devices where there are no resizable windows, autosizing happens automatically as the layout is shown on different-sized screens and in different orientations.

Arranging Objects

The Arrange & Align section of the position inspector lets you manage objects.

- **Grouping Objects**—Objects can be grouped together to form a new object. You do so by selecting the desired objects and choosing Arrange, Group or pressing (Command-R) [Ctrl+R]. The resulting object behaves just like any other object. It has a single set of selection handles, and you can move and resize it as described in the previous sections. Any formatting applied to the grouped object is applied to each of the elements of the group, as if you had simply selected all the elements individually. Grouped objects can be further grouped with other objects to form new objects.

  To ungroup an object, select the object and then choose Arrange, Ungroup or press (Command-Shift-R) [Ctrl+Shift+R]. If an object was formatted as a button, ungrouping it deletes the button definition.

- **Locking Objects**—To prevent an object from being moved, resized, reformatted, or deleted, you can lock it by selecting it and choosing Arrange, Lock or pressing (Command-Option-L) [Ctrl+Alt+L]. When you select a locked object, its handles appear grayed out rather than black.

  When you select a combination of locked and unlocked objects and attempt to move or resize them as a set, only the unlocked objects are affected. If you attempt to change the formatting of the selected set, you see an error that the formatting can't be applied to some objects in the set because they are locked.

  Locking objects is useful when you have objects stacked on top of or overlapping one another. It's as if the locked objects become a backdrop against which you do your work. Whether you leave the objects permanently or temporarily locked, it becomes much easier to select and work with certain objects when the objects behind them are locked. To unlock an object, choose Arrange, Unlock, or press (Command-Option-Shift-L) [Ctrl+Alt+Shift+L].

- **Layering Objects**—FileMaker maintains a stacking order for objects on a layout. When you add a new object to a layout, it becomes the frontmost item in the stacking order. The stacking order becomes important when objects overlap one another. If two objects overlap, object A appears in front of object B if it is forward in the stacking order. In addition, if object B is completely behind object A, it is impossible to select object B simply by clicking it. When you click a spot on a layout where multiple objects overlap, you select the frontmost of the objects.

There is no way to review the stacking order of the objects on a layout visually. But you can manipulate the stacking order by using the Bring to Front, Bring Forward, Send to Back, and Send Backward functions, all of which can be found under the Arrange menu.

*The stacking order also determines the tab order of layouts published to the Web with Instant Web Publishing. For more on IWP, **see "Layout Design," p. 591.***

The stacking order also determines the order in which objects draw on the screen. With a local file or on a fast network, the drawing is probably imperceptible, but on slow networks, you will sometimes see the objects draw one by one, from back to front.

- **Rotating and Selecting Objects by Type**—You can rotate object by using commands in the shortcut menu or the Arrange menu of the toolbar.

Aligning Objects

Aligning objects on a layout relative to one another often is desirable, and FileMaker Pro has some built-in tools to make this easy to do. For instance, a layout might have ten fields that you want to be aligned along their left edges. You can use the Align, Distribute, and Resize To menu options, under the Arrange menu, to manipulate objects relative to each other.

You can specify a Top to Bottom alignment, or a Left to Right alignment, or both. You can also distribute objects or resize to the largest or smallest dimensions of the selected objects.

When you align a set of objects relative to one another, one of the objects usually serves as the reference point. For instance, when you left-align a set of objects, the leftmost object is the reference point. The other objects move left while the leftmost object remains in place. Similar results are obtained for aligning to the right, top, and bottom. The exception to this is when one or more of the selected objects is locked. If this is the case, and you want to, say, left-align a set of objects, the leftmost locked object becomes the reference point.

The rules for centering are slightly different. When you are centering left to right, the objects align on the midpoint between the leftmost and rightmost selection points. For top-to-bottom centering, they align on the midpoint between the topmost and bottommost selection points.

The option to distribute space is useful when you want to be sure that objects in a set are equidistant from one another. The two outermost objects, whether left-to-right or top-to-bottom, act as anchors for the distribution: The selected objects in between them are spaced apart evenly.

tip

Even the sloppiest of developers can benefit from this simple process: Select a group of irregularly placed and irregularly sized fields. Then choose Arrange, Align, Left Edges, followed by Arrange, Distribute, Vertically. Last, select Resize To, Largest Width and Height. Voilà—your layout objects are now nicely sized and positioned.

Working with Objects on a Layout

Many tools and techniques exist for configuring and manipulating layout objects. Some apply only to specific types of objects, whereas others are more general in nature. The better you know how to work with the tools for crafting layouts, the better your user interface will be, although there are, of course, no guarantees.

Adding Objects to a Layout

Normally, after you finish creating an object, FileMaker reselects the pointer tool automatically. At times, however, you'll want to create multiple objects of the same type at once. In those cases, it's useful to lock in the selection of a particular tool. You can do this by double-clicking the tool in the Status toolbar. There's also a preference named Always Lock Layout Tools that is located on the Layout tab of the application preferences screen, although we advise against enabling it.

The Insert menu provides another means for adding objects to a layout. At the top of this menu, you'll find selections for adding all the object types found in the Status toolbar.

To insert a picture or another graphic element developed externally, you can use the Insert, Graphic menu command. Alternatively, you can simply cut and paste objects from many other applications directly into your FileMaker layouts.

The Format Painter Tool

You can copy the formatting attributes from one object to other objects on your layout by using the Format Painter tool. The Format Painter can be found under the Format menu and in the Status toolbar.

To use the Format Painter, you select an object that has the formatting attributes you want to propagate and then turn on the Format Painter, using either of the two methods just mentioned. A small paintbrush appears next to your mouse pointer, indicating that the Format Painter tool is active. Then select the object or set of objects to which you want to apply the formats. You can lock in the Format Painter tool by double-clicking its icon on the Status toolbar. This enables you to click several objects and apply formats as you go.

Duplicating Layout Objects

Any object on a layout can be duplicated in one of two ways. When you duplicate an object or a set of objects, the new objects have all the same attributes of the source objects. It is, therefore, often faster and more efficient to create a new object by duplicating an existing one and modifying it rather than by adding a new one using the layout tools.

The first way is simply to select some set of objects and choose Edit, Duplicate or press (Command-D) [Ctrl+D]. The entire set of objects is duplicated, with the new objects appearing 9 points to the right of and 9 points below the original set. The new objects are selected (as opposed to the original set), so you can easily move them to wherever you want.

Points Versus Pixels

A *point* is a unit of measurement; specifically, 1/72 of an inch. *Pixels* are elements of a display. Until the advent of the Retina Display, the assumption was that there were 72 pixels to an inch. As a result, *point* and *pixel* could be used interchangeably. With the higher density of pixels on Retina Display screens and other high-resolution screens, this interchangeability no longer exists. You will notice that in the Position tab of the Inspector, the units are points (*pt*), not pixels (*px*).

A useful technique exists for creating multiple copies of an object spaced out at consistent intervals. Begin by selecting a set of objects, which we'll call set A, and duplicate it as described, creating set B. Without deselecting any of the objects in set B, move them to some desired place on a layout. Choose Edit, Duplicate again; the new copy, set C, instead of having the "6 pixels to the right, 6 pixels down" relationship to its source, is spaced an equal distance from set B as B is from A. Continued selection of Edit, Duplicate results in additional new sets, each positioned a consistent distance from its source. This technique is useful for creating columnar lists and grids of equally spaced lines.

The second way to duplicate layout objects is to (Option-drag) [Ctrl+drag] them. Simply select a set of objects, and then start to drag them as if you intended to move them to a new location on the layout. As you move the objects, however, hold down the (Option) [Ctrl] key. Continue to hold down this key until after you release the mouse click; the objects are not moved, but a copy of them is placed at the new location.

Positioning Objects on a Layout

Much of layout design is simply moving things around until they look just right. This is also one of the most intuitive things for new developers to learn. So much so, in fact, that many never learn some of the fine points of working with objects on a layout. We attempt to remedy that problem here.

Selecting Objects on a Layout

Most object formatting and positioning on a layout begins with the selection of a set of objects to work with. You can go about selecting objects in several ways; knowing these methods can greatly increase your efficiency at designing layouts. Here are your options:

- **Click an object**—You can select any object simply by clicking it. When you do so, small squares, called *handles*, appear at the four corners of the object, indicating that the object is indeed selected. Another set of handles appears in the middle of the top and bottom sides.

- **Shift+click**—When you have one or more objects selected, you can Shift+click an additional object to add it to the selected set. Similarly, Shift+clicking an already-selected object removes it from the selected set.
- **Selection box**—If you click the background of the layout (that is, any place there's not an object) and drag a rectangle across the screen, any objects that were contained within your selection box are selected when you release the mouse. This is typically the easiest and quickest way to select multiple objects.
- **Select all objects**—To select all the objects on a layout, choose Edit, Select All, or use the (Command-A) [Ctrl+A]) keyboard shortcut.
- **Select all instances of a type of object**—It's also possible to select all instances of a particular type of object, such as all the text objects, or all the fields, or all the rectangles. There are several ways to do this. You can select an object and then press (Command-Option-A) [Ctrl+Alt+A] to accomplish the same thing. Finally, if you have a tool other than the Button or Portal tool selected from the layout tools, you can select all the objects of that type by choosing Edit, Select All.

Moving Objects

After you select a set of objects, you can move those objects around on the layout—provided that they are not locked—in a few ways. First, you can click the interior of any object in the selected set and drag the set to a new location. You can also use the arrow keys on your keyboard to move a selected set of objects point by point.

Resizing Objects

When you select an object, four handles appear at the corners of the object and another four appear at the midpoints of all four sides of the object. All objects, even circular ones, have a rectangular footprint defined by the four handles at the corners. Gray handles indicate that the object is locked; it can't be moved or resized in this state.

*For more information on this topic, **see** the bulleted entry, "Locking Objects," **p. 142**.*

You can resize an object by clicking one of the handles and dragging in the desired direction. If you have selected multiple objects, resizing any one of them causes all the objects to resize by a similar amount. This capability is useful in cases where you want to select, for instance, five fields and make them all slightly longer or shorter. Resizing them as a set ensures that they all change by the same relative amount.

To use the Resize To alignment tools, access the Arrange menu in Layout mode. These tools allow developers to make a group of objects consistent by resizing all objects in the group to the largest or smallest width or height of the objects selected.

The Object Grid

You have the option, when working with layouts, of enabling or disabling an object grid. You can change the status of the object grid by toggling the Object Grids command found at the bottom of the Arrange menu. You can also toggle the status of the object grid by pressing (Option-Command-Y) [Alt+Ctrl+Y]. In addition, the grid options are available at the bottom of the Position tab of the Inspector in the Grid section.

When object grids are enabled, all movement and resizing of objects takes place against a virtual grid.

The object grids are defined relative to each object; that is, there's no static grid to which everything snaps. If object A and object B are 2 points apart, with object grids enabled, you could move each object one "chunk" in any direction and they'd still be 2 points apart, each having moved 6 points from its original location.

Whether you choose to have object grids enabled as you design layouts is purely a personal preference. Some developers love object grids; others loathe them. The benefit of using the object grids is that they make it easy to keep things arranged and sized nicely. It's much easier to notice visually when an object is 6 points off-line rather than 1 point. Plus, if you ever need to move things in finer increments, you can simply use the arrow keys to nudge the objects into line. In addition, you can temporarily suspend the object grids by holding down the (Command) [Alt] key as you move or resize an object. On the con side, for developers accustomed to positioning things exactly to the point, the object grid can get in the way and prove simply cumbersome to work around.

The object grid's status is a file-level setting. That is, as you work on different layouts within a file, the grid status carries through to them all. But if you have multiple files in a solution, you could conceivably have the object grid enabled in some files and not in others.

note

The grid spacing can be set in the Grid section at the bottom of the Position tab of the Inspector. It might start out as 6 points, but you can modify it.

For more positioning tools, **see** *"Using Guides" and "Using Dynamic Guides,"* ***p. 392 and 393****.*

Working with the Tab Control Object

The tab control does one thing: It extends the amount of screen real estate you can provide users by allowing them to flip from one pane to another while remaining on the same layout and same record. For an example, refer to Figure 4.4: at the right is a tab control with two tabs—one for home address and the other for work address. This is a frequent type of use for a tab control because both tabs are presenting variations on the same data element.

Adding a Tab Control Object to a Layout

To add a tab control object to a layout, click the Tab Control button in the Status toolbar and draw a rectangular area on your layout. You are presented with the Tab Control Setup dialog, as shown in Figure 4.24.

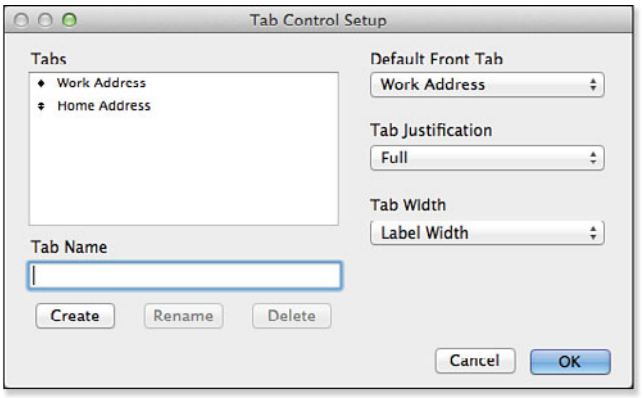


Figure 4.24
The Tab Control Setup dialog enables you to create however many panes you require.

In the Tab Control Setup dialog, you can add as many tab panes as necessary and then choose alignment and tab styles. Although the options aren't exhaustive, the simplicity of working with the tab control object will no doubt quickly win you over.

The width of the tabs on the tab control can be set in the Tab Control Setup dialog. Your choices are as follows:

- Label Width
- Label Width + Margin of (to provide a small space on each side)
- Width of Widest Label
- Minimum of
- Fixed Width of

If you select a tab in the tab control, you can use the Position tab of the Inspector to name it. Then, when using a script to go to a layout with a tab control on it, you can go to a specific tab—even changing the tab depending on circumstances. After you close the dialog, you remain in Layout mode and can add layout objects—including additional tab control objects—to the tab pane currently selected.

The tab control is operational in Layout mode. If you click a tab once, you flip to the pane it represents. If you want to return to editing in the Tab Control Setup dialog, double-click the tab control

object. If you want to edit the tab pane's properties (color, line weight, and line color), click the tab a second time. You will see an active rectangle appear.

When you select the tab control object, notice that its rectangular area includes its tab space. The negative space next to your tabs when they're not set to Full justification is still considered part of the selected pane. One handy technique we've learned is to place a button or text or even field objects in that space: They appear and disappear just as all objects for a pane do.

Working with Fields

The primary purpose of a layout is to allow users to interact with data. By interact, we mean everything from viewing, editing, and formatting to finding and sorting. Although a field is at some level just another type of layout object and can be manipulated using the same tools as other layout objects, a number of tools are designed specifically for working with fields. They provide you with a great deal of freedom and flexibility for creating the interfaces that work best for your users and your solution. We don't cover every option of every tool here, but rather try to give you a sense of what the tools are and some of the situations in which to use them.

Adding Fields to Layouts

There are several ways you can add fields to a layout: by using the Field tool in the Status toolbar to draw it, by using the Add Field button in the Status toolbar, and by duplicating an existing field. The first of these—which is generally also the first method that people learn—involves clicking and dragging the Field button in the Status Area out to the section of the layout where you want to place the field. The current theme format attributes govern the attributes of a field added this way.

There is another way to add fields to a layout. If Add Newly Defined Fields to Current Layout is set in the Layout tab of Preferences, the Manage Database dialog will add fields automatically using the default settings if the current layout's base table is the table to which you have added the fields.

As with other layout objects, when you duplicate an existing field, the new field has all the attributes of the previously existing field (including its width). Remember, to duplicate any layout object, you can select it and either choose Edit, Duplicate or press (Command-D) [Ctrl+D], or select it and then (Option-drag) [Ctrl-drag] to a new location. In either case, if you have selected a single field, when you duplicate it, you see the Specify Field dialog and can select the new field. On the other hand, if you select multiple objects, when you duplicate them, you get just the duplicated objects. Keep in mind that you duplicate all the attributes of a field—including any button behaviors you attached to it, tooltips you assigned, and so on.

note

There are some issues to be aware of when copying and pasting fields from a layout in one file to a layout in another file. See "Copying and Pasting Fields Between Files" in the "Troubleshooting" section at the end of this chapter.

Each field object on a layout is defined to display data from a particular field. Unless you selected Sample Data in the View, Show menu, you see the field's name on the object when you're in Layout

mode. If you see :: at the beginning of the field name, that's an indication that the object is linked to a related field. The Specify Field dialog lets you view and change the database field used in the field object on a layout. In Layout mode, double-click the field to open that dialog. You can also view and change the database field for a field object on a layout using the Field section of the Data tab in the Inspector.

Setting the Tab Order

When moving from field to field on a layout with the Tab key—or Return or Enter, as described in the preceding section—the order in which the fields are activated is known as the *tab order*. The default tab order is the order in which the fields appear on the layout from top to bottom. Rearranging fields changes the tab order.

Tab order is stored with the layout, so there's no opportunity to customize the tab order for different users. The Set Tab Order dialog is shown in Figure 4.25. After you edit the tab order manually, rearranging fields doesn't change the tab order. New fields are added to the end of the tab order automatically, regardless of position.

tip

If a field has been defined as a button, double-clicking it takes you to the Button Definition dialog, not the Specify Field dialog. Similarly, if multiple fields are grouped together, right-clicking gives you only the control elements of the dialog. This is one of the reasons many people prefer to use the Inspector: Its behavior is more predictable in some ambiguous situations.

note

If you click the Add Field button in the Status toolbar, you are simply asked to select the field to use. You must later use the Inspector to modify its attributes.

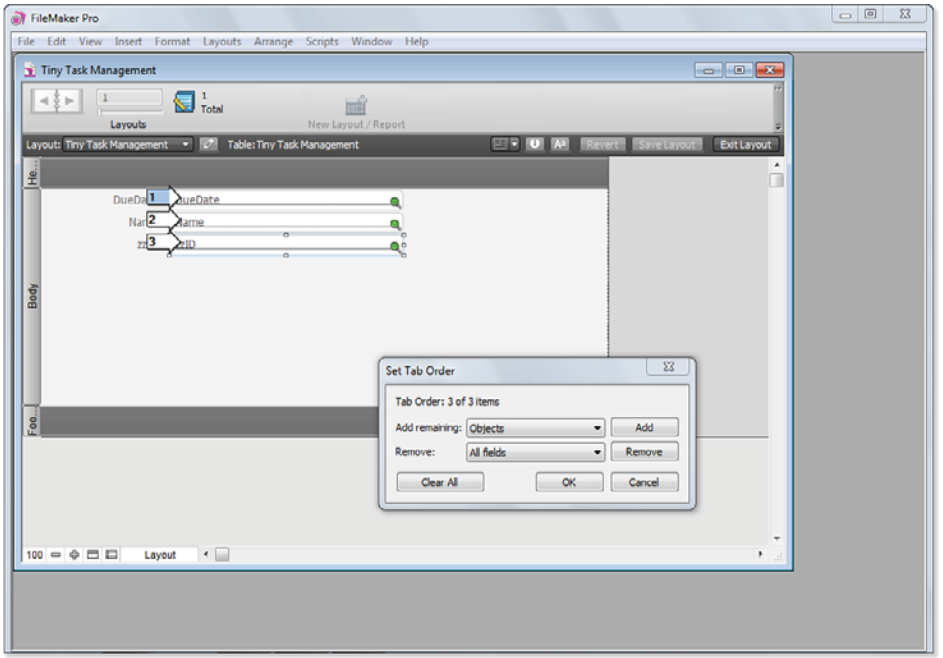


Figure 4.25 You can change the tab order of a layout to make data entry flow in a logical progression for end users.

Anyone with the ability to modify a layout can change its tab order; this is done by selecting Layouts, Set Tab Order. You can click both fields (the arrows to the left of objects) and objects themselves (the arrows to the right of objects) to manually edit the tab order for a given layout. You can remove items by pressing the Delete or Backspace key.

The dialog operates on like objects. If you want to add all fields to the tab order or remove all objects, choose from the two menu options in the dialog and click the button for the appropriate command.

Note that it is entirely possible to attach a button behavior to a field and for that field to appear twice in the tab order. One instance tabs into the field for editing, and another selects the field to perform the button action.

The Set Tab Order dialog allows developers to add and remove both fields and objects (including tab control object tabs) from the tab order of a given layout.

To further enable users to work from keyboard commands, and review adding keyboard shortcuts via Custom Menus, ***see*** *"Working with Custom Menus,"* ***p. 396****.*

Merge Fields

If you've ever done a mail merge, the concept of merge fields should be familiar to you. Merge fields give you a way of incorporating field data within text blocks on a layout. This feature is useful for creating form letters, labels, and reports.

Merge fields display field data, but they don't behave like or have all the properties of normal fields. A user can't click into a merge field to do data entry, for instance.

To add a merge field to a layout, choose Insert, Merge Field, or press (Command-Option-M) [Ctrl+Alt+M]. You are prompted to specify a field. After you make your selection, the field name shows up on your layout surrounded by angle brackets. Note that you can add a merge field alone to a layout or you can incorporate it, and potentially others, into a block of text within a text object. A single merge field can contain several actual fields as well as constant text so that it might appear as follows:

```
Total: <<sTotalPopulation>>; Average Age: <<sAverageAge>>
```

The primary benefit of merge fields is that field data can be flexibly placed within a text block; text before and after the merge field is repositioned to close up any extra space. Thus, within a text block, you could have "Hi, <<First Name>>, how are you?" On one record, that would come out as "Hi, Joe, how are you?", whereas on another it might be "Hi, Frederick, how are you?"

Text, number, date, and time formatting applied to a text block is applied to any appropriate merge field within the text block. It is not possible to have a single text block that contains multiple merge fields having different formats applied to them.

note

Merge fields provide a good way of displaying uneditable data on a layout so that it wraps as you would expect it to in a word processing program. (On report layouts shown in Preview mode or that are printed, the field-sliding controls can do the same thing.) However, remember that a field can be searchable, whereas a merge field cannot. You might want to use a field where the behavior does not allow editing in Browse mode but does allow editing in Find mode so that a title is searchable.

Troubleshooting

Copying and Pasting Fields Between Files

When I copy and paste fields from a layout in one file into another file, sometimes the fields retain their proper identity, sometimes they have no identity, and sometimes they have the wrong identity. Why is that?

When you copy fields from a layout in one file and paste them into another file, they might or might not retain their identity, as you've discovered. A field retains its identity when there exists a field in the destination file that has the same source table and field name as the source field. Additionally, the layouts must be based on identically named table occurrences. It's not enough for the source tables to be named the same. If the table occurrences match, but no similarly named field is found in that table, the field displays <Field Missing> when it's pasted into the destination file. If the table occurrence names don't match, the field shows up without any identity in the destination file.

Given the ease with which you can copy and paste tables using FileMaker Pro Advanced, we recommend first creating a compatible schema in the destination file and then copying your layout objects.

Determining Which Records Will Be Displayed on a Layout

I created a table occurrence that's supposed to display only invoices that are more than 60 days overdue. However, when I build a layout based on this table occurrence, I still see all the invoice records. What did I do wrong?

The problem here isn't anything you've done or haven't done, but rather your expectations. The table occurrence to which a layout is tied never determines which records from the source table are displayed on that layout. It merely determines the starting point on the Relationships Graph from which any action or object involving a relationship is evaluated. To view a set of related records, you must establish a perspective through which those records are viewed; in other words, you'll need a portal.

If you have a layout that's tied to an occurrence—any occurrence—of an Invoice source table, all the records from the Invoice table can be viewed from the context of that table occurrence. Think of it this way: A layout's table occurrence doesn't determine what records you can view from that layout; rather, it determines what records the records of that table can view. Therefore, in the case of your table occurrence, which is supposed to show only invoices that are more than 60 days overdue, you'd need to view those via a portal from a layout tied, say, to a Customer table.

5

RELATIONAL DATABASE DESIGN

Understanding Database Design

By now you've designed a simple FileMaker database and built some nice data entry screens and some reports. Your friends and coworkers are clamoring for you to add features. Can your system do invoicing? Inventory tracking? Barcoding?

Well, it can probably do all those things, but it's going to take some planning. If this is your first time out with FileMaker, you're like the home carpenter who's just built a birdhouse. It's a nice birdhouse, but the kids wanted a tree fort. That's not just going to take more work; it's going to take more thought as well.

FileMaker is a tool for building database solutions. The solution consists of both the database and the interface that you build in FileMaker. The work of designing and building the database is related to the work of designing and building the interface, but the two tasks have different features. As a broad generalization, you can think of the design and implementation of a database as being focused on the data, while the design and implementation of the interface are focused on the people who will use the data.

In this chapter, you'll see the fundamentals of database design. When you're designing a simple contact manager or recipe book, the database structure is pretty clear as long as you understand the problem you are addressing. You know what fields you need to track and what kinds of fields they are. However, as you will see throughout this part of the book, sometimes the first intuitive ideas are actually not the best choices. And when you get into tracking additional categories of data in the same database, things can really get trickier. If you want to build bigger and better databases, you need a firm grounding in database analysis and database design. Don't worry if that sounds ominous. It's easier than it appears.

Database Analysis

One of the great beauties of FileMaker is that it's very easy to just jump right in and start building things that work. And this is fine, as long as you can keep the whole plan in your head.

Earlier chapters have looked at some practical techniques for separating and organizing data in a FileMaker database system. This chapter takes that work another step. Here, you learn some tools for analyzing database problems and translating them into buildable designs.

This chapter approaches things and their relationships somewhat abstractly. Your goal here isn't a finished FileMaker solution, but rather a more general design document. You learn a simple but powerful design process to help you take a real-world problem description and translate it into a blueprint that a database designer could use to build the database in a real-world database development system. This design document is an entity-relationship diagram (ERD). The process for creating an entity-relationship diagram, somewhat simplified, looks like this:

1. Identify all the types of things involved in the problem being modeled (customers and sales, for example, or trucks, drivers, and routes).
2. For each type of thing, identify its attributes (customers have first and last names; truck routes have a beginning and an end).
3. Looking across all the types of things, determine the fundamental relationships between them (truck drivers have routes; trucks have drivers).
4. Draw up your findings into an entity-relationship diagram.

The ERD, again, is an abstract document that you can implement (build) with FileMaker or some other database tool. The sections that follow examine each of the steps of this process in much more detail.

Working with Entities and Attributes

When you set out to design a database, there are two concepts you simply must be familiar with before you can say you have a solid planning foundation. You need to know the types of things your system will track, and you need to know the characteristics of each of those things. In a recipe list, for example, you track one kind of thing: recipes. A recipe's characteristics are, for example, recipe name, recipe type, calories, ingredients, and instructions. A bigger database might store information about several kinds of things, each with its own set of characteristics. For example, if you want to write a database for a motorcycle company, you might want to track information about motorcycles, customers, and sales. Now you have three kinds of things, each with its own set of characteristics.

In database design terminology, the things in your database system are *entities*. Each entity is a specific, distinct kind of thing about which you need to track information. This system tracks data about three distinct kinds of things, and each kind of thing has certain characteristics, which in the technical jargon are *attributes*. The motorcycle example includes three entities, and each has some specific number of attributes (see Table 5.1).

Table 5.1 Simple Analysis of a Database Structure

Motorcycle	Customer	Sale
Model Number	First Name	Customer Name
Model Year	Last Name	Date
Vehicle ID Number	Birth Date	Amount
Factory Serial Number	Street Address	
Accessories	City	
Manufacturer	State	
Model Name	ZIP	

The first step in database design is to determine what entities (things) your proposed system needs to track, and what the attributes (characteristics) of each entity are. Your list of entities and their attributes will change during your analysis.

An entity is a class of things that all look more or less alike. In other words, from a database standpoint, you track many instances of an entity, and you track the same kind of information about each instance. In a banking system, you'd probably have an entity called Customer because a banking database needs to keep track of many different customers, and needs to record roughly the same kinds of data about each one. You'll always want to know a customer's birth date, Social Security number, home address, and the like.

Attributes, on the other hand, refer to the kinds of information you track about each entity. If Customer is an entity in your banking database, some of the attributes of the customer might include birth date, home address, and Social Security number.

Entities often correspond to actual database tables, and attributes often correspond to database fields. More likely than not, a banking database will have a Customer table with fields for date of birth, address, and Social Security number.

It's fairly easy to represent entities and attributes in the graphical notation of an ERD. Sometimes it's more convenient to draw an entity without showing any of its attributes, in which case you can draw it in a simple box, as shown in Figure 5.1.

note

The entities in these diagrams are purely abstract things. They might or might not translate directly into database tables or even physical objects. Your FileMaker solution might (and almost certainly will) end up with tables not represented on your design diagram.

Figure 5.1
A simple, preliminary ERD showing entities for customers and accounts, with no attributes shown.

Sometimes it's appropriate to show entities with some or all of their attributes, in which case you can add the attributes as shown in Figure 5.2.

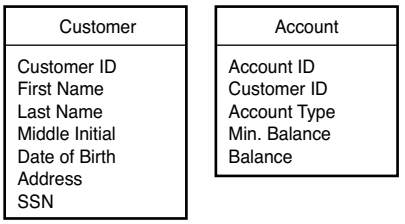


Figure 5.2
An ERD showing entities for customers and accounts, with attributes shown.

tip

All design processes are iterative. As you move along, constantly check your logic; it is easy to enthusiastically wander down a garden path into a swamp. For example, in the database structure described here, look to see whether you made any invalid assumptions. Not every assumption is wrong, but you should know if you are in any way limiting or distorting the data.

For example, the attributes of customers include a birth date. That means that a customer can be only a single person with a single birth date. Is that a reasonable assumption? It might be. Perhaps there is a terminology tweak: If your customer is a company, the birth date field might be the date of incorporation or registration of the company. You would then change the name of the field so that both circumstances are covered. Depending on the purpose for which the field is to be used (in the case of two people joining together to buy a motorcycle, for example), the birth date might be that of the younger (or older) customer.

Or, after realizing that the inclusion of that field raises questions, you might go back to the user (or to yourself if you are the user) and ask why that field is included. Sometimes the review and questioning process clarifies the data structure by simplifying it and removing unneeded data.

Entities Versus Attributes: A Case Study

The focus of this chapter is in taking descriptions of real-world problems and turning them into usable ERDs. As was noted earlier, your first step in trying to model a problem into an ERD is sorting out the entities from the attributes. To see how to tackle this, let's begin with an example of a simple process description:

> Maurizio's Fish Shack is ready to go digital. Maurizio sells fish out of his storefront, but he's not worried about electronically recording his sales to consumers just yet. He just wants to keep track of all the fish he buys wholesale. Every time he buys a load of fish, he wants to know the kind, the quantity, the cost of the purchase, and the vendor he bought it from. This information will give him a better handle on how much he's buying and from whom, and may help him negotiate some volume discounts.

Now you know the basics of Maurizio's business. Next, you need to develop a list of potential entities. Here are some possibilities:

Fish	Load of Fish	Purchase
Storefront	Variety	Vendor
Sale	Quantity	Volume Discount
Consumer	Cost	

These possibilities are typically referred to as *candidate entities* in that they all represent possible entities in the system. But are they all entities? You can immediately cross Storefront, Sale, and Consumer off the list, for the simple reason that the process description already says that these are parts of his business that Maurizio doesn't want to automate at this time. That leaves us with the following potential entities:

Usually, the rule of thumb to apply when coming up with a list of possible entities is to pull out every word that's a noun; in other words, every word that represents a specific thing.

Fish	Quantity	Vendor
Load of Fish	Cost	Volume Discount
Variety	Purchase	

Well, Fish and Load of Fish look as though they refer to the same thing. According to the process description, a load of fish is actually a quantity of fish that Maurizio bought to resell. Put in those terms, it's clearly the same thing as a purchase. Now the list looks like this:

Purchase (of Fish)	Quantity	Vendor
Variety	Cost	Volume Discount

These possibilities all seem like reasonable things to track in a database system. But are they all entities? Remember that an entity is a kind of thing. The thing will probably appear many times in a database, and the system will always track a coherent set of information about the thing. Put that way, a purchase of fish sounds like an entity. You'll record information about many fish purchases in Mauriziodatabase.

What about something like cost? Cost in the process description refers to the price Maurizio paid for a load of fish, so cost isn't really an entity. It's the price paid for one load of fish. It's actually a piece of information about a fish purchase because each fish purchase has an associated cost. The same is also true for Variety and Quantity. These are all attributes of the Purchase entity.

Then you get to Vendor. A vendor is clearly a category of things; you'll probably want to store information about many vendors in this database, so you can consider a vendor to be an entity. This leaves Volume Discount. Well, that one's a bit tricky. It probably applies to a vendor and might reasonably be called an attribute of a vendor. If you assume that each vendor can offer a discount of

some kind, it makes sense for Volume Discount to be an attribute of Vendor. Figure 5.3 shows what the fledgling ERD for this system might look like with the two entities from the process description and their various attributes.

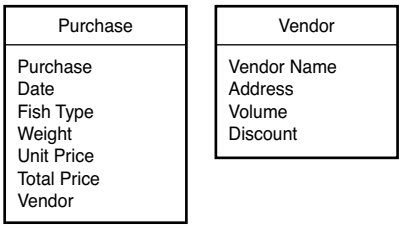


Figure 5.3
An ERD showing entities for fish purchases and vendors, with attributes shown.

Design as an Iterative Process

Your general task when designing a database (or indeed any piece of software) is to take a set of things in a real-world problem domain and translate them into corresponding things in the software domain. In your software, you create a simplified model of reality. Concepts such as "fish purchase" and "fish vendor" in the problem domain turn into concepts such as "purchase entity" and "vendor entity" in a design, and might ultimately turn into things such as "purchase table" and "vendor table" in the finished database.

But this translation (from problem domain to software) is not a one-way street. It's rare that a single, unambiguous software model exists that corresponds perfectly to a real-world problem. Usually, your software constructs are approximations of the real world, and how you arrive at those approximations depends a lot on the goal toward which you're working.

For example, in your initial reading of the design problem, you might miss an entity or two, or you might create entities you don't really need on later examination. Later, as you do more work on the project and learn more about the problem domain, you may revise your understanding of the model. Some entities might disappear and become attributes of other entities. Some attributes might turn out to be entities in their own right. You might find you can combine two similar entities into one. Or you might find out that one entity really needs to be split in two. At this stage, you are working on paper, so even a massive redesign is not costly.

Consciously looking at the problem from varying perspectives can be a useful design tool. You can easily get trapped in one way of looking at things. Another point of view (or a break from the design process) can often help you create a simpler structure.

Understanding Relationships

We've dealt with the first two steps of the design process now: the sorting out of entities and attributes. After you have what you think is a decent draft of a set of entities and attributes, the next step is to start considering how these entities relate to one another. You need to become familiar with the fundamental types of entity relationships and with a simple notation for representing relationships graphically in a diagram.

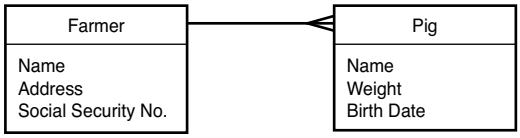
Representing Relationships in a Diagram

Consider a system that stores information about farmers and pigs, among other things. Farmers and pigs are each entities, and these two entities have a direct relationship, in that each pig ties back to a single farmer.

There's a name for the farmer-pig relationship. It's called a *one-to-many relationship*, meaning that for each farmer there may be any number of pigs.

Now you can expand on the entity-relationship notation. You already have a graphical shorthand for depicting the entities and attributes in a database system. Next, you should add some conventions for showing the relationships among them. Each entity can be represented by a box, as before, and each relationship can be represented by a line that indicates the relationship type. In this simple notation, you would depict the relationship between farmers and pigs along the lines of what's shown in Figure 5.4.

Figure 5.4
Entity-relationship notation for a database that stores information about farmers and pigs.



Notice that the line between the two entities that depicts their relationship branches out where it touches the Pig entity into a kind of crow's foot. In a one-to-many relationship, this crow's foot indicates the "many" end of the relationship. This notation tells us that one farmer can be linked to many pigs. If the fork were on the other end, it would imply that one pig could be associated with many farmers, which would be a very different assertion about the data we're trying to model.

Relationship Types

Those simple graphical conventions are the foundation of what you need to draw your entity-relationship diagrams. Another important concept is an understanding of the different relationship types you could encounter. You need to reckon with four types: the one-to-one relationship; the one-to-many and many-to-one relationships (the latter is simply a one-to-many relationship looked at from the other direction); and the many-to-many relationship. (This is sometimes referred to as *cardinality*). We'll consider each of these relationship types in turn and show how to represent them in the ERD notation.

One-to-One Relationships

Consider a data set concerning children and their birth records. Let's say that for now, you've decided that children and birth records should represent separate entities.

In a standard analysis sequence, after you've decided on entities and attributes, you'll start to ask questions about relationships. What's the relationship between children and birth records? Can one child have many birth records? No, each child is born only once. And can one birth record pertain to

more than one child? Again, probably not. Therefore, the relationship between a child and a birth record appears to be one-to-one. You can depict that as shown in Figure 5.5.

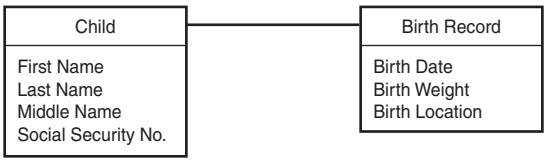


Figure 5.5
This ERD shows the one-to-one relationship between children and birth records. A single line with no crow's-foot is used.

You'll often fold one of the one-to-one entities into the other. In this case, you might decide to move all the attributes of a birth record into the Child entity and get rid of Birth Record as a separate entity.

But there are many cases in which a one-to-one relationship is the appropriate choice. Perhaps the most common is the situation in which the data involved comes from two separate domains (or even two databases, using external data sources such as ODBC databases). If you have a complete database structure for contacts, with one-to-many relationships for contacts and their multiple addresses as well as contacts and their multiple activities, you might choose to leave that entire structure intact. You could choose to forge a one-to-one relationship between employee, a key part of your own database, and the contact, which is in another database or another area of your database.

One of the biggest obstacles to successful database development is the tendency for some designers to insist that their database is the center of the universe. In fact, the most successful databases are those that work well with other well-designed databases.

One-to-Many Relationships

We've already devoted some attention to the one-to-many relationship. The relationships of a customer to sales, of a farmer to pigs, and of a worker to timesheets are all examples of one-to-many relationships. And as you've seen, the crow's-foot notation is used for indicating these relationships in which the fork notation indicates the "many" side of the relationship.

You'll frequently see the entity that represents the "one" side of the relationship referred to as the *parent entity*, whereas the "many" side is often referred to as the *child entity*.

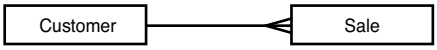
Many-to-One Relationships

There's no difference at all between the concepts of a one-to-many and a many-to-one relationship. They're the same idea, just seen from different points of view. If the relationship between customers and sales is one-to-many, it's equally true that the relationship between sales and customers is many-to-one. Customer is the parent of Sale, and Sale is the child of Customer. These statements are equivalent. Figure 5.6 shows the Customer-Sale relationship.

note

Whether you choose to describe this as a one-to-many or a many-to-one relationship depends on which side you start from in your description. The relationship of a customer to a sale is one-to-many; the relationship of a sale to a customer is many-to-one. One-to-many and many-to-one are two sides of the same coin; a relationship can't be one without being the other.

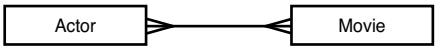
Figure 5.6
The Customer-Sale relationship drawn as both a one-to-many and a many-to-one relationship.



Many-to-Many Relationships

Consider the relationship between actors and movies. One actor can play roles in many movies, and one movie involves roles played by many actors. Therefore, each actor can relate to many movies, and each movie may be associated with many actors. In fact, one actor might even play several roles in a single movie. This is a classic many-to-many relationship. You can depict it as shown in Figure 5.7.

Figure 5.7
Entity-relationship notation for a many-to-many relationship.



Many-to-many relationships are extremely common in relational database systems. Here are examples of some other many-to-many relationships:

- **Attorney-Case**—One attorney can serve on many cases, and one case can involve many attorneys.
- **Player-Game**—One player can play in many games, and one game involves many players.
- **Product-Invoice**—One invoice can contain orders for many products, and one product can be ordered on many different invoices.
- **Student-Class**—One student can participate in many classes, and one class can have many students enrolled.

Many-to-many relationships are a bit trickier than the others to actually implement in real life. When we get to the details of how to build a FileMaker database based on an ERD, you'll see the specific techniques you need to bring a many-to-many relationship to life in FileMaker. For now, though, we'll just use the ERD as an analysis tool and not worry about implementation.

Understanding the Role of Keys in Database Design

So far there's been no discussion of exactly how a relationship between two entities is created and maintained. The answer is simple: We create fields in each entity called *keys*, which allow instances of one entity to be associated with instances of another. You might relate orders to customers, for example, by using a customer's customer number as a key. Each order would then contain the customer number of the related customer as one of its attributes. The following sections explore the concept of keys in more detail.

Keys That Determine Uniqueness

One of the crucial tenets of relational database theory is that it has to be possible to identify any database row, anywhere, without ambiguity. Put differently, every row in every table should have a unique identifier. If I have a record in a table of orders, I want to be able to ask it "What customer do you tie to?" and get an unambiguous answer. I need a simple answer: "Customer 400." End of story. The number 400, as it appears in the customer table, is a unique identifier.

A piece of data capable of uniquely identifying a database row is a *primary key*. A primary key is an attribute whose values are (and always will be) unique for every single row in the database. It's a unique identifier, such as a customer number, a book's ISBN number, or a library card catalog number. A unique key is not a requirement in FileMaker, but it is a good idea to provide one.

What Makes a Good Primary Key?

So far, we've mentioned that every database table should have a primary key, and that those keys have to be unique, to distinguish one row from another absolutely. There's one other important rule: Primary keys are best if they're meaningless.

The important idea here is that data chosen to act as a primary key should be free of real-world meaning or significance. When data has meaning in the real world, such meaning is subject to change. In simple terms, data that is supposedly unique might turn out not to be.

Here's an example. You're designing a database that holds information about the different offices of a company. Offices are stored in their own table. You decide that because there's no more than one office in a city, the City field in the Office table would make a great primary key. It's unique, after all, and every Office record has a City value.

Just to be sure, you check with someone highly placed in the firm, and they assure you that, no, the company will never need to open more than one office in any one city. You go ahead and build a database structure around the assertion that the City field in the Office table is unique. Seven months later, the company announces plans to open its second office in New Delhi, and you're left to explain why an important part of the database structure has to be rewritten.

Imagine instead that you had decided that the database system itself should generate a primary key. Offices will be numbered sequentially starting from 1. The important thing about this data is that it has meaning only to the database system itself. No one else cares, or even knows, that the New Delhi office is office number 14. The number 14 has no business significance.

The critical difference here is that when you used the City field as a primary key, you relied on the stability of an assertion about the real world (a place notoriously subject to change). By contrast, when you create your own key, you're working in an environment that no one but the database programmers care about, so you're at liberty to design uniqueness rules unaffected by decisions beyond your control.

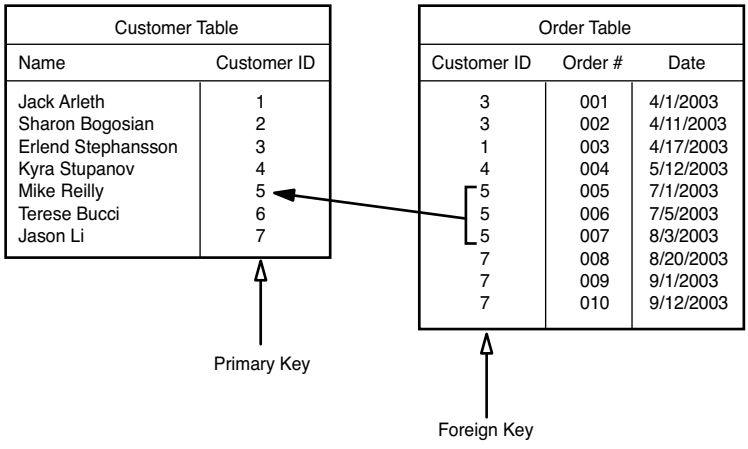
Keys That Refer to Other Tables

Keys are essential to specifying relationships between tables. Going back to the example of customers and orders, the relationship between these entities is one-to-many: One customer can have many orders.

If you've followed the rule about always having a primary key, your Customer entity has a primary key, which you might call Customer ID. Now, each unique customer may have many related orders. To forge that relationship, each record in the Order table needs to store the Customer ID (the primary key) of the related customer. This value, when it's stored in the Order table, is a *foreign key*. The reason for the term is simple: The value in the Order table refers to a primary key value from a different ("foreign") table.

Figure 5.8 demonstrates how primary and foreign keys work together to create relationships between database tables. In a one-to-many relationship, the "many" side of the relationship always needs to contain a foreign key that points back to the "one" side. The child record thus "knows" who its parent is.

Figure 5.8
A one-to-many relationship between customers and orders, showing primary and foreign keys.



FileMaker Pro has several built-in capabilities that help you add strong key structures to your FileMaker databases. For some ideas on how best to define key fields in FileMaker Pro, **see** *"Working with Keys and Match Fields,"* **p. 176.**

Many-to-Many Relationships

Assume that you're building a class registration database. It's intended to show which students are enrolled in which classes. It sounds as if you just need to deal with two entities: students and classes. Students and classes have a many-to-many relationship. One student might participate in many classes, and one class can contain many students. That sounds fine, but how would you actually construct the relationship?

Based on the fundamental rule mentioned earlier, you need a primary key for each entity. Student needs a Student ID, and Class needs a Class ID. If you look at things from the student side for a

moment, you know that one student can have many classes. Accordingly, from that viewpoint, Student and Class have a one-to-many relationship. If that's the case, from what you now know about foreign keys, you might conclude that each Class record should store a Student ID to indicate the student record to which it relates.

This won't work, though, for the simple reason that one class can contain many students. This means that the Student ID attribute in Class would have to contain not just one student ID, but a list of student IDs—one for each enrolled student. The same would be true in the other direction: Each student record needs a Class ID attribute that stores a list of all classes in which the student is enrolled.

Using Join Tables

One rule of relational database design that has already been touched on is that it's almost always a bad idea to store lists of things in database fields. As a general rule, when you find you're using a field to store a list of some kind, that's a sign you need to add another entity to your system where you can then store the list items as single records. This should suggest to you that the many-to-many problem can't be solved without some kind of additional entity. This is true, and it leads to a simple rule:

> Resolve a many-to-many relationship by adding an additional entity between the two in question.

Figure 5.9 shows an ERD for students and classes with an additional entity to solve the many-to-many problem.

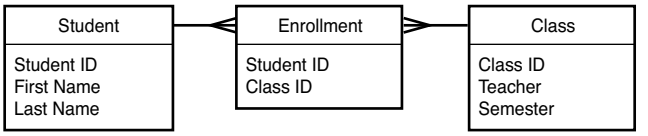


Figure 5.9
An ERD for students and classes.

This middle entity is often called a *join table*. Each of the outer entities now has a one-to-many relationship with this middle entity. Not surprisingly, then, the middle entity has two foreign keys because it's on the "many" side of two different relationships. It needs to hold both a Student ID and a Class ID.

What, if anything, does this entity represent in the real world and what should it be called? One useful exercise, after you've resolved a many-to-many relationship, is to say to yourself, "This join entity represents the association of one A with one B." In the example of students and classes, the middle entity represents the association of a specific student with a specific class. If you think of the entity as a database table (which it will almost certainly become), each row of the table holds one student ID and one class ID. If such a row holds the student ID for student number 1009023 (Sam Tanaka) and the class ID for class H440 (History of the Sub-Sahara), this record tells us that Sam is (or was at some point) enrolled in History 440. This also suggests a good name for the entity: Enrollment. Each record in this table stores the enrollment of one student in one class.

Using Checkboxes and Multiple Values

With FileMaker, you can store multiple values in a field. Although it is normally not a good idea to store lists, these multiple values are an important part of FileMaker, and you can safely use them for many-to-many joins. Each value is separated by a return character. When you create a relationship from a multivalued field to another field, all the matching values can be used, giving you a many-to-many join.

And just to make things even simpler, the simplest way to store multiple values in a field is by using a checkbox set. FileMaker takes care of inserting the return characters as needed. This gives you a simple way of managing many-to-many joins. Using a separate join table is essential if you're going to want to store data in the join table (see "Attributes in a Join Entity" next), or if you want to use the five standard fields to track when each of the components of the many-to-many join was created and by whom.

Attributes in a Join Entity

You've seen that this join entity needs, at the very least, two foreign keys: one pointing to each side of a many-to-many relationship. What other attributes does it need?

Well, looking at the example of students and classes, you might wonder where you'd store an important piece of information such as a student's GPA. A student has only one GPA at one time, so you should store that as an attribute of the student. But what about course grades? Where do you record the fact that Sam earned a B+ in H440? Well, Sam can be enrolled in many courses and thus can receive many grades. Therefore, it's not appropriate to try to store the grade somewhere on Sam's student record. It belongs instead on the enrollment record for that specific course. And, if attendance was being taken, Sam's attendance would logically go on his enrollment record as well.

Sometimes join entities have attributes of their own, and sometimes they don't. You'll have to ask yourself whether you're merely trying to record the fact that the entities are associated or whether there are additional attributes of their association.

note

The question of adding keys to a join table is moot if you follow the suggestion to use five standard fields in every table (zzID, zzCreator, zzModifier, zzCreationTS, and zzModificationTS). They take up little space, and FileMaker takes care of maintaining the data. You can use them to track down anomalies and bugs in the database.

Normalizing Data: What Goes Where

In addition to looking at the relationships among entities, a standard process of relational database design involves normalizing the data. This is a process whereby the data within tables is examined to see whether its logical structure can be simplified. Fully normalized relational database tables generally function more efficiently than non-normalized (or less normalized) tables. There is a widespread belief that normalizing data can use more storage space or even more processing resources, but experienced database designers tend not to agree with that belief.

First Normal Form: Eliminate Repeating Groups

Perhaps the most common type of repeating group is addresses or phone numbers for a contact. The Contact Management Starter Solutions demonstrates this problem: Each contact can have two addresses—one for home and one for work.

But what if someone has three addresses? And what if someone has two jobs? Some lucky people have two homes. There is no room for a third address.

A normalized database would move the address information to a subsidiary table. In that way, a contact could have any number of addresses. The need to create a subsidiary table and a relationship (which is very easy to do in FileMaker Pro) is the reason some people believe that normalized data is expensive. But the power of the related table is worth the cost. Not only is there no wasted storage and no arbitrary limit on the number of addresses to be stored, but you can add a Type field to each address so that people can flag each address as Home, Work, Weekend, and the like. There is no need to create and use (or not use) fields in the main table such as Work Phone, Weekend Phone, Mobile Phone, Voicemail Phone, and so forth.

tip

Repeating fields might be giveaways of violations of first normal form. Fields ending with numbers also are clues, as in Employee 1, Employee 2, and so forth, or Ingredient 1, Ingredient 2, and the like.

Second Normal Form: Eliminate Redundant Data

There is no reason to store the same data in two places. Doing so wastes space, and the two representations of the same data can easily get out of sync.

In the case of a subsidiary table of addresses for a contact, you would not store the name of the contact in both the main and the subsidiary table—doing so would violate second normal form. But if a different data item, such as the name to use in addressing correspondence, varies depending on the address, you would store that data twice. Sometimes people in the public eye are known by one name in town and by another name—perhaps just an initial for the first name—in a weekend getaway.

Another common situation arises with invoicing of items sold. The price of an item is stored in a table describing the item. To store the price again in an invoice table might violate the second normal form: storing the same data twice.

In this case, the data is not the same. If the product table contains the current price of a product, then the price that is stored in the invoice table is the price that was charged when the item was invoiced—a different piece of information. Storing the price in two places (with two meanings) means that you can, indeed, have the two values get out of sync—which is what you want. You want to be able to change the product price without inadvertently changing already-invoiced prices.

note

With FileMaker Pro, you can modify this stricture. By using calculation fields, you make it clear that the data you are using is derived. The main point of third normal form is not to create fields that you fill yourself with a derivable computation.

Third Normal Form: Eliminate Fields Not Dependent on the Key

Third normal form is a way of saying that you should not store data that can be derived from data that you do store. Do store

the data if the derivation might differ over time as related data changes, but do not bother with unchanging data.

FileMaker Extra: Complex Many-to-Many Relationships

Most of the examples in this chapter involved fairly simple, commonly found data-modeling problems. But in the real world, matters can get quite complex. Some problems are hard to model in the language of relational databases. Others involve concepts you've already seen, but in more complex forms.

Let's say that you have to sketch out a database system for a trucking company. The company needs to track which drivers are driving which trucks and where they're driving them. After some thought, you decide you're dealing with three entities: Driver, Truck, and Route. A route consists of a start location, a destination, and a number of miles driven.

With the entities fixed, you start to think about relationships. Driver and Truck seem to have a many-to-many relationship: One driver can (over time) drive many different trucks for the company, and one truck will be driven by many drivers (again, over time). Driver and Route also seem to have a many-to-many relationship. Thus, you have three entities, and three many-to-many relationships.

Earlier you learned how to resolve a many-to-many relationship. For any two entities that have a many-to-many relationship, you add a join entity between them that holds a primary key from each side of the relationship. You might take that to suggest that you should interpose a join table between each of the many-to-many relationships, but if you take a high-level view of things, you can add a single join table that manages all the many-to-many relationships.

These three associations (Truck-Driver, Truck-Route, Driver-Route) are not independent of each other. They all happen at the same time. When a trucker drives a truck from point A to point B, all three associations happen at once. Why not put them all into just one record? That's the right answer, as it turns out, and it implies the structure shown in Figure 5.10.

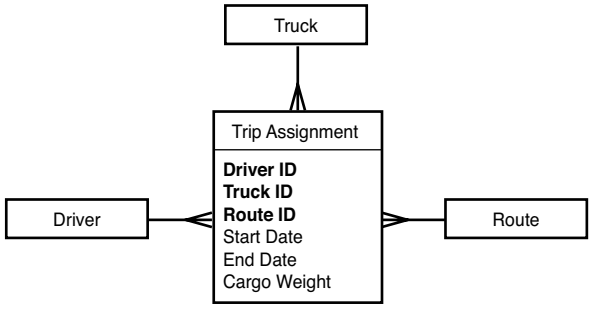


Figure 5.10
The trucking system ERD with a single central join entity.

What you're dealing with here is not three many-to-many relationships, but a single "many-to-many-to-many" relationship. This kind of structure is sometimes referred to as a *star join*. The central entity in a star join (which in the example stores information about the associations between a truck, a driver, and a route) is sometimes called a *fact table*. If you see a number of join entities in your diagram that are symmetrical, as they are here, and seem to capture different pieces of the same data, you might want to think about whether you have a star join of some kind on your hands.

6

WORKING WITH MULTIPLE TABLES

Multitable Systems in FileMaker Pro

This chapter shows you how to take the ideas from Chapter 5, "Relational Database Design," and use them to build FileMaker database systems. You learn how to use FileMaker to create database systems that model the types of relationships covered in the preceding chapter.

Chapter 5 laid out a set of design concepts that centered around the ideas of entities, their attributes, and the relationships between entities. In FileMaker Pro, you generally represent a database entity ("student," for example) as a table. You generally represent an entity's attributes ("first name," "year of graduation," for example) by the fields of that table. And you create relationships among tables with FileMaker's Relationships Graph.

The focus of this chapter is on implementing the relational model. Further chapters on layout design help you with additional interface features and modifications to the basic layout.

note

The sequence of steps involved in building a database solution such as this varies from person to person. Some people prefer to build the tables first and then add the fields and their options. Other people, knowing the basic fields used in each table, prefer to create a table with its fields and options and then move onto the next one. Do whatever makes most sense to you, realizing that FileMaker database development, more than most database development, is an iterative process. The only necessity is that to create a relationship in the Relationships Graph, you must have the two tables involved and the two fields that you want to use as the primary and foreign keys.

Creating a One-to-Many Relationship in FileMaker

In the Small Task Management database that you build in this chapter, there are going to be three basic tables (this is a multitable solution). Each table will have the five basic fields previously mentioned. Each is auto-entered by FileMaker Pro, and none can be modified by the user during data entry. To show the database structure more clearly in this chapter, only zzID is shown in the figures.

- **zzID—**This is the auto-entered serial number. Because it is auto-entered and unique, you can use it as the primary key of each record in each table of the database.
- **zzCreator—**This is the name of the creator of the record. It could be set to the account name if you prefer.
- **zzModifier—**This is the name of the last modifier of the record. It could be set to the account name if you prefer.
- **zzCreationTS—**This is the creation timestamp of the record.
- **zzModificationTS—**This is the last modification timestamp of the record.

note

If you are using FileMaker Pro Advanced, you can create a table with these fields and then copy and paste it for each new table you create. Alternatively, you can create the fields in a single table and copy and paste them into other tables.

These are the tables:

- **Tasks—**Each task has a name, a type of task, and a due date. Each task belongs to a single project.
- **Projects—**Each project has a name. It is related to zero or more tasks.
- **Contacts—**These are people or organizations who are assigned to tasks. Each contact can be assigned to zero or more tasks; each task can have zero or more contacts assigned to it.

Each entity in an entity-relationship diagram (ERD) generally translates into one table in a FileMaker system. The following sections describe how to begin.

Creating the First Table in a Multitable System

When you create a FileMaker database for the first time, you get a single table with the same name as the database file, along with a single layout that also has that name. If you create a new database called Small Task Management, you get within it a single table, also called Small Task Management.

By default, FileMaker Pro automatically switches to the spreadsheet-like table view shown in Figure 6.1. (You can change this in Preferences using the General tab and the "Use Manage Database dialog to create files" setting.)

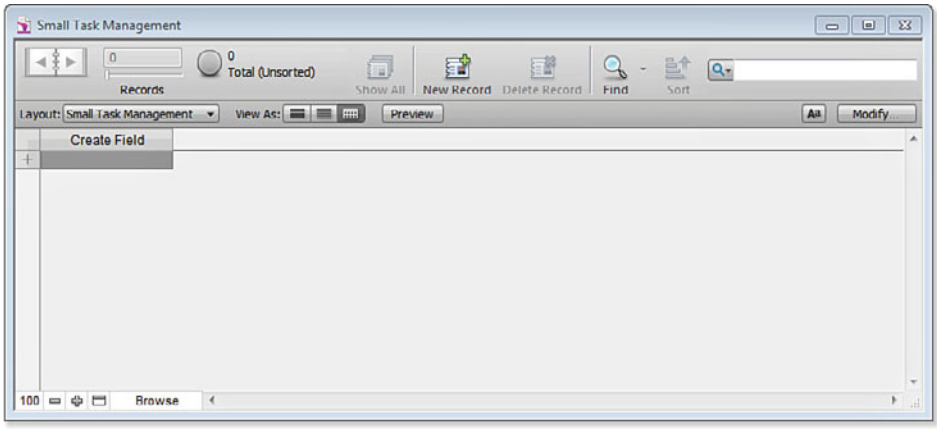


Figure 6.1 FileMaker Pro automatically creates your first table.

You create new fields by typing in the top row and entering the field name to replace Create Field. Press Tab. You will see that you now have a new field at the right that is named Field 2, as shown in Figure 6.2. If you use the Return key after entering a field name, you commit the name and stay in the same column (that is, no Field 2 is created).

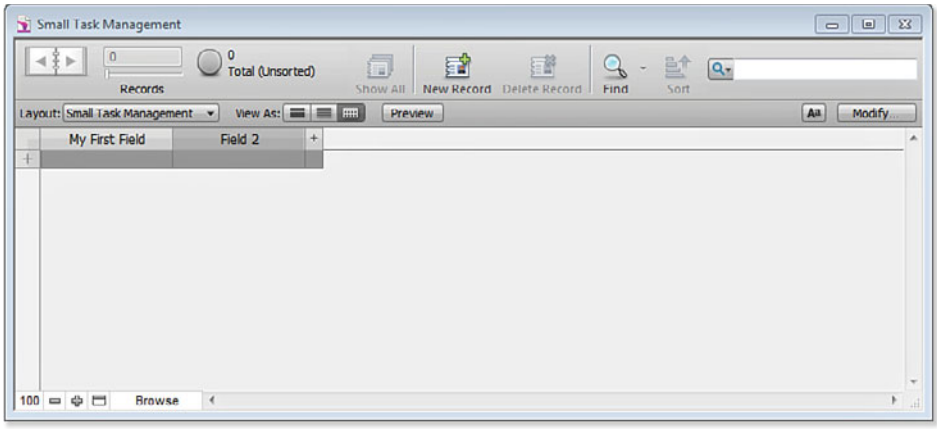


Figure 6.2 Create new fields.

When you click in a field title, a down-pointing arrow lets you choose field types and options, as shown in Figure 6.3.

You can also manage fields and tables from the Manage Database dialog found in the File, Manage submenu, as shown in Figure 6.4. The Manage Database dialog offers you more options. Use the table view or Manage Database dialog, depending on your preference at the time.

tip

Many people use both interfaces in different circumstances.

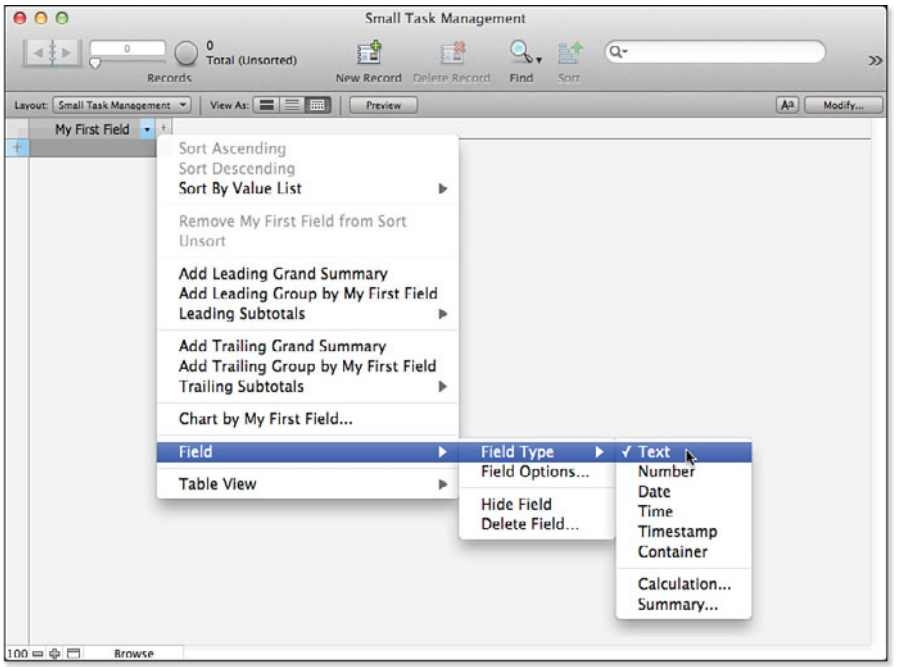


Figure 6.3
Set the field types and options.

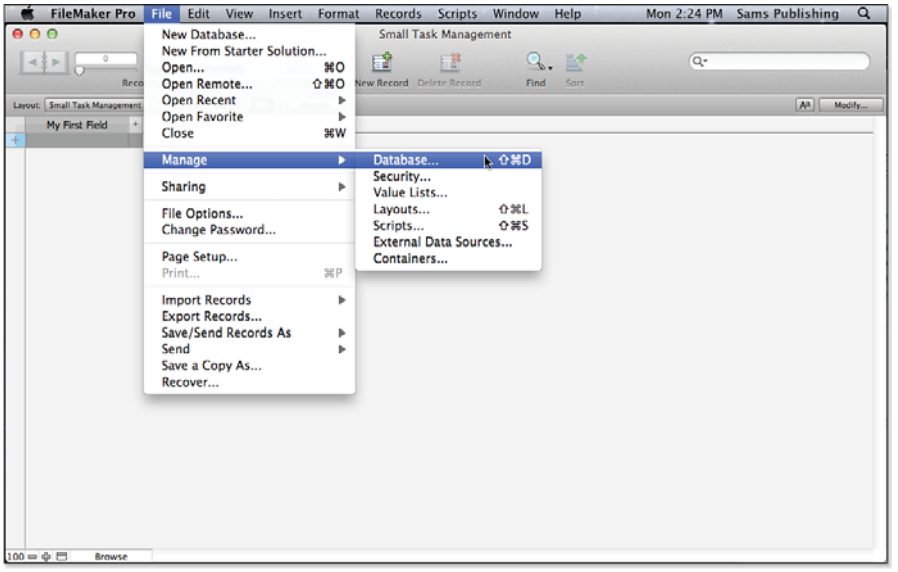


Figure 6.4
Use the Manage Database dialog.

One circumstance in which you probably want to use the Manage Database dialog is if you want to create several tables, as is the case here. For a multitable solution, the first step might be to rename the first table. Figure 6.5 shows the first table selected in the Tables tab of the Manage Database dialog; its name is about to be changed to Tasks.

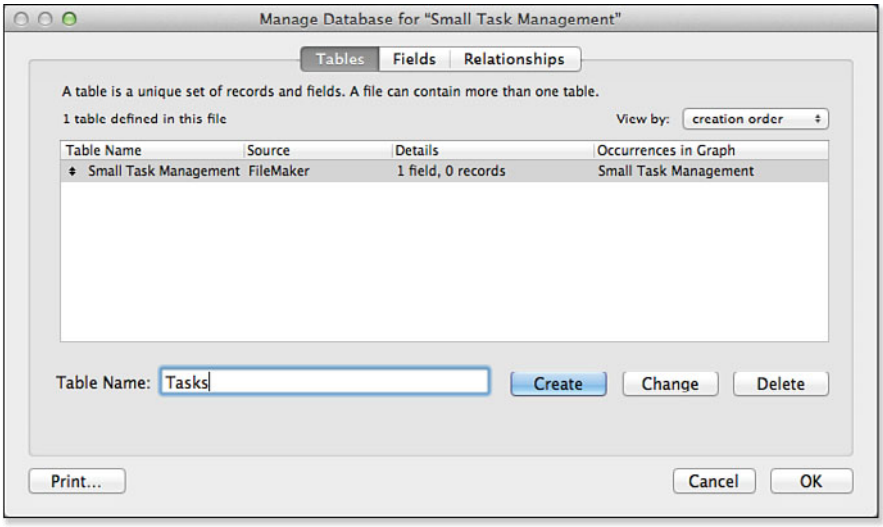


Figure 6.5
Rename the first table.

In Figure 6.6, you can see four fields created in the newly renamed Tasks table. They are the primary key, zzID; the due date for the task; the name of the task; and the foreign key that will identify the project in the project table when that is created. Later, there will also be the other four basic timestamp and identification fields described previously.

For a refresher on the details of creating fields within a single table in FileMaker, **see** *Chapter 3, "Defining and Working with Fields and Tables."*

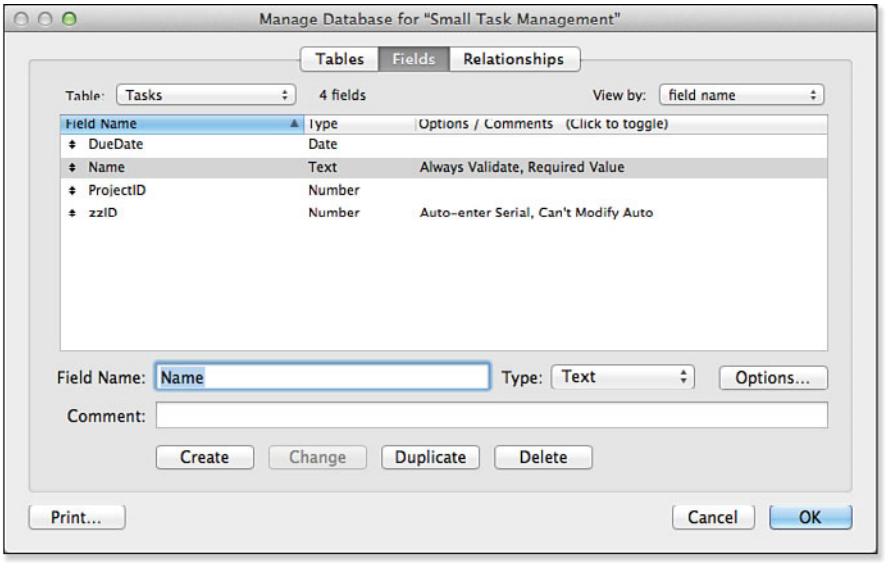


Figure 6.6
Field definitions for an initial table in a database of Tasks information.

Adding a Table to a Multitable System

You've taken care of the Tasks table. To add a table for Projects, stay in the Manage Database dialog, but switch to the Tables tab. You see just one table, which in this example is called Tasks. To add a new table, type the name in the Table Name box and click Create. The new table is added to the list. This will be the Projects table, as shown in Figure 6.7.

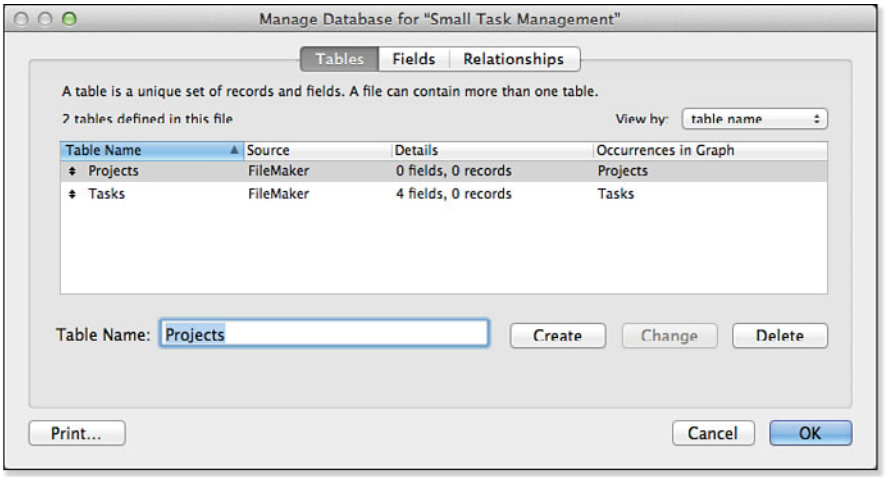


Figure 6.7
FileMaker's Tables view, showing a database with multiple tables.

You're now free to add fields to the new table. Figure 6.8 shows the basic starting fields for the Projects table.

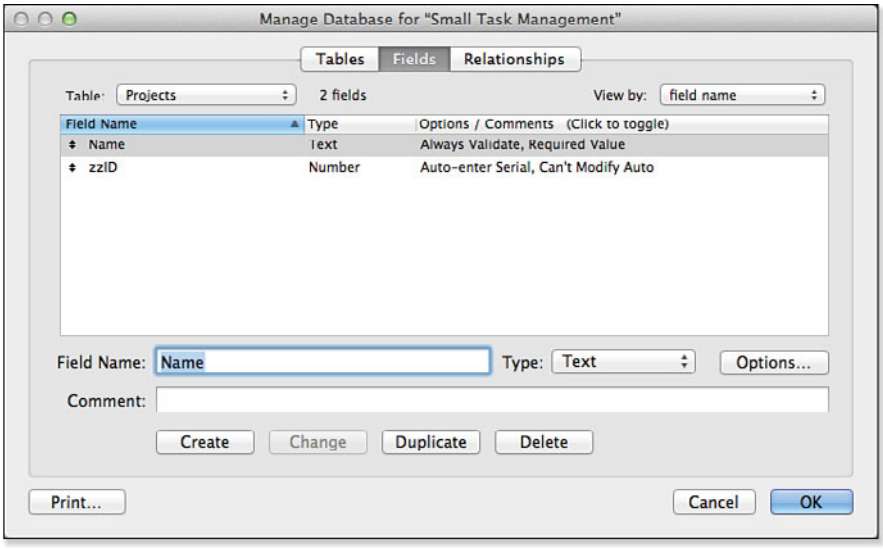


Figure 6.8
Field structure for a table of projects.

The Projects table automatically has a primary key (as do all tables that have an auto-entered zzID serial number). It does not need a foreign key as the tasks table does. The next section shows you why.

For a refresher on primary and foreign keys, **see** *"Understanding the Role of Keys in Database Design,"* **p. 161.**

Adding a Relationship

You now have two tables, as well as the primary and foreign keys that good database design demands. To create a relationship between these two tables, move to the Relationships tab of the Manage Database dialog. This window, known as the Relationships Graph, should have a couple of graphical elements already displayed. Each one represents one of the database tables that exist in this database. These elements are known as *table occurrences*. Each shows the name of the table it represents, along with that table's fields.

Adding a relationship between these two table occurrences is simple: Draw a line from ProjectID in the Tasks table to zzID in the Projects table. You can also draw it in the other direction, from zzID in the Projects table to ProjectID in the Tasks table. Relationships have no direction in FileMaker Pro. You should see a line extend from one table to the other. When you release the mouse, FileMaker creates the relationship and displays it as a link between one or more match fields at the top of the table occurrence pair. Figure 6.9 shows how the Graph will look as a result.

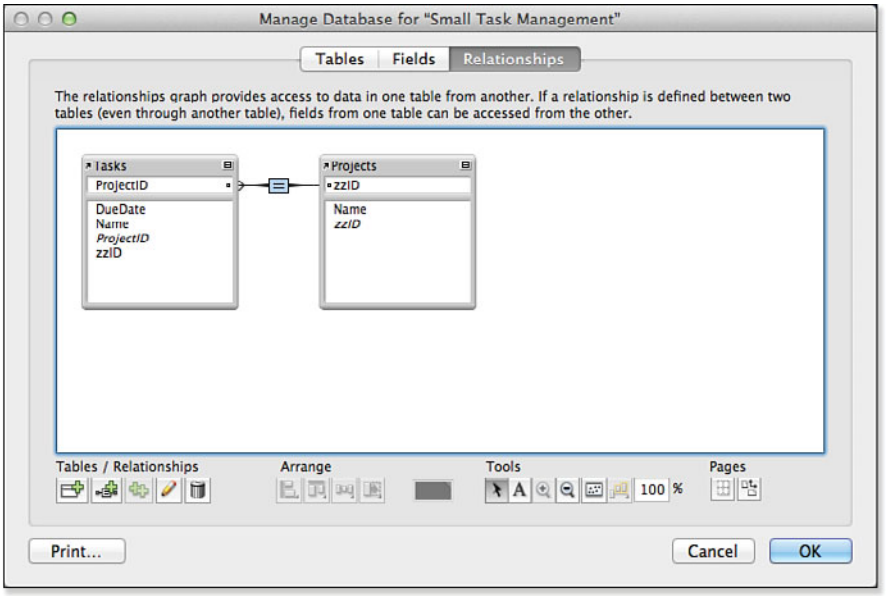


Figure 6.9 FileMaker's Relationships Graph, with a relationship between two table occurrences.

You might have noticed the crow's foot at the end of the relationship line where it touches the Projects table occurrence. This is none other than the indicator that you're accustomed to seeing on the ERDs from the preceding chapter. It's intended to indicate the "many" side of a one-to-many relationship.

At this point you've seen how to add a new table to FileMaker's default one-table database configuration and how to define a one-to-many relationship between two FileMaker tables. The next sections clarify some important points about multitable systems.

note

Be warned, though! FileMaker provides this graphical adornment as a kind of a hint or guess about the relationship; it might not always be accurate (although, in this case, it is). You find out more in the next section, where the creation of key fields in FileMaker is discussed.

Working with Keys and Match Fields

You should remember from Chapter 5 that keys are fields within tables—fields that are essential elements in forming the relational structure of a multitable system. FileMaker takes a somewhat broader view of keys than many other relational databases, and for that reason these fields are referred to as *match fields* when you're working in a FileMaker context. A match field in FileMaker

is any field that participates in a relationship between two FileMaker tables. Primary keys and foreign keys fit this definition, of course, but so do a number of other types of fields explored in the next chapter.

For more details on the broader uses of match fields in FileMaker Pro, **see** *"Relationships as Queries,"* **p. 194,** *as well as other sections of Chapter 7, "Working with Relationships."*

Key fields (which form the structural backbone of the system) need to play by some special rules—especially primary keys. Consider the current example, the Small Task Management database system, and consider the zzID field in the Projects table. This field has been identified as the primary key for the Projects table. To play the role of primary key, it must have a unique value. This was done automatically when you made it an auto-enter serial number in the Field Options dialog; this was further implemented by forcing it to have a unique value in the Validation tab and not allowing the user to modify it during data entry.

Cardinality in the Relationships Graph

This discussion provides an opportune moment to look again at that crow's foot that FileMaker so cleverly applied to the Projects-Tasks relationship created earlier. FileMaker looks at the field definition options to try to determine the cardinality of a relationship. Any field that is either defined to be unique or has an auto-entered serial number is assumed by FileMaker to be the "one" side of a relationship. Lacking either of those characteristics, it's assumed to represent the "many" side. That, in brief, is how FileMaker determines how to draw the cardinality indicators (that is, the crow's foot) in the Relationships Graph. It's a useful indicator, to be sure, but not bulletproof, and is really just advisory. The cardinality indicator neither creates nor enforces any rules, and it can't be changed from FileMaker's default "guess" value. It simply tells you what FileMaker thinks is going on.

The Database So Far

At this point, you have created two tables and added fields to each of them. You have used the Manage Database dialog to create a relationship between them. You have also created two default layouts—or at least FileMaker Pro has done it automatically for you while you were working in the Manage Database dialog. The order of the fields in the layout reflects the sequence in which you created them.

You can create a new record by choosing New Record from the Records menu in Browse mode. The serial number (1) is automatically filled in, and you can type a project name, such as **Chapter 6**. Figure 6.10 shows the Tasks layout created for you with a single record entered.

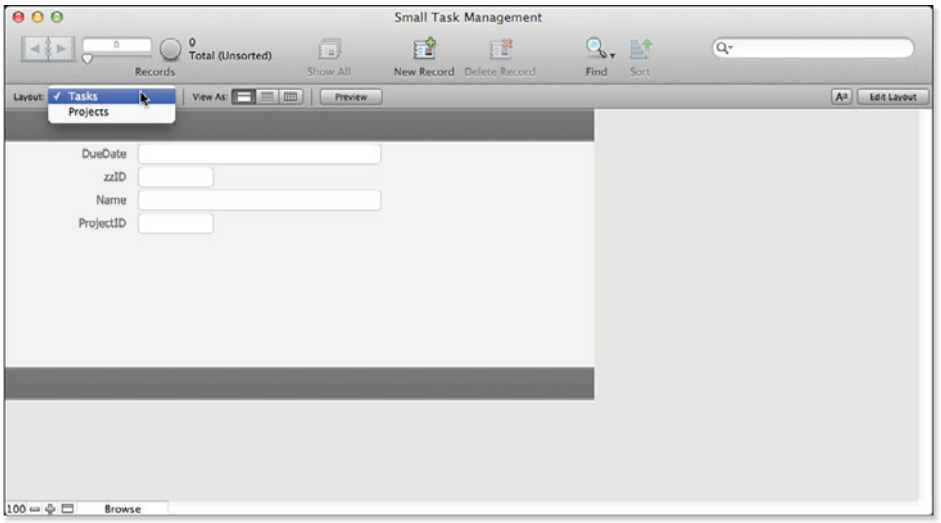


Figure 6.10 FileMaker Pro has created two layouts for you.

Working with Related Data

So far in this chapter you've learned how to create additional tables in a FileMaker system and how to build relationships between those tables based on well-constructed match fields. The following sections show you how to begin to use your relationships to work with and create data in multiple tables at once.

Using a Portal to View Related Child Data

The Small Task Management database system has two tables in it now, and there is a relationship between Projects and Tasks. In this example, a single record has now been created in the Projects table.

Now it is reasonable to create tasks for the project and to display them. You can do this with a FileMaker layout element called a *portal*. A portal lets you display multiple related records for a parent record in a layout.

For additional discussion of the parent/child naming convention, **see** *"Creating a One-to-Many Relationship in FileMaker,"* **p. 170.**

To create a portal on the Projects layout, choose the portal tool from the Status toolbar tools in Layout mode, as Figure 6.11 shows, and draw the portal in the approximate location you want it on the layout.

note

Although nothing prevents you from using a portal to display a one-to-one relationship, its design is optimized for the display of multiple records in a one-to-many relationship. The context or base table for the layout in which a portal is placed should be the "one" side of the relationship; the portal will be the "many" side. You can have multiple portals in a single layout, but they will all be the "many" sides of various relationships of which the base table is the "one."

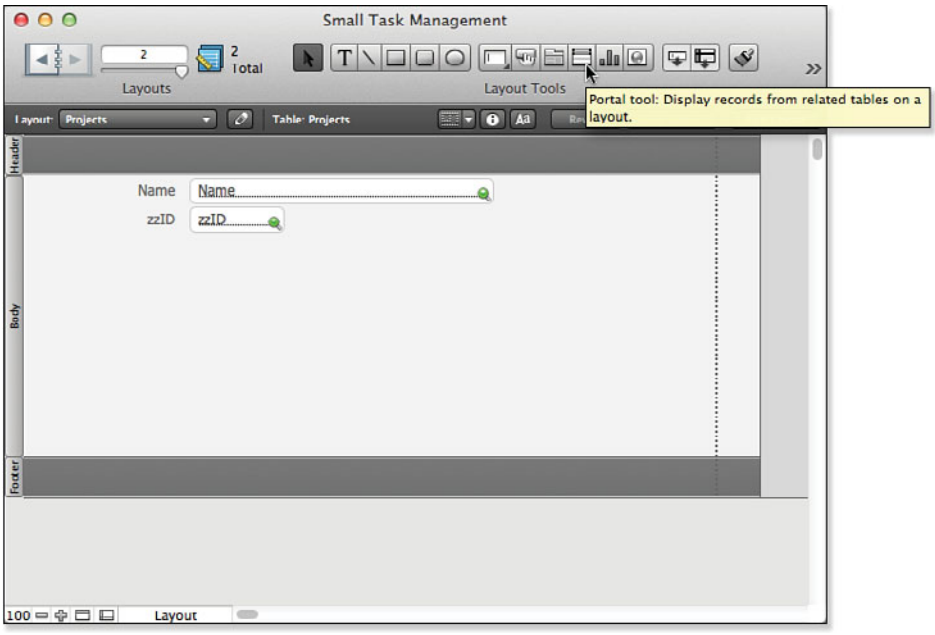


Figure 6.11
This is a layout in Layout mode, showing the FileMaker portal tool.

Click once on the tool, and then, on the layout, drag out a box wide enough to show the Task information and release the mouse. You get a dialog asking for details about the portal's contents, behavior, and display. The dialog is shown in Figure 6.12.

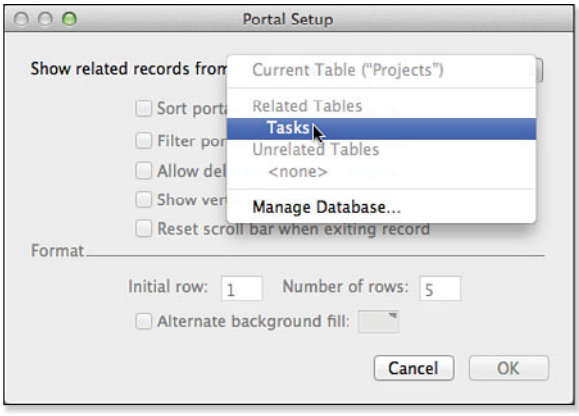


Figure 6.12
FileMaker's Portal Setup dialog.

In general, when you set up a portal for the first time, you need to do the following:

- Choose a table occurrence from which to display data.
- Choose additional portal options.
- Choose data fields for display in the portal.

More details on each of these steps follow.

First, you need to specify where the portal gets its data. In the Portal Setup dialog, the Show Related Records From list enables you to choose which table to draw data from. The list is divided into sections: one for related tables and one for unrelated tables. (The Relationships Graph determines the question of whether a table is related or unrelated.) In the current example, for a portal on a layout in which the table context is the Projects table, there should be only one available choice in the menu: the Tasks table, which is the only other table related to Projects in the Relationships Graph. By choosing Tasks from this menu, you're instructing FileMaker to show you all Tasks records related to the currently visible Projects record.

The Portal Setup dialog contains a number of other choices as well. For now, you can opt to display just four portal rows and put a vertical scroll bar on the portal so that you can scroll down if a project has more than four tasks. You can also apply coloring or striping to the portal if you choose.

FileMaker also displays a dialog at the end of the portal-creation process, asking which fields from the related table you want to show in the portal, as shown in Figure 6.13.

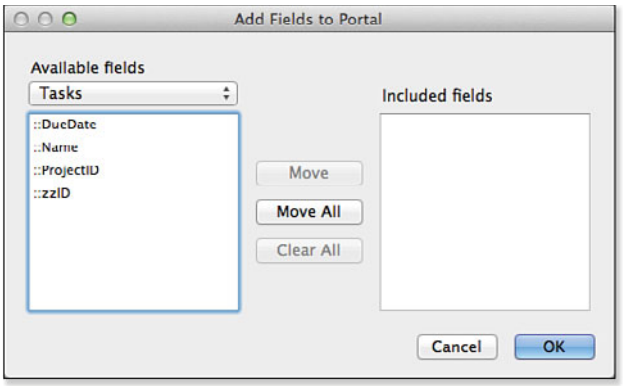
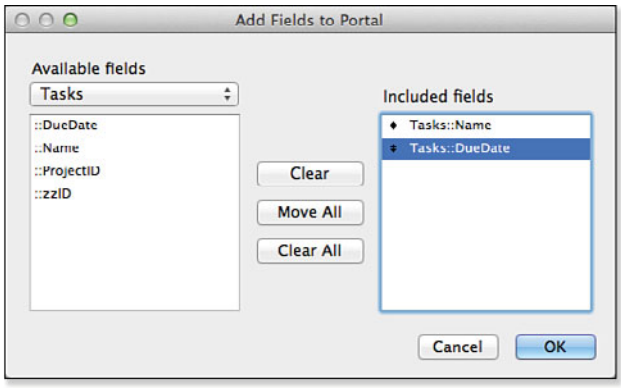


Figure 6.13
The Add Fields dialog shows the available fields in the related table.

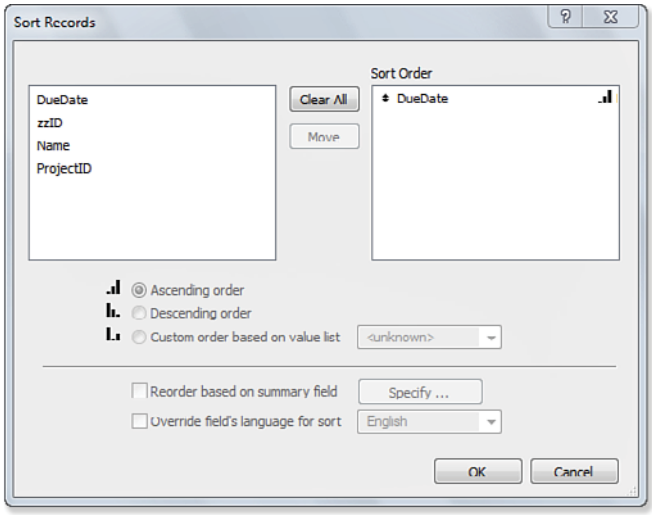
You can double-click field names or select them and click Move to add fields to the portal. As with everything else in the portal design, you can come back later to make modifications if you want. Note also that the pop-up menu at the upper left of the Add Fields dialog lets you select other related table fields to display in the portal. After you have added fields to the portal list on the right, you can rearrange them by dragging the double-headed arrows up or down (see Figure 6.14).

Figure 6.14
Reorder portal fields if necessary.



You can reopen the Portal Setup dialog by double-clicking the portal itself. Doing so lets you make other changes. You could, for example, click the Sort button to sort the data displayed in the portal, as shown in Figure 6.15. Here, the data is sorted in ascending order by date due (earliest first).

Figure 6.15
You can sort portal data.



In addition to sorting the portal, you usually want to add a scroll bar to it in the Portal Setup dialog. A check box lets you do this, as shown in Figure 6.16.

Now that you have set up your portal, you are ready to enter related records.

note

Fields are automatically added to a portal row the first time the Portal Setup dialog is shown, immediately after you have drawn the portal. Thereafter, you need to add them manually.

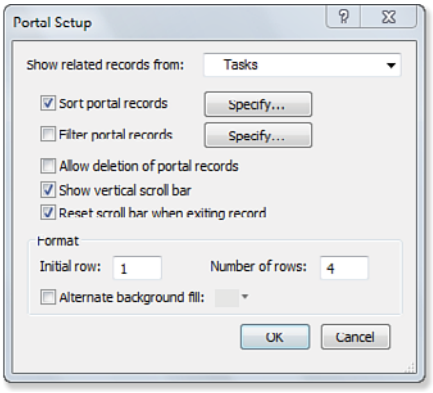


Figure 6.16
In the Portal Setup dialog, you can control where the data comes from, how it is sorted, and whether there is a scroll bar.

Using a Portal to Add Related Records

You can use portals for data entry as well as data viewing. You can even configure the portal and its underlying relationship so that a user can add Tasks to a Projects record by typing directly into the portal rows. To accomplish this task, you need to edit the relationship between Tasks and Projects. On the Relationships Graph, double-clicking the relationship line between the two tables brings up the Edit Relationship dialog, shown in Figure 6.17.

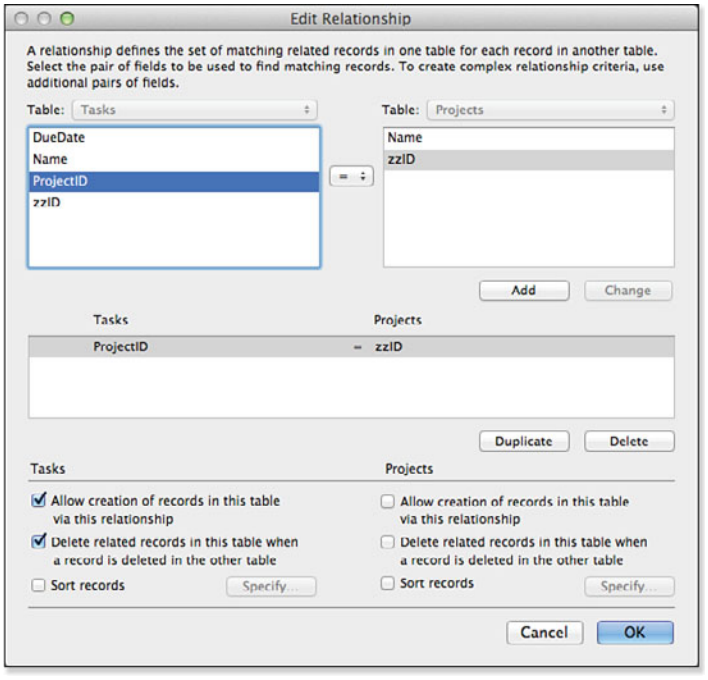


Figure 6.17
In the Edit Relationship dialog, you can edit individual relationships in the Relationships Graph.

For each table participating in a relationship, there's an Allow Creation of Records in This Table via This Relationship check box under it. If you check this box on the Tasks side of the dialog, you are able to create task records via this relationship. You also can choose to delete related records automatically, which in this case is a good idea.

Relationships in FileMaker have no direction, so you cannot tell which side a table will appear on in this dialog (the direction in which you drew the relationship actually determines it).

note

You cannot have a task without an associated project, so if the project is deleted, all its tasks should be.

If you check this option and return to the Projects layout in the parent table, you'll discover that you can now click an empty row of the portal and type in a task. If you do so, you automatically create a second, empty row for yet another task as soon as you have clicked out of the field into which you entered data. When the check box to allow creation of records is checked, you always have an empty row in the portal for data entry, as shown in Figure 6.18. With portals, it's easy to view, create, and manipulate records on the "many" side of a one-to-many relationship.

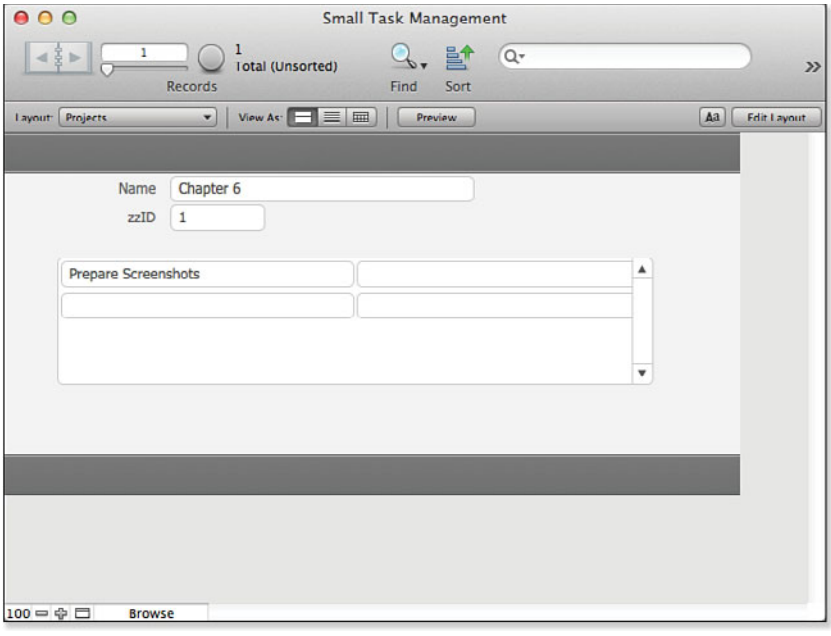


Figure 6.18
You can enter data in the portal for a related record.

You can add a feature to the portal to increase its usability: You can add a widget so that you can go to the related record. In Layout mode, select a widget from a Starter Solution or create one yourself and paste it into the portal row, as shown in Figure 6.19. Any graphic will do; at this point it is merely an image with no functionality.

With the widget selected, choose Button Setup from the Format menu to open the dialog shown in Figure 6.20. (Alternatively, just double-click the widget.)

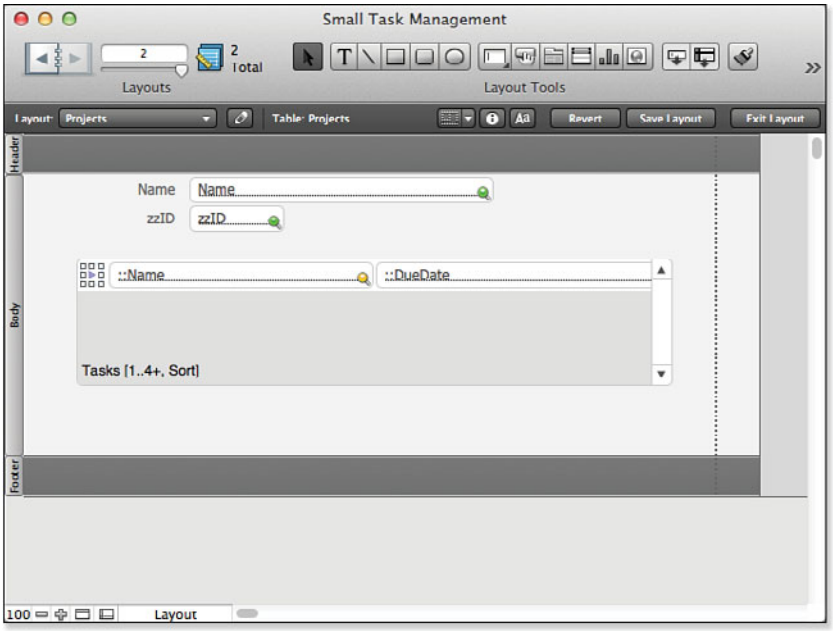


Figure 6.19
Add a widget to the portal row.

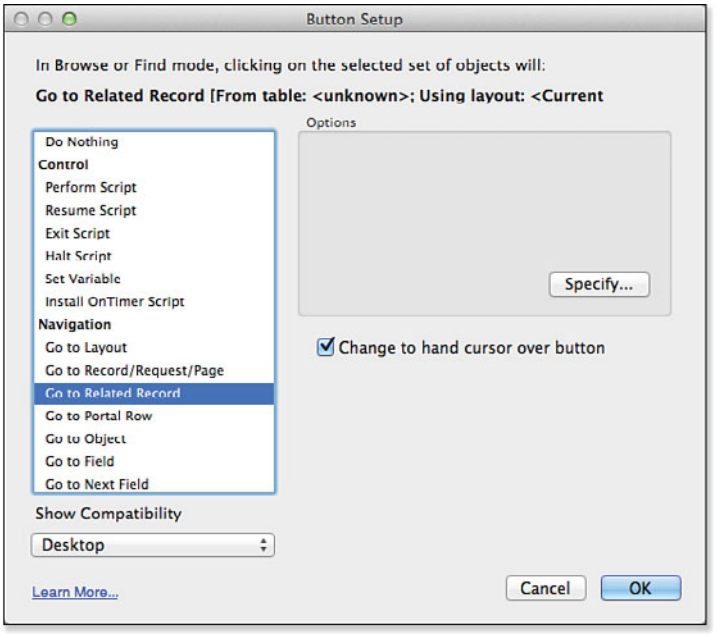
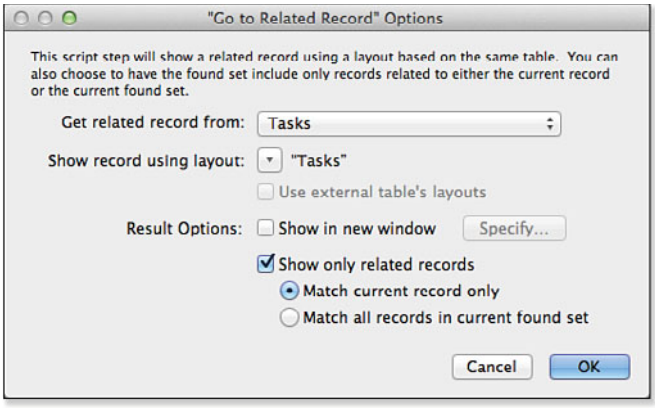


Figure 6.20
Use the Button Setup dialog to implement a button.

Choose Go to Related Record for the widget's action, as shown in Figure 6.20. Click Specify to open the Go to Related Record Options dialog, shown in Figure 6.21.

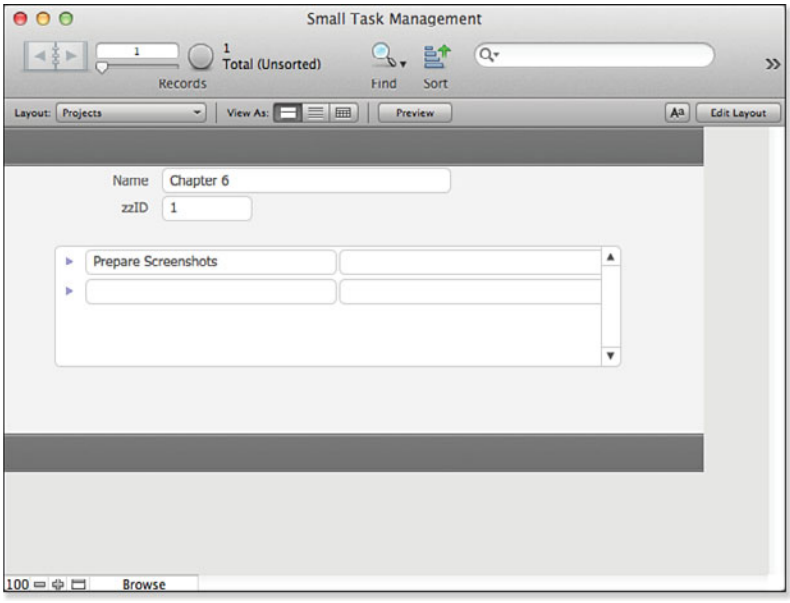
Figure 6.21
Set the table and layout to go to.



You will want to display records from the Tasks table. You can choose the layout to use (there is only one at this point, the default Tasks layout). Make certain that you use the check box at the bottom to show only related records.

Back in Browse mode, you will see your widget in the portal row; it is also in the next (empty) row at the bottom of the portal, as shown in Figure 6.22. If you click the widget in the first row (where you have entered data), you should go to the related Tasks record using the Tasks layout.

Figure 6.22
The widget lets you move to a related record in its own layout.



You have implemented a relationship in the database and implemented half the navigation needed to move around the relationship.

Working with Related Parent Data in a Child File

In the Tasks layout, you can make two improvements that will complete the process. First, add a merge field (Insert, Merge Field) in the header of the layout. From the Specify Field dialog shown in Figure 6.23, select Name from the Projects table.

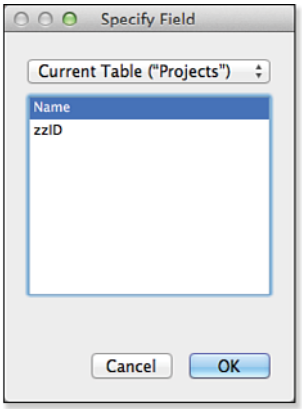


Figure 6.23
Add a merge field with the name of the related Projects record.

You can select the merge field and make it clickable by choosing Format, Button Setup. Repeat the process you used for the widget, but this time go to the related Projects record (you are already viewing the Tasks record), and use the Projects layout.

As shown in Figure 6.24, you now have the ability to move back and forth between projects and tasks.

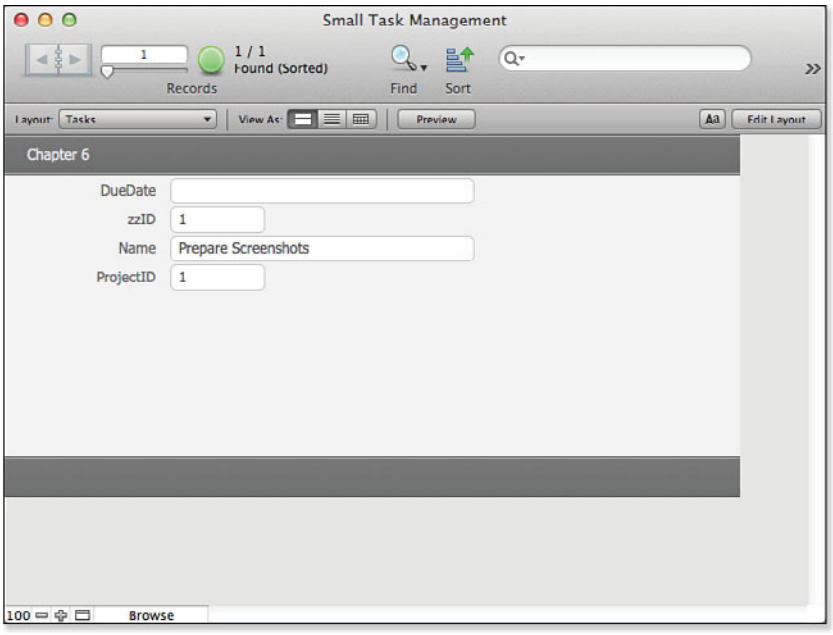


Figure 6.24
The Tasks layout now has a link to the related Project when you click the project name.

Creating a Many-to-Many Relationship

The preceding sections introduced you to most of FileMaker's fundamental tools for working with multiple related tables. Now it's time to extend those concepts and see how to use them to create a many-to-many relationship structure.

> *As noted in "Many-to-Many Relationships" in Chapter 5, you can also implement a many-to-many join using a multivalued field and checkboxes.*

Building the Structure

As described in the preceding chapter, a many-to-many relationship requires a join table. You have to add two tables to the Relationships Graph to implement a many-to-many relationship between tasks and contacts. The first is a Contacts table, which you can create exactly as you did the Projects and Tasks tables. For now, it is sufficient to give it a zzID and a Name field. Then create an Assignments table. This is the join table, and it needs three fields: its own zzID field, a TaskID field, and a ContactID field.

In the Manage Database dialog, create relationships between the Assignments table and the Contacts and Tasks tables. The process is the same as it was for relating Tasks to Projects. Your Relationships Graph should look like Figure 6.25 (subject to spacing variations).

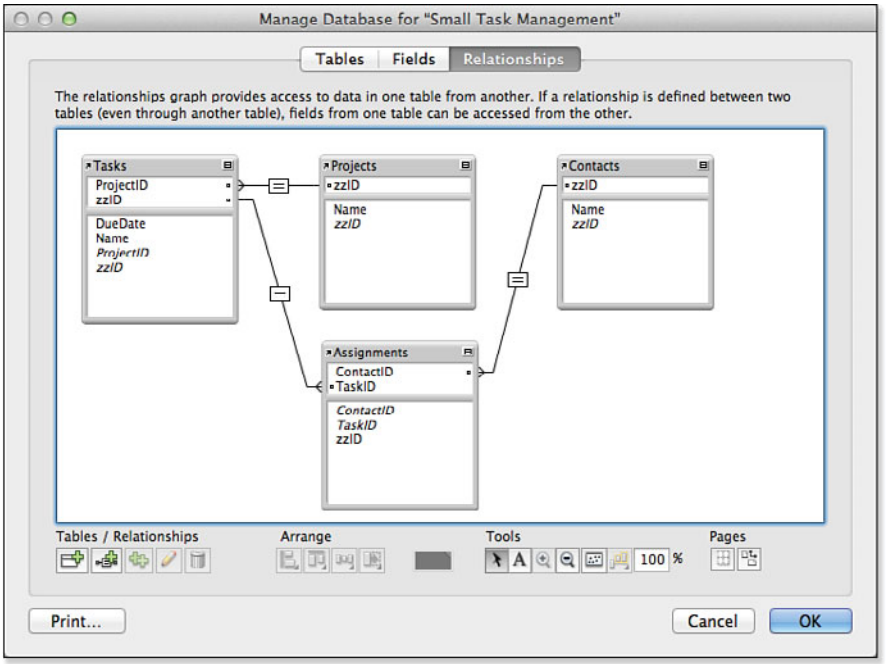


Figure 6.25
Add relationships for the join table.

Creating Value Lists

You can use value lists to let users choose the values for the fields in the Assignments table. This is a good idea because the table itself uses the meaningless ID numbers. You must have created the Contacts and Tasks tables to complete this step. To create a value list, choose Manage, Value Lists from the File menu or use the Manage icon in the status toolbar, as shown in Figure 6.26. When the Manage Value Lists dialog opens, click New to create a new value list.

note

The Manage icon shown in Figure 6.26 is not part of the standard status toolbar, but you can add it using View, Customize Status Toolbar. It is a very useful icon to add; many people add it to the Layout mode status toolbar.

The Edit Value List dialog shown in Figure 6.27 opens next. You want to use data from a field, so name the new value list **Contacts**, and then choose the first radio button; it opens the Specify Fields dialog.

On the Specify Fields dialog shown in Figure 6.28, choose the zzID field from Contacts. Choose to show data from a second field—the Name field in Contacts—at the right of the dialog. Sort the data on the second field. Create a value list for Tasks in the same way.

Figure 6.26
Manage the value lists.

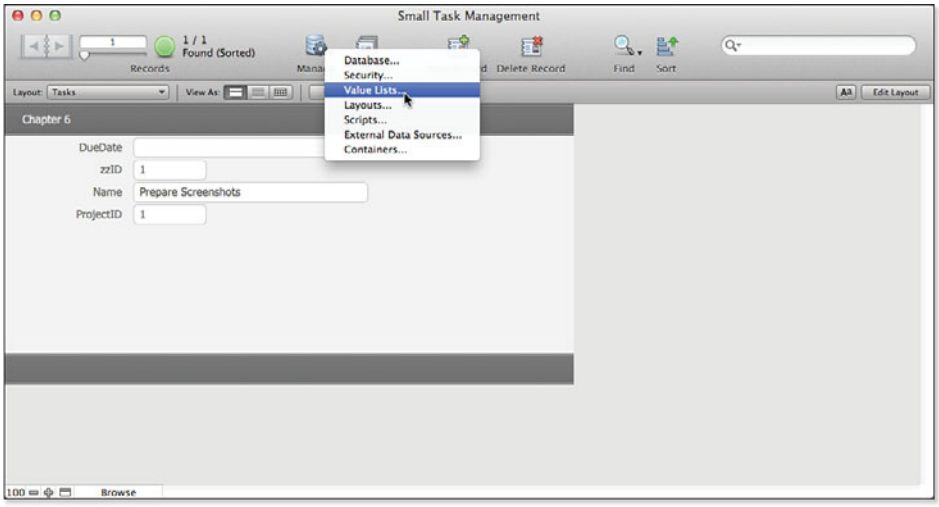
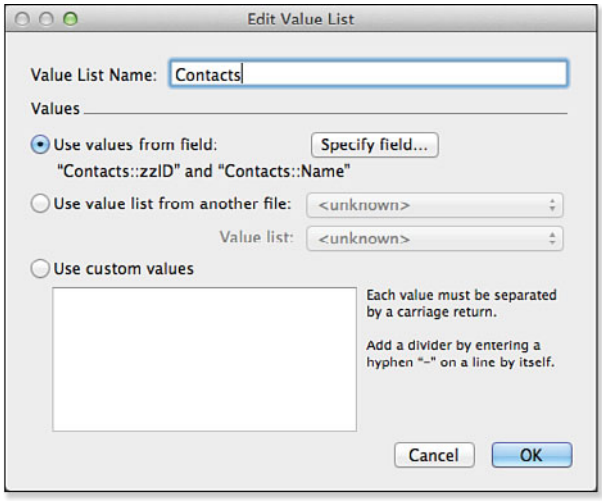


Figure 6.27
Use data from a field.



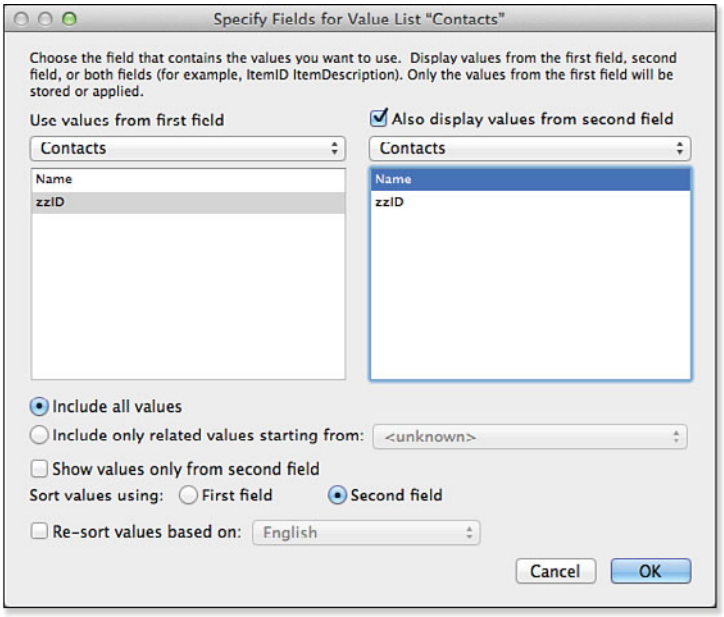


Figure 6.28
Choose the zzID and Name fields.

Designing the Interface

FileMaker Pro has created an Assignments layout for you automatically. You can modify that layout to make it more user friendly. The first thing to do is to make the ContactID and TaskID fields pop-up menus using the value lists you just created. Select the ContactID field in the default layout and set it to use the Contacts value list as a pop-up menu, as shown in Figure 6.29. Do the same for the TaskID field using the Tasks value list.

If you add some new tasks to the database, your interface might look like Figure 6.30.

tip

You will find out more about relationships and layouts throughout this book. The principles are quite simple. It is important to remember that, compared to pre–FileMaker 7 databases, relationships can span many tables in the Relationships Graph. As long as a path can be found between two tables, they are related and you can use the relationship to construct portals and related fields.

As you build layouts, it is frequently a good idea to implement both sides of navigation—from a parent to child records (frequently using portals) and from a child record to its parent (with a widget—often in a portal row—that uses the Go to Related Record action). If you don't do this, you might inadvertently construct traps that a user cannot easily get out of.

Another tip to remember is that frequently (but not always) it is the join table that should be the base table for a layout that will show a many-to-many relationship. Although the join table normally stays in the background, it has the information about both sides of the relationship.

Figure 6.29
Use the value list for data entry.

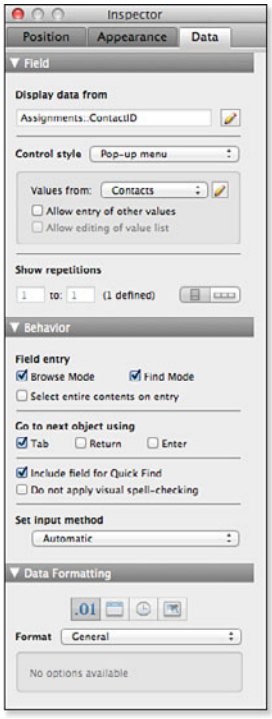
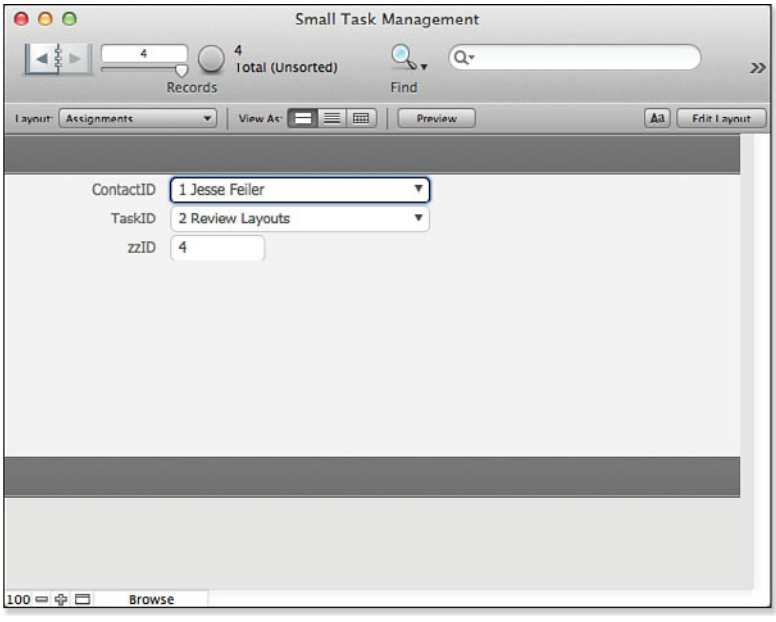


Figure 6.30
Your basic join table and its interface are complete.



Rapid Multitable Development

Working with the complex database schemas of a multitable file, or a solution composed of several such files, can sometimes be daunting. Using FileMaker Pro Advanced, you can import the definitions for multiple tables from one file to another. This schema import does not import any data, only the table and field definitions for selected tables. If you like to add a standard set of tables to most of your solutions (utility tables, logging tables, and resource tables), this procedure is now as simple as importing the table schemas from one file to another. It's also possible to copy and paste table definitions, between or within files.

The same is also true of field definitions. They can be copied and pasted, either between files or within files, allowing you to quickly reuse blocks of standard fields. The enhancements extend to ScriptMaker as well, where you can copy and paste scripts and script steps with ease.

Troubleshooting

Repeating Portals

I've created a portal, but instead of seeing a set of different records, I see that every row of the portal shows exactly the same data.

This problem indicates a mismatch of table occurrences. Specifically, it suggests that although the portal is set to look at records from table occurrence A, the fields you've chosen to display in the portal are actually from table occurrence B. Because it's possible to have several different table occurrences based on the same underlying table, it's possible to see the same field list for several different table occurrences. Nevertheless, if the portal and the fields displayed in it draw from different table occurrences, you probably won't get a meaningful display even if all the different table occurrences are based on the same underlying table.

Accidental Delete Restrictions

I set up a cascade-delete relationship between my Customer table and my Invoice table so that when I delete a customer, all related invoices are deleted as well. But when I try to delete a customer, I receive a message that I don't have sufficient privileges. I checked my privileges, and I do have delete privileges in the Customer table.

Check to make sure you have delete privileges on the Invoice table as well. To perform a delete operation successfully in FileMaker, a user needs delete access to any and all records to be deleted. If you have delete privileges for Customer but not for Invoice, the entire deletion operation is forbidden.

7

WORKING WITH RELATIONSHIPS

Relationships Graphs and ERDs

Chapter 5, "Relational Database Design," outlined some database theory that helps produce an ERD—an entity-relationship diagram that shows the fundamental building blocks of a database system and the ways in which they relate. In Chapter 6, "Working with Multiple Tables," you saw how to use FileMaker's relationship tools to turn an ERD into a working FileMaker database. This might mislead you into thinking that the Relationships Graph is really the same thing as an ERD, and that the relationships you build there match one-to-one with the relationships you sketch out on your ERD.

In fact, there's a lot more to relationships in FileMaker. The Relationships Graph certainly handles all the structural relationships present on an ERD. But there are many other ways to use relationships in FileMaker. The ERD-based relationships are the structural core of any FileMaker database (or any relational database), but this chapter takes you beyond the core and shows you some other ways you can use relationships in FileMaker. It also delves further into the features of the Relationships Graph and discusses different ways of organizing files, tables, and table occurrences in a FileMaker system.

note

Like the preceding chapter, this chapter uses the Small Task Management database. You can download it from the author's website as described in the Introduction. Although the name is the same as the database from Chapter 6, the downloadable database for this chapter incorporates the changes made here, so make certain that you look at the Chapter 7 version if you want to compare the downloadable database with the text of the chapter.

Relationships as Queries

Relationships can be more than classical database relationships in FileMaker Pro: They can be saved queries (something that does not exist in the world of SQL). In the Small Task Management database created in Chapter 6, you saw how to add a Tasks portal to the layout based on the Projects table.

The portal looks into the table of tasks and is based on the one-to-many relationship between a project and task set, based on a shared key called zzID in Projects and ProjectID in Tasks. But the portal also represents a kind of query, which says, "Show me all tasks for this project."

Relationships are often (but not always) simple matches of key values, but queries typically involve many components. You can create a relationship that is a saved query that will, for example, display not only all the tasks for a project, but also those tasks for a project that have a due date before or after a given date.


note

The distinction between a relationship and a saved query in FileMaker is in the purposes for which you use them: All are FileMaker relationships.

The database as it is now contains the basic relationship of tasks for a project. To be able to create a new relationship to implement this query, we need to add a field to the database and then delve into three new concepts in FileMaker relationships:

- Nonequijoins
- Table occurrence
- Multiple match

The new field to add is a date field that you can use in the relationship as a cut-off date before or after which data will be included (or excluded) from the Tasks table. This new date field is added to Projects so that it can control the date-sensitive relationship from Projects to Tasks. Fields that control relationships such as this one are frequently global fields: There is no need to have a different value for each record. Create a global date field in Projects and call it gDateLimit.


tip

Not using a global for this type of control field matters only if you want to allow multiple windows to be open using the same relationship at the same time and if you want each to be able to have its own control value.

Nonequijoins

In the Relationships Graphs shown in Chapter 6, all the relationships have a box with an equal sign in the middle of the line connecting the two table occurrences. To explore what that means, you can double-click that box to edit the relationship. Figure 7.1 shows the Edit Relationship dialog.

This relationship is built between a ProjectID in Tasks and a zzID in Projects. The match criterion is based on equality, meaning that records match (and hence are displayed in a portal that shows records from this relationship) if and only if the ProjectID in Tasks is exactly equal to the zzID in Projects. This is the correct behavior for the structural relationship represented on the ERD. Such a relationship, based on equality, is called an *equijoin*.

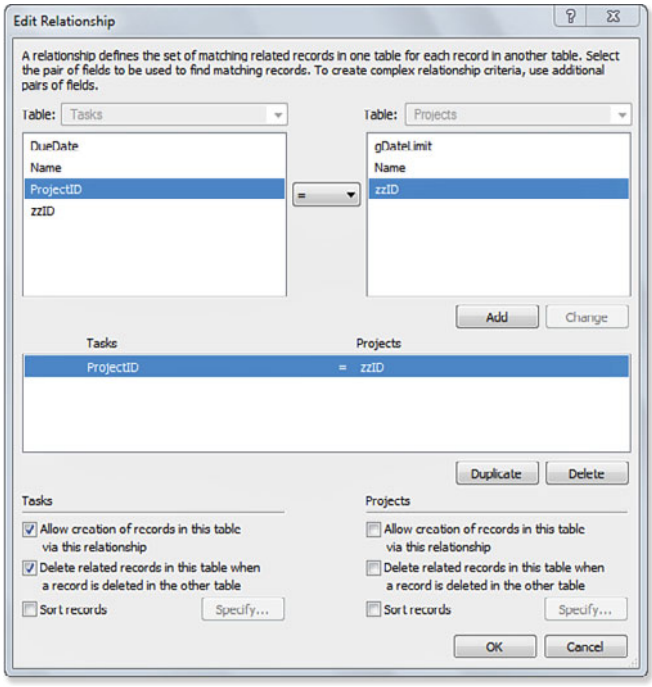


Figure 7.1
FileMaker can use any of seven different operators to compare match fields.

The upper part of the Edit Relationship dialog is the place where the match actually is defined. Notice, in Figure 7.1, that equality is not the only operator available for defining a match. In fact, you can build relationships based on combinations of any of the seven comparison operators.

To build the new relationship that implements the query showing you a project's tasks that are due before or after gDateLimit, you need to build a nonequijoin relationship that uses < or > to match to the date field in Tasks. You could update the existing Relationships Graph to modify the relationship between Tasks and Projects, but it makes more sense to add a new relationship that implements this query while leaving the existing one intact because both relationships have useful but different purposes in the database.

Adding a Table Occurrence to the Relationships Graph

The Relationships Graph displays table occurrences and the relationships between them. A table occurrence is based on an actual table (that is, one that appears in the Tables tab of the Manage Database dialog or in an external data source), but it is not the table itself. When you create a new table in the Tables tab, a table occurrence with that name is added to the Relationships Graph, so it is easy to think that the occurrence is the table itself.

A single table can have multiple table occurrences in the Relationships Graph. Indeed, there are certain cases in which you need to do this. Paths in the Relationships Graph must be unambiguous. Consider the case of three tables with three table occurrences in a Relationships Graph: call them

table/table occurrence A, B, and C. If there is a relationship from A to B and another from B to C, you have established a path from A to C, and it is unambiguous.

Now add table and table occurrence D. You can create a relationship from A to D with no problem. But if you attempt to create a relationship from D to C, FileMaker Pro will not let you. To do so would create a second path from A to C using the A-to-D relationship and the D-to-C relationship. The criteria for the intermediate relationships almost certainly will be different, so if you traverse the A-B-C path, the specific data you retrieve will be different than if you traverse the A-D-C path.

The solution is to create a new table occurrence based on C. Call it C2 for the moment. Now you have two unambiguous paths from table A to table C. One goes through the table occurrences A-B-C, and the other goes through the table occurrences A-D-C2. In both cases, you can access data from table C from table A, but by using distinct paths, you can control the relationships that are used.

In the current example, you want a new view of Tasks from the perspective of Projects based on a nonequijoin match of dates. Therefore, you need to add a new occurrence of the Tasks table to the Graph. (All the existing table occurrences were automatically added when you created the underlying tables.) To add the new table occurrence of Tasks, open the Relationships Graph in the Manage Database dialog and click the Add Table Occurrence icon in the lower-left corner (it is the leftmost icon in the bottom row). Figure 7.2 shows the resulting Specify Table dialog.

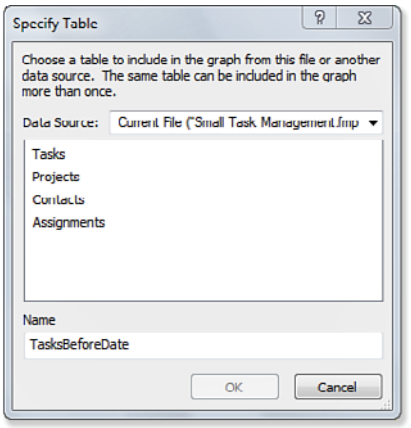


Figure 7.2
You can add a new table occurrence to the Relationships Graph.

In the Specify Table dialog, choose a source table to include in the Graph. In this case, you want to add another occurrence of Tasks. At the bottom of the box is a place for you to name the table occurrence. Because the original occurrence of the Tasks table is already named Tasks, you need a new name. FileMaker automatically generates a name with a number added to the last occurrence name. We recommend a name that says something about the way the new relationship will be used. In this case, TasksBeforeDate fits the bill. Figure 7.3 shows the Relationships Graph with the new table occurrence, as well as the gDateLimit field you added to Projects.

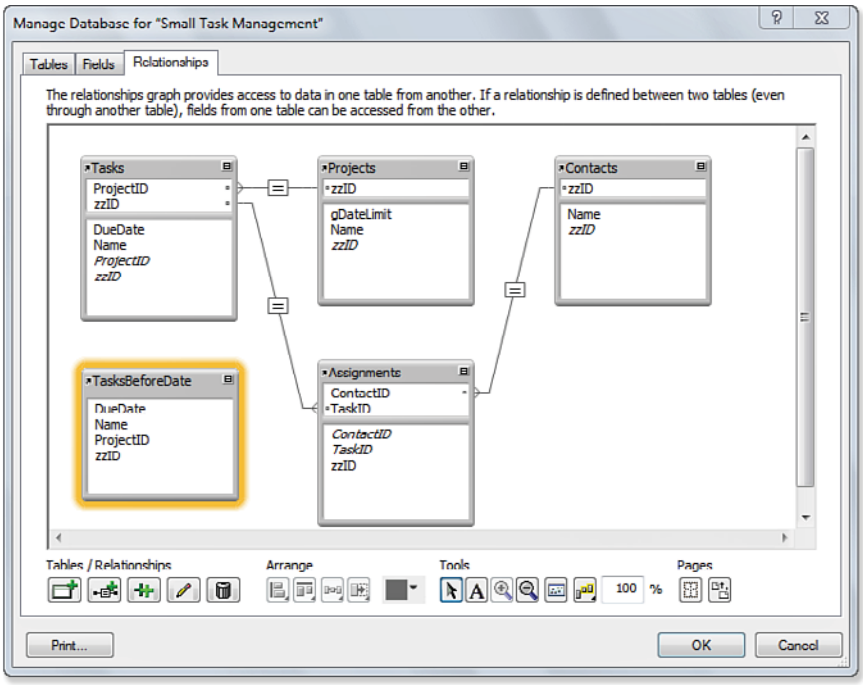


Figure 7.3
A second occurrence of the Tasks table has been added to the Graph.

All that's left is to create a relationship from Projects to this new table occurrence, which will incorporate the gDateLimit field into the match criteria.

Defining a Relationship with Multiple Match Criteria

Chapter 6 showed you how to define new relationships in the Relationships Graph with a graphical technique consisting of dragging from one match field to another. In addition, you can add a new relationship simply by clicking the small Add Relationship icon (the second icon in the Tables/Relationships icon group at the lower left of the Relationships Graph, as shown in Figure 7.3). Clicking that icon brings you the familiar Edit Relationship dialog, but it's initially completely empty.

Begin by selecting the two tables that are to participate in the relationship. Draw the relationship between Projects and TasksBeforeDate. It doesn't matter which direction you draw the relationship, but you need to notice which table is on which side in the Edit Relationship dialog. Define the first match criterion. Select the ProjectID field from TasksBeforeDate and the zzID field from Projects, make sure that the menu of operators in the middle shows an equal sign, and click the Add button. So far it looks exactly like the Edit Relationship screen for the original Projects-Tasks relationship that was shown in Figure 7.1, except that the table occurrence on the left is now TasksBeforeDate instead of Tasks.

But you still need to tell FileMaker to consider only those Tasks on which the due date is before the cut-off date in the gDateLimit field. To make this happen, you add another criterion to the relationship. Select gDateLimit on the right in Projects, DueDate on the left in TasksBeforeDate, and from the operator menu in the middle, select the < sign. Click Add, and the new match criterion is added in the middlemost box, as shown in Figure 7.4.

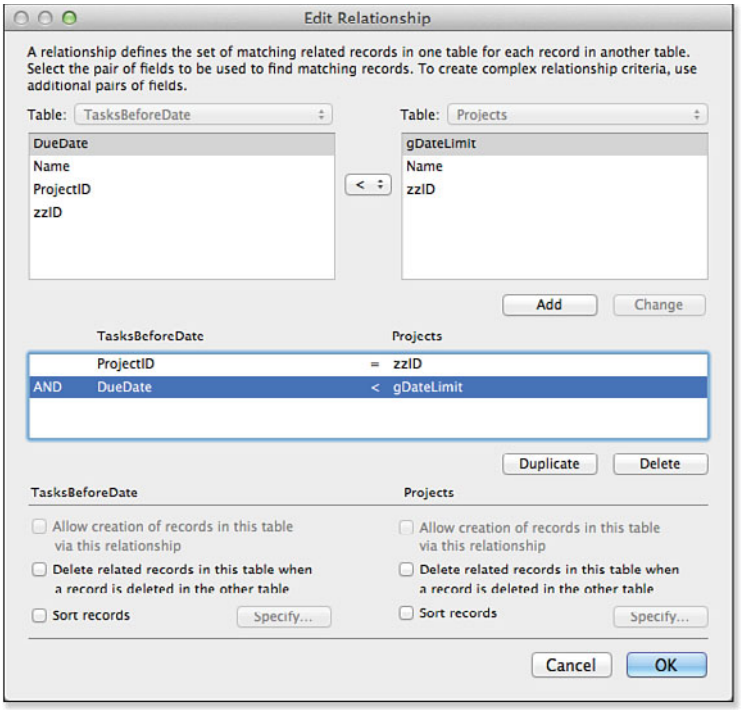


Figure 7.4
Using a nonequality condition to build a relationship.

Notice what that middle box is saying now. There's a large AND in the left margin, which says that this relationship pulls only those tasks for which the project ID matches AND the due date is before the cut-off date.

note

Relationships in the Relationships Graph are bidirectional; you can access one table occurrence from the other side. This does not matter with equijoins. However, with nonequijoins, you must make certain that the logic recognizes which table is on which side. In this case, because the cut-off date is in Projects on the right, the operator to select due dates for Tasks (on the left) must be <. If TasksBeforeDate were on the right, the operator to select due dates would be ≥. Both relationships would enforce the same logic.

caution

You might be wondering how to create a multiple-match relationship that works if any one of the criteria is true, as opposed to those that work only if all the criteria are true. This isn't possible, unfortunately. To learn more, see "No OR Conditions with Multiple Match Criteria" in the "Troubleshooting" section near the end of this chapter.

Notice also how FileMaker represents this new relationship in the Relationships Graph. Each end of the relationship line forks to indicate the multiple match criteria—and the operator symbol in the middle of the line is a curious kind of X, indicating a complex match with multiple operators at work. Figure 7.5 shows the Relationships Graph with the new relationship.

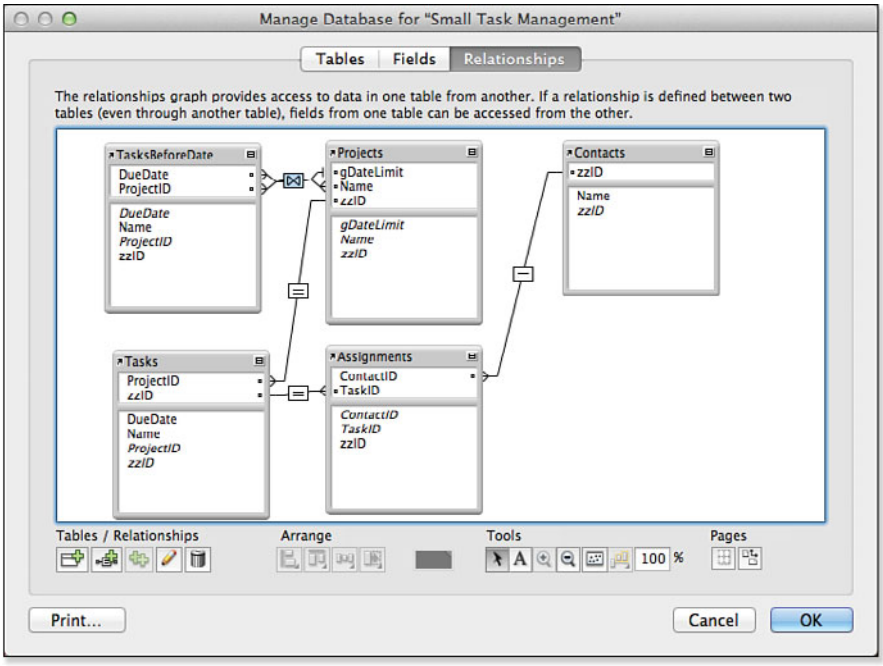


Figure 7.5
The Relationships Graph indicates when a relationship is based on multiple match fields. The [X] comparison operator shows that multiple operators are in use as well.

To use the new relationship, you could draw another portal on the Projects layout. Base it on TasksBeforeDate instead of plain Tasks, and use the same data fields. The result should be similar to what you see in Figure 7.6. When you simply change the date, different data is automatically displayed. (You use the Projects layout on the base because that is the place where the gDateLimit field is located; the portal is to TasksBeforeDate.)

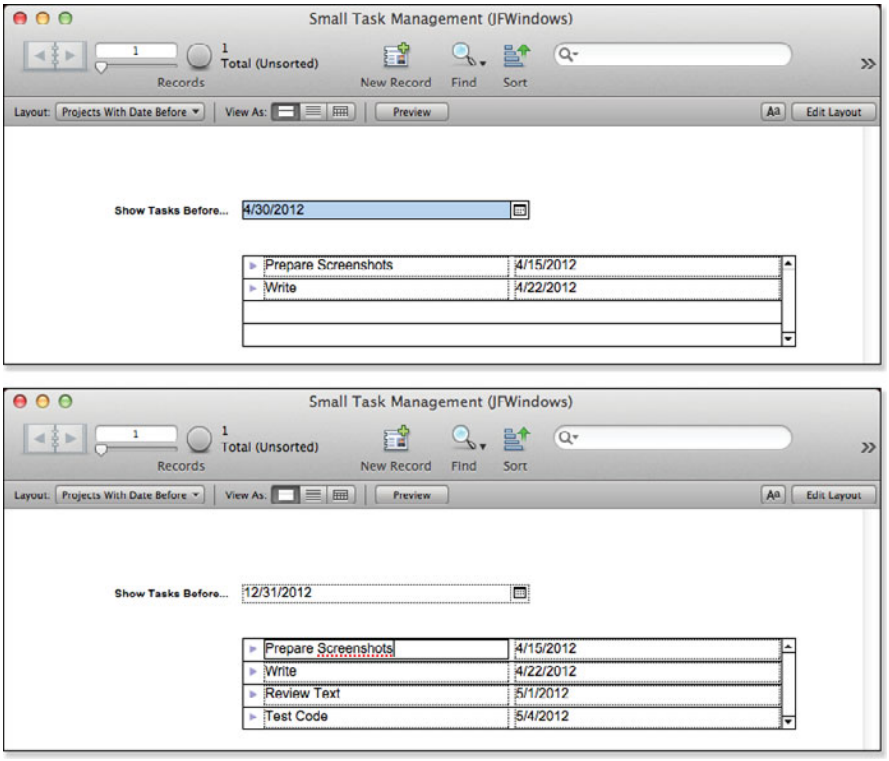


Figure 7.6
More complex relationships can produce sophisticated views, such as the TasksBeforeDate view shown here, which uses a portal onto the complex relationship as well as the FileMaker Pro drop-down calendar interface elements.

tip

You can also duplicate the existing Projects layout, rename it, and double-click the portal to change its source from Tasks to TasksBeforeDate. Remember to double-click the fields in the portal to also change their source from Tasks to TasksBeforeDate. That is the method used in the downloadable example.

To add a comparable TasksAfterDate relationship, you can repeat the steps in this section; just reverse the direction of the relationship operator (in other words, the cut-off date will be before the task due dates).

note

In designing an interface like this, you have to decide how to handle exact matches. If you use less than or equal or greater than or equal, you will display exact matches to the cut-off date in both portals, which could be misleading. You can add a third portal that displays the exact date's events, or, provided that you clarify this in the interface, you can choose a combination of less than or equal and greater than (or the reverse) so that exact match dates appear in only one place or the other.

These three concepts—nonequijoins, multiple table occurrences, and multiple match criteria—afford you extraordinary flexibility as a database developer. The sections ahead explore examples that show how to use these tools to solve particular problems of database design.

Creating Self-Relationships

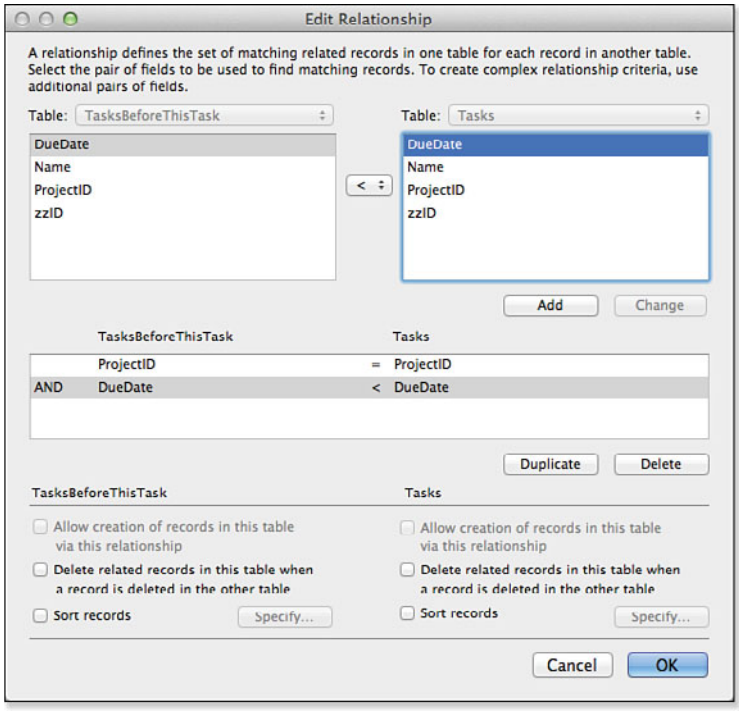
In the preceding section, you created a relationship controlled by a date in the Projects table to display related data from Tasks. You can use the same logic to relate one occurrence of the Tasks table to another occurrence of the Tasks table so that you can see tasks due before a given task.

The basic process is the same: You select a field in the controlling table (which is a Tasks occurrence, such as the original Tasks table), and you implement the relationship to a new table occurrence, which drives a portal that is displayed in the window of the basic Tasks occurrence. You do not even have to add another date field. You can use this mechanism to display the tasks due before the date of a given task in the same project. This section shows you how to do that.

First, create a new table occurrence based on the Tasks table called TasksBeforeThisTask using the same technique illustrated previously in Figure 7.2.

Then, instead of creating a relationship to Projects, you can create a relationship between the new occurrence, TasksBeforeThisTask, and the original Tasks occurrence. Figure 7.7 shows that relationship.

Figure 7.7
Create a self-join relationship.



Both occurrences are based on the Tasks table, so the fields are the same. Match the DueDate field of Tasks to DueDate in the new occurrence so that DueDate in Tasks is greater than DueDate in the new occurrence (or vice versa—it doesn't matter as long as you are consistent in your naming). You also need to match the ProjectID in both Tasks records so that you are including only peer tasks with the same Project ID.

This is the same logic you applied in the preceding section, and you can now update the Tasks layout so that it displays data for a Tasks record at the top and displays a TasksBeforeThisTask portal below, as shown in Figure 7.8. As you page through tasks, you will see that the portal is always updated to show peer tasks on the project with due dates before the current task's due date.

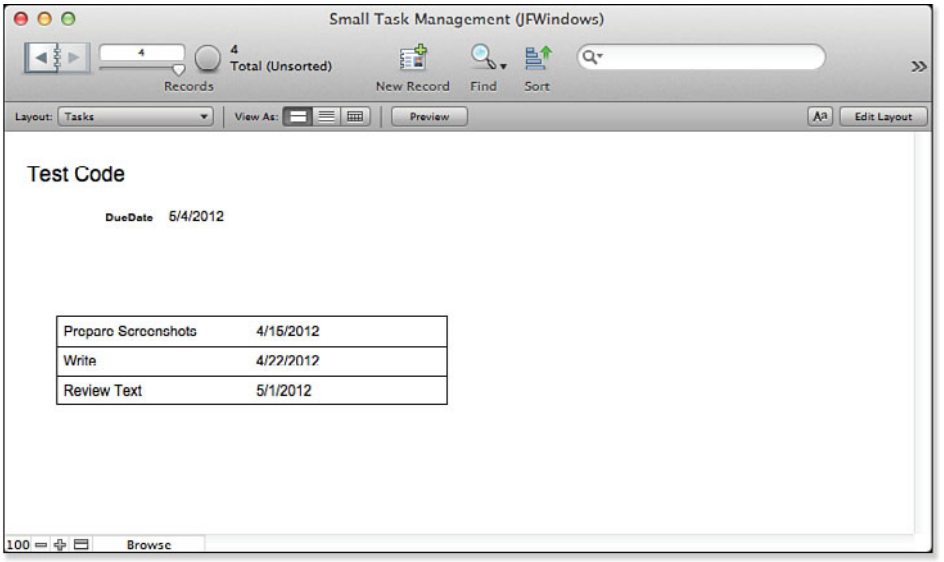


Figure 7.8 Display the self-join.

Creating a Relationship with a Global Value

You have seen how to implement the Project-Tasks relationship where a global date field controls the selected Tasks records. (Remember that you have to take into account the issues of multiple windows in deciding whether to use a global field.)

You can also use a constant global value, one that is not entered dynamically by the user, to implement a relationship.

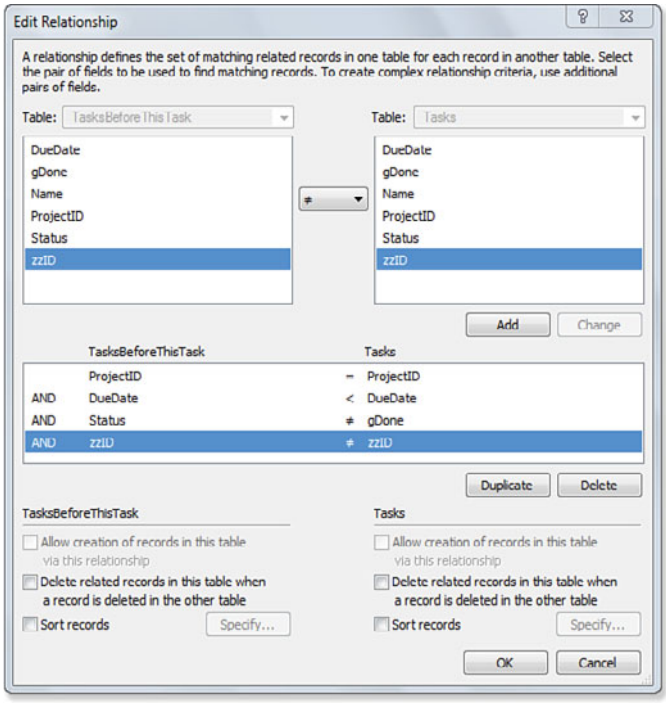
An obvious candidate for such a relationship is a modification to the preceding self-join. You can add a third component to the join so that you select peer projects (that is, those with the same Project ID), due dates before the given task's due date, and a new field—status—equal to a global status value. To implement this, you add two fields to the Tasks table: Status and gStatus. One is the status of the task, and the other is a global value that contains a status value from the status value list.

You can build on the relationship between Tasks and TasksBeforeThisTask to incorporate the new status field. The clause that you want to add is a match in which Status in TasksBeforeThisTask matches gStatus.

For more details on globals in a multiuser environment, **see** *Chapter 19, "Debugging and Troubleshooting."*

Figure 7.9 shows the completed relationship. One additional feature has been added: The zzID field on both sides must not match. This is one way of resolving the issue noted previously in which a greater than or equal (or less than or equal) operator can include the base task itself in the list of related tasks. This prevents the task from being related to itself.

Figure 7.9
Enhance the relationship.



As always, when you add fields to the Tasks table, you will notice that the Relationships Graph is automatically updated so that all table occurrences display the new fields.

If you create a Status value list, you can add a set of radio buttons to the Task layout (see Figure 7.10); if that layout or another one displays related tasks (such as TasksBeforeThisTask described in the preceding section), you can add the status field to the portal.

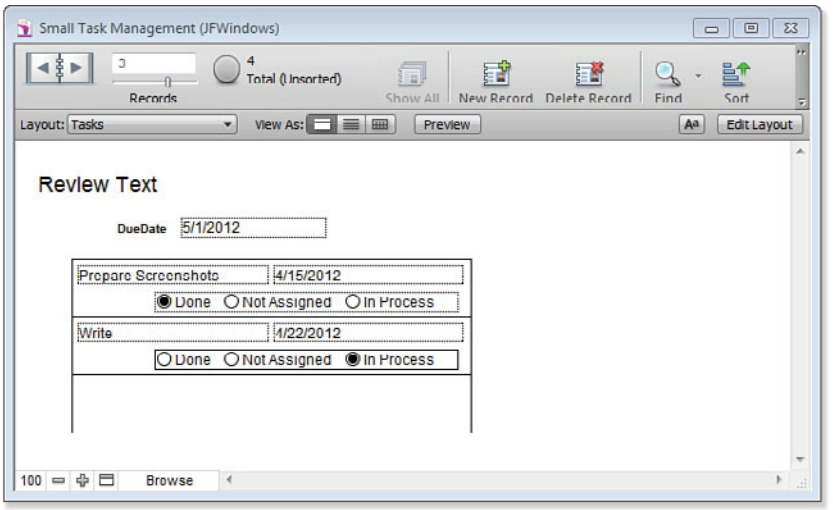


Figure 7.10
Finish up the layout.

tip

This relationship, whether implemented between two tables or as a self-join, demonstrates the basics of a vast number of queries that you can implement in FileMaker Pro as relationships. One of the virtues of implementing queries as relationships is that they are always there: The logic is supplied in the relationship, and you do not have to worry about implementing find requests. Everything that you can implement in the database—be it in the Relationships Graph, in validation rules, or with auto-entry of data—represents something that does not have to be implemented in the scripts and interface or with user commands. That means after it is done, it is there and correct for all uses.

Self-joins come into play whenever you want to relate like objects. For example, you can use them to implement hierarchies such as employment structures in which each employee has a manager, who in turn has a manager.

Creating Cross-Product Relationships

In working with nonequijoin relationship matches, you might have noticed one oddball operator in the little menu of match criteria. Most of them are familiar comparison operators—but what about the last one, the one that looks like a ×?

That operator is known as a *cross product* (or Cartesian product). The cross product provides a "universal match" between the records in two tables. This means that it does no limiting of any kind. If you think of a relationship again as a kind of query, a cross-product relationship is a "find all" query. If you define a cross-product relationship from Projects to Tasks, a portal based on that relationship would always show all Tasks, no matter which Project record was being viewed. The choice of fields on the left and right sides is more or less unimportant; this "all to all" relationship is fulfilled regardless of the choice of match fields.

Cross products really make sense only by themselves, in single-match relationships. They have no effect at all if they're added into multimatch criteria sets. A cross-product match condition is always true, so it can never further limit the potential matches of other criteria.

The cross product is the ultimate nonstructural relationship. After all, its purpose is to show all of something. These are generally used for various user-interface purposes. Sometimes you might want users to pick from a list of things, for example, and it's more pleasing to allow them to pick from a scrolling list in a portal than from a drop-down list or menu. Generally, such techniques need to be coupled with some scripting to react to users' choices.

Working with Multiple Files

In all the discussions of multitable systems in Chapters 5 and 6 and so far in this chapter, we've assumed that all the tables you want to work with live within a single FileMaker file. The capability to have many tables in a single physical file is, after all, one of the more convenient features of FileMaker. But there are still many reasons to build systems that are multifile, in addition to being multitable. This and the following sections review the mechanics of working with several files at once and then discuss different design strategies that use a multifile structure.

The FileMaker History of Multiple Files

The simplicity of placing multiple tables in a single database file and relating them to one another using the Relationships Graph was introduced in FileMaker 7. Prior to that, there was really no distinction between tables and files: A database file was treated as a single table, and relationships were made between files (not tables, because the separate concept of a table did not exist).

Relationships were built from one file to another, and, in part because there was no overall Relationships Graph, relationships could not extend beyond a single other file (this was often referred to as a *hop*). The relationship between Tasks and Contacts shown previously in Figure 7.2 exists because of the relationship between Tasks and Assignments, and then the relationship between Assignments and Contacts, which together create a path from Tasks to Contacts.

To be able to access data more than one hop away, you needed to use a simple workaround before FileMaker 7. The relationships that were shown in Figure 7.2 would have been implemented as follows: In Assignments, a calculation field would be created that used a relationship to Contacts to store some Contacts data into the calculation field in Assignments. Then Tasks could access this calculated field one hop away in Assignments, which itself contained related data from Contacts.

It is important to know this history because if you are working with a database that was initially created before FileMaker 7, you might still see these intermediate calculated values. The automatic conversion of pre–FileMaker 7 databases to FileMaker 7 does not change this architecture, but as you modify these older databases, you might want to implement the more modern multihop relationships and remove the calculated fields.

So far we've looked just at relationships between tables within the same file. But it's also possible to build relationships between tables in different files. A very common situation arises when you are building a solution that uses data from an existing FileMaker database. In the case of the example used in this chapter, perhaps the Contacts table already exists in its own database file.

To reference the Contacts table in another file from your file, you only need create an external data source that uses the other file. You can then use that external data source to create new table occurrences for those tables. The data remains in the external file, but you can access it through the file references.

The use of external data sources lets you use other FileMaker database files. You can also reference external databases using SQL and ODBC, as described in Chapter 21, "Connecting to External SQL Data Sources."

Creating an External Data Source

External data sources are an extremely important topic in FileMaker. In a number of places in FileMaker, you might want to refer to or work with another file. Here are some of the things you can do with other files in FileMaker:

- Call a script in the other file.
- Use a value list defined in the other file.
- Refer to one or more tables from the other file in your Relationships Graph.

To do any of these things, you must first create a reference to the other file using an external data source. A file reference simply tells FileMaker where and how to find another file. FileMaker is capable of working with external files present on a local hard drive, present on a shared network volume, or present on an available FileMaker server. You can also specify multiple search locations for a file, and the priority in which they should be searched. You can, for instance, create a file reference that says, "First search for the file on the FileMaker server at 192.168.100.2. If you don't find it there, look on the FileMaker server at 10.11.1.5. If you don't find it there, give up or go on to these locations…"

Before FileMaker 7, versions kept track of these references behind the scenes and didn't let you alter the order in which FileMaker searched for a given file. Problems with file references were harder to spot in previous versions and could occasionally give rise to a problem called *crosstalk*, in which the wrong copies of files could be accessed by mistake.

Since FileMaker 7, each physical file maintains its own list of file references. You can work with these references centrally and create them on the fly as needed. Let's see how this works in practice.

Your first step is to define an external data source that will contain a file reference to the external file. To do this, choose File, Manage, External Data Sources. Click the New button on the next screen, and you see the Manage External Data Sources dialog, as shown in Figure 7.11.

When you click New or select an existing data source and click Edit, the dialog shown in Figure 7.12 opens. You can name the data source and select whether it is an external FileMaker file or one accessed via ODBC.

Figure 7.11
Add external data sources to a FileMaker database file.

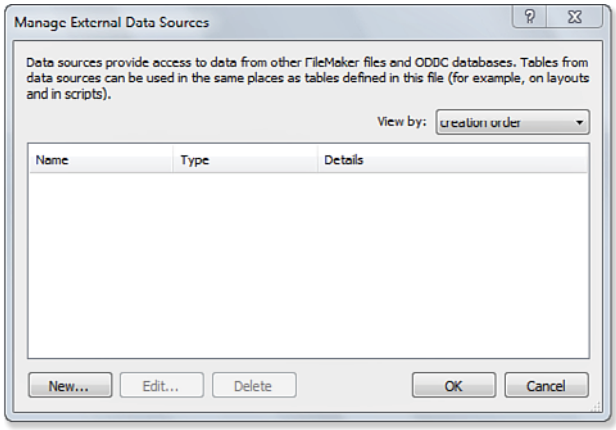
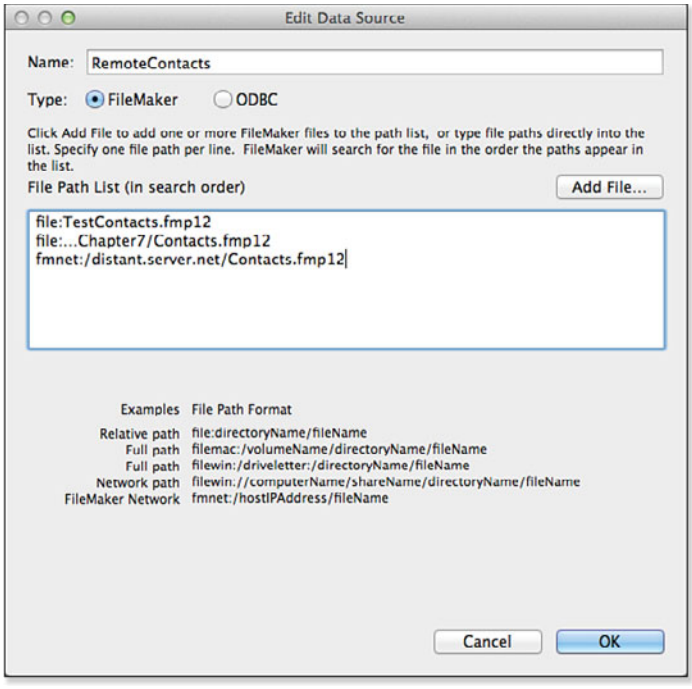


Figure 7.12
Edit the external data sources.



Additional information on working with external FileMaker files is included in Chapter 21, "Connecting to External SQL Data Sources."

Then you provide the actual file references for the data source. You can type in the path to the file that you want to use, or you can click Add File to use the standard Open File dialog to select the file from your local hard disk or the network. As you can see in Figure 7.12, you can provide several file references. FileMaker will search them in order until it finds a file to use.

In Figure 7.12, you can see a reference to a file in the same folder as the database into which the file reference is created. Below that is a file reference to a file in a folder with a common parent folder as that of the database file. On the third line is a reference to a file accessed through FileMaker Server or FileMaker peer-to-peer networking. In that case, as is always true when you use the Open Remote command, you connect to a copy of FileMaker and through it to the database. The other syntax connects to the database using your own copy of FileMaker.

As you can see at the bottom of the dialog, there are different forms of the syntax for OS X and Windows. Because the file references are searched in order, you can vary them for your development process.

If none of the file references can be resolved, you get an error message.

note

In general, all the different file paths in the path list point to the same file; that is, a file with the same name and contents. In theory, you could also use a single path list to point to a number of different files, indicating that the later ones should be used if the earlier ones can't be found. You could perhaps use this feature to fall back to other versions of a file or system if necessary.

caution

If you inserted a local reference to the file to aid offline development, you need to remember to remove that reference later, or perhaps move it lower on the list. Otherwise, FileMaker continues to search your local drive first, which is probably not desirable.

note

File references are resolved as FileMaker needs to access the given file rather than when the database opens. That improves the efficiency of cases in which not all file references are used in a particular session of FileMaker. It also is the reason why resolving file references occasionally can provide some frustration. If you have a situation in which sometimes there is an error in your FileMaker solution but not always, it might be that a file reference is to blame and the apparent randomness of the error is caused by the sequence in which file references need to be resolved. Sometimes you can track down these intermittently appearing bugs by reviewing all the file references in your external data sources and verifying that they are correct. Trailing spaces at the end of the file references are a notorious source of error.

Adding an External Table to the Relationships Graph

After you create the file references in your external data sources list, you can add tables from these external files to your Relationships Graph. If you open the Relationships Graph and click the Add Table Occurrence icon, you'll notice something we didn't highlight before. In the resulting Specify Table dialog, a menu lets you choose which file you want to browse for table choices. This menu always includes the current file and any file references you defined using the techniques covered in the previous sections of this chapter. Figure 7.13 illustrates this point.

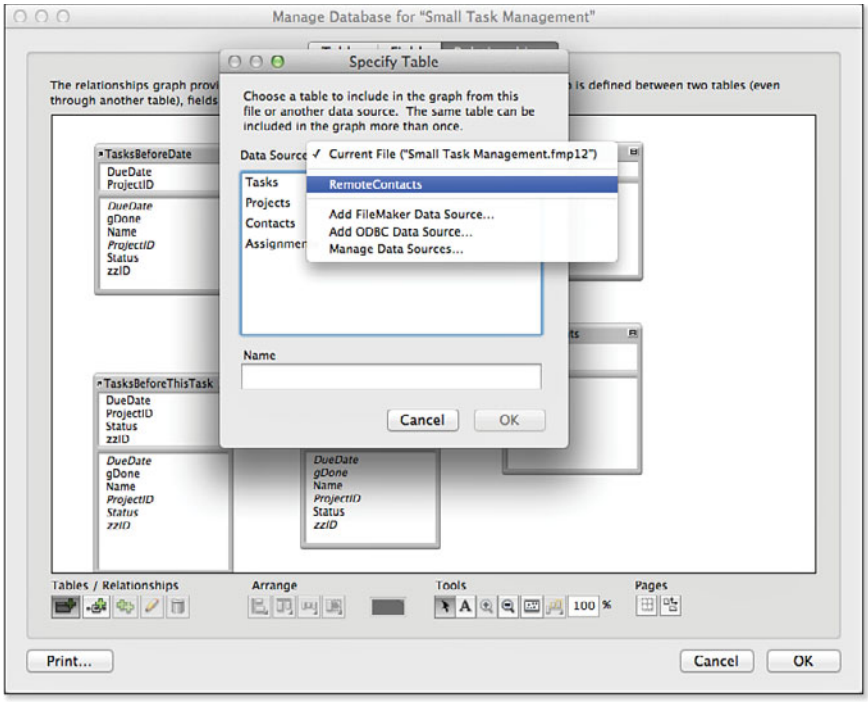


Figure 7.13
When adding a table occurrence to the Relationships Graph, you can base the new occurrence on a table in an external data source.

Because you name the external data source, you will see that name in the pop-up menu, as shown in Figure 7.13. Not only might you be accessing files in various locations, they may have different names, but the external data source name that appears in this dialog is what you work with.

The table is added to the Relationships Graph, much as all the other tables we've seen. The result is shown in Figure 7.14. There's one subtle visual indication that the Contacts 2 table occurrence is based on a table from another file: The table occurrence name for Contacts 2 is italicized. In addition, if you hover the mouse pointer over the arrow in the upper left of the table, you can see the data source name. Otherwise, it's just as though you were working with a table in the same file.

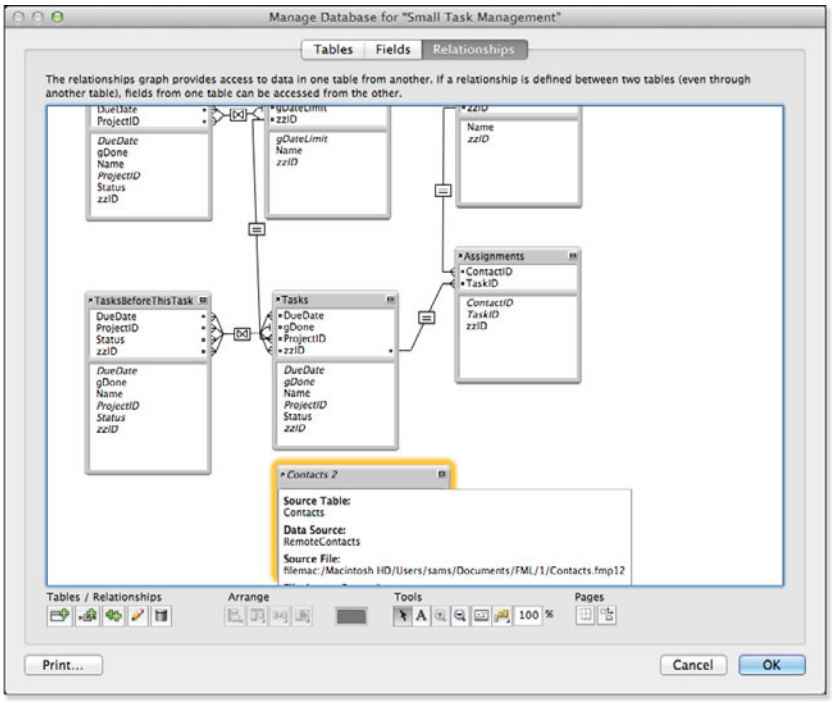


Figure 7.14
In this Relationships Graph, the italicized title of the Contacts 2 table occurrence shows that the source table exists in an external file.

Troubleshooting

Trouble Creating Related Records with Nonequijoins

I want to create a relationship that allows creation of related records on one side of the relationship, but the box that enables that capability is grayed out.

You might have noticed that the option Allow Creation of Records in This Table via This Relationship has mysteriously been disabled when a nonequijoin is part of the relationship. This suggests that you have one or more nonequality conditions in your relationship match criteria. The rule is this: FileMaker can allow the creation of related records only if the relationship in question consists only of conditions involving an equality comparison. This limits such relationships to using only the equal (=), less than or equal (≤), or greater than or equal (≥) operators.

Multiple match criteria are fine, as long as they're all based on one of those three operators. This can actually be rather useful: A multimatch relationship that allows the creation of related records automatically fills in all the key fields of the related record. But as soon as any nonequality condition becomes involved in the match, the capability to create related records goes away.

This makes sense if you think about it. FileMaker can create a record via an equijoin because only one condition satisfies the match criteria for the current record. Suppose that you're on a Customer

Layout, looking at customer number 17, and you have a portal into Invoices, in which the relationship to Invoices is an equijoin on CustomerID. FileMaker can create a new record in the portal by creating a new invoice record and setting the CustomerID to 17. But suppose that the relationship instead were based on a "not equal to" relationship? To create a record on the other side, FileMaker would need to create an Invoice record with a customer ID other than 17. Fine, but what customer ID should it use? There's really no way to say. Similar reasoning holds for other nonequijoin types: There's no sensible way for FileMaker to decide what match data should go into the related record.

If the capability to create related records is enabled, the key fields in the related record will always be populated with values equal to the key field in the parent record, regardless of which of the three allowable relational operators is chosen. If you need to create related records where the relationship is not equality, you almost always do it with a script and, in many cases, that script is launched by a button that you place next to a portal.

No OR Conditions with Multiple Match Criteria

Whenever I add multiple match criteria to a relationship, FileMaker always tells me the match will work if condition 1 AND condition 2 AND condition 3 are true. But I have a match that needs to work if 1 OR 2 OR 3 is true. Where do I set that up?

You don't, unfortunately. Using the native FileMaker relationship features, relational matches are always AND matches whenever multiple match criteria are specified. If you want to mimic the effect of an OR search in another table, you need to find another means of doing that. Say, for example, that you have a database with tables for teachers, classes, enrollments, and students. From the viewpoint of a teacher, you want to be able to view all students who are outside the norm—they have either a very low GPA or a very high GPA. You could try to do this with two match criteria, but that would necessarily be an AND match, which would never be fulfilled (no student would have both a low and a high GPA at once). The solution here would be to create a stored calculation in the student table called something like ExceptionalGPA, defined as follows:

```
If ( GPA < 2 or GPA > 3.75; 1; 0)
```

The calculation will have the value 1 when the student's GPA is exceptionally high or low, and a value of 0 otherwise.

You could now create a field called Constant in the teacher table and define it as a calculation that evaluates to 1. Then specify a relationship between the teacher table and the student table, with multiple match criteria: TeacherID=TeacherID and Constant=ExceptionalGPA, meaning "Find me all students with the same teacher ID and an exceptional GPA."

FileMaker Extra: Managing the Relationships Graph

The Relationships Graph in FileMaker is a nice answer to developers who clamored for years for a visual representation of relationships in FileMaker systems. But for large or complex systems, with many table occurrences, the Graph has the potential to be a bit unwieldy. Table occurrences in the Graph take up a fair amount of space, and it can be difficult to organize the occurrences without creating a web of overlapping relationship lines.

Using Formatting Tools to Manage the Relationships Graph

You can use a number of tools for Graph management. For one thing, the small "windowshade" icon at the upper right of a table occurrence can be used to hide the fields in the table occurrence, leaving only the match fields used in relationships. This can save valuable space. If you like to work from the keyboard, (⌘) [Ctrl+T] will cycle through the various table occurrence display states (fully open, key fields only, fully closed). If you use (⌘) [Ctrl+A] to select all objects in the Graph, you can windowshade your entire Graph with a few keystrokes.

You can also manually resize an individual table occurrence to save space. This, again, needs to be done one table occurrence at a time. It's also possible to zoom out from the Graph as a whole and view it at 75% or 50% of regular size, or smaller.

It might also be useful to you to organize your table occurrences into logical groups of some kind within the Relationships Graph. Let's say you're working on a trucking module with four table occurrences, and you also have a file reference to an external user-management module and you've used that to bring a number of user-oriented table occurrences into the Graph. FileMaker enables you to color-code table occurrences in the Graph, so it's possible to give each group of table occurrences its own color.

You can add notes directly to the Graph. If you drag a rectangle in the Graph while holding (⌘-N) [Ctrl+N], you'll create something like a sticky note. You can choose the color and typeface, and adjust the size and position. Notes appear behind other objects in the Graph.

While you are working, several keyboard shortcuts can make your life easier. Pressing (⌘-Y) [Ctrl+Y] will select all related table occurrences that are one step away from the current table occurrence. Pressing (⌘-U) [Ctrl+U] will select all table occurrences with the same source table as the current table. Finally, you can use (⌘-D) [Ctrl+D] to duplicate one or more selected table objects, as well as any relationships between them. This last point is a big convenience: You can select a complex group of related table occurrences and duplicate the entire cluster and its relationships at once. You can perform all these functions with the mouse by clicking new buttons that appear in the Relationships Graph.

Using Table Occurrences to Manage the Relationships Graph

As you have seen in this chapter, there must be unique paths between any two related table occurrences. A Customers table may be related to an Invoices table, which is in turn related to an Invoice Line Items table, which ultimately is related to Products. Thus, a path from Customers to Products exists.

If you want to track inquiries about products from a customer, you might have a path from Customers to Queries to Products. But that can't be done, because now there are two paths between Customers and Products. The solution is to create a new table occurrence and name it something like ProductQueriesForCustomer. Even relatively small FileMaker solutions quickly wind up with special-purpose table occurrences of this sort. There is nothing wrong with this: It is just part of the development process.

Rather than creating duplicate table occurrences on an as-needed basis, many developers have begun to create them as part of the design process. Instead of a spider web of related and duplicative table occurrences, you can build smaller relationship sets that are tied into specific layouts and

their purposes. Remember that a layout is always based on a primary table occurrence; from that table occurrence, related table occurrences can be shown in portals or individual fields. The set of table occurrences together with the layouts that use them form a logical group.

In the example described previously, you could create a Customer Query layout based on Queries. Right from the start, you can create new table occurrences such as Customers For Queries and Products For Queries. Relate both of those to Queries and use them in your Customer Query layout.

Now create a Customer Order Layout based on Orders. Again, create new table occurrences such as Customers for Orders, Order Line Items for Orders, and Products for Orders (or, more accurately, Products for Order Line Items).

This approach gives you more table occurrences in most cases, but the sections of your Relationships Graph are separate and easily understood. Each cluster will use its own table occurrences to support its own layouts. When you implement scripts or buttons to go to related records, you can go to a related record from the appropriate table occurrence.

Remember that you always have the option to display that related record in its own layout. Your layout choices consist of the layouts based on the underlying table (not the table occurrence). It is that particular feature that lets you "jump" from cluster to cluster.

For more details on using table occurrences, **see "**`Go to Related Record`**," p. 478.**

This page intentionally left blank

8

GETTING STARTED WITH CALCULATIONS

Understanding How and Where Calculations Are Used

Calculations are among the most important and powerful tools at your disposal in the development of FileMaker Pro solutions. Some people find learning calculations to be an easy task, whereas others can find writing complex calculations to be daunting. Whichever camp you fall into, calculations will enable you to unlock much of the advanced power within FileMaker. This chapter and its companion, Chapter 15, "Advanced Calculation Techniques," will provide you with a solid grounding.

This chapter focuses on basic calculation functions and techniques for using them well. Chapter 15 looks at more advanced calculation formulas and specific techniques. If you're new to FileMaker, you should start here.

From the outset, it's important to understand the difference between calculation fields and calculation formulas. The term *calculation* is often used to denote both concepts. A *calculation field* is a particular type of field whose value is determined through the evaluation of a calculation formula. *Calculation formula* is a broader concept that refers to any use of a formula to determine an output, and that output can be a value that is stored in a calculation field or it can be a value that is used in evaluating an `if` statement, constructing a tooltip, determining whether access to a field is allowed, or dynamically specifying a layout to go to. When you learn "calculations," you're really learning calculation formulas. It so happens that you'll use calculation formulas to construct calculation fields, but the formulas are applied widely throughout FileMaker solutions.

note

Many of the examples in this chapter are based on the Time Billing Starter Solution.

Exploring Calculation Expressions

Calculations are grouped into categories, such as those for manipulating text, those for managing statistical calculations, a wide variety of functions that interact directly with FileMaker, and a growing group of introspective functions.

You are able to use functions to query the environment in which a calculation is running. Querying the environment is referred to in the programming world as *introspection*. Certain functions in FileMaker can let you know whether you are running in FileMaker for Windows or FileMaker for Mac. With the advent of FileMaker Go, mobile devices, and web publishing, you can find introspective functions that help to identify the device and operating system on which your calculation is being evaluated.

This is not just a matter of idle curiosity. Introspective functions let you automatically switch from a desktop-based layout to one designed for a mobile device or for the Web. All of this depends on the presence of introspective functions (often the `Get` function, described later in this chapter) and your knowledge of the use of those functions.

Writing Calculation Formulas

Essentially, the purpose of a calculation formula is to evaluate an expression and return a value. In Figure 8.1, for example, you can see the field definition for a calculation field called `Amount` (this is from the Time Billing Starter Solution). The value of this field is defined to be the result of the rate multiplied by the time as specified in the `Time in Unit Total` field. The `Time in Unit Total` field itself is also a calculation field—one that converts seconds into hours, which is the unit on which `Rate` is based. Thus, you have a chain of calculations that are all evaluated as necessary so that the `Amount` field is calculated and ready for you to use as you wish.

note

You can use calculations in any of the FileMaker products; however, you can create them only in FileMaker Pro and FileMaker Pro Advanced.

Most of the expressions you use in calculation formulas are intended to return a value, and that value might be a number, a text string, a date or time, or even a reference to a file to place in a container field. You select the calculation result type in the lower left of the Specify Calculation dialog.

Another class of formulas, however, is intended to evaluate the truth of an equation or statement. The value returned by these formulas is either a `1`, indicating that the equation or statement is true, or `0`, indicating that the equation or statement is false. Typically, calculations are used in this manner in `If` script steps, in calculated validations, and for defining field access restrictions.

The calculation itself is created in the space in the lower center of the Specify Calculation dialog, as you see later in this chapter. Just above that space, you can see the distinction between a calculation that is used to define a calculation field and one that is designed to return a Boolean value. In the first case, as you see in Figure 8.1, the name of the calculation field (Amount, in this case) is shown. In the second case, that area is blank.

*To learn more about field validation, **see** Chapter 3, "Defining and Working with Fields and Tables."*

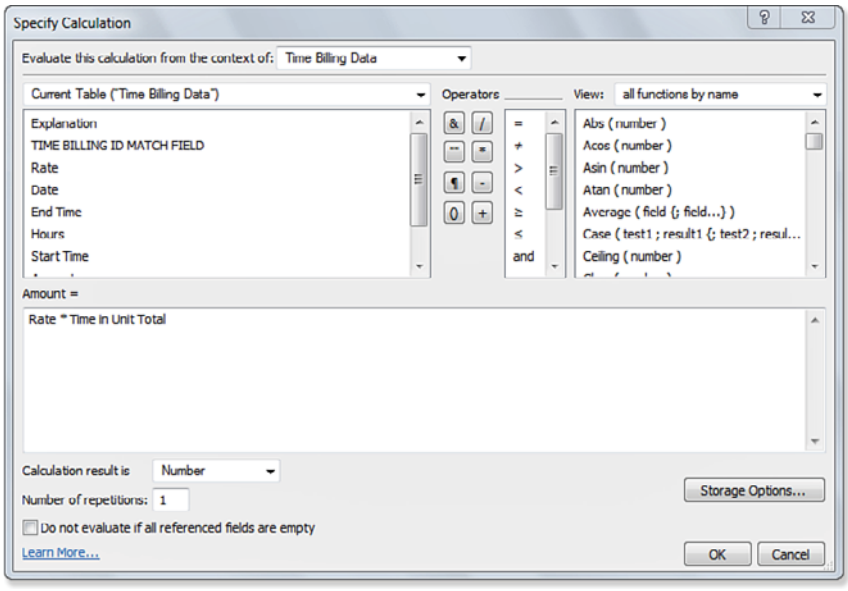


Figure 8.1
When defining calculation fields, you specify an expression to evaluate in the Specify Calculation dialog.

In Figure 8.2, you can see a calculation dialog that specifies the condition for an `If` script step. This very common calculation checks to see whether the `SystemPlatform` value is greater than `2` (in other words, it checks to see whether you are running on an iPad). The result of this calculation is used to switch between desktop and iPad layouts. (This is one of those introspective functions.)

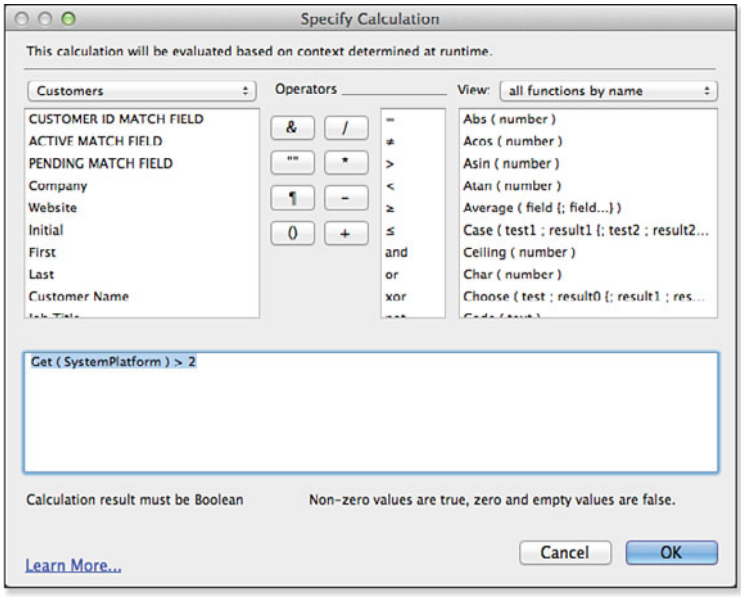


Figure 8.2
Calculation formulas are often used to determine the truthfulness of an equation or a statement.

In situations where FileMaker is expecting a formula that returns a true/false result, you see the words "Calculation result must be Boolean" near the bottom of the Specify Calculation dialog. The `If` script step shown earlier is a typical example of this situation. *Boolean* is a software programming term for a value with one of two states: true or false. Any value returned other than `0` or a null value (for example, an empty string) is considered true.

note

Boolean rules apply for text values, dates, negative numbers, and so on. `"Hello"` is true (not zero and not null), a single space character (`" "`) is true, and `-1` is true. Note also that the results of a formula evaluate in the same way: `(0 * 100)` is false. `(0 + 100)` is true. Last, also note that you can use comparative operators: `1 and 1` is true (where each clause on both sides of the `and` operator evaluates to true), `1 or 1` is true, `1 xor 1` is false, and so on. You learn about operators later in the chapter.

caution

The `GetAsBoolean()` function treats all data as numeric such that, for example, `"hello"` evaluates as false and `"hello999"` evaluates as true. This is inconsistent with the way in which other Boolean logic operates, so be sure to take note of it.

Uses for Calculation Formulas

This chapter focuses on the use of calculation formulas in field definitions, but it's important that you understand that calculation formulas are used in other places as well. Briefly, they include the following:

- **Script steps**—Calculation formulas come into play in many script steps. The `If`, `Set Field`, and `Set Variable` script steps are notable examples. Many other script steps allow you to use a calculation formula to act as a parameter. A sampling includes `Go to Layout`, `Go to Field`, `Go to Record`, `Pause/Resume Script`, and `Omit Multiple`. Additionally, calculation formulas can be used to define script parameters and script results.

- **Field validation**—One of the options available to you for validating data entry is validating by calculation. This, in effect, lets you define your own rules for validation. For example, you might want to test that a due date falls on a weekday, or perhaps that a status field not allow a value of "complete" if data is missing elsewhere in a record.

  The equation you provide is evaluated every time a user modifies the field. If it evaluates as true, the user's entry is committed. If it doesn't, the user receives an error message. For instance, if a user is supposed to enter a callback date on a contact record, you might want to validate that the entry is a future date. To do this, you might use the formula `Call_Back_Date > Get ( CurrentDate )` as the validation for the `Call_Back_Date` field.

- **Record-level security**—When you define privilege sets, you have the option of limiting a user's access to view, edit, and delete records based on a calculation formula you provide. If the equation you provide evaluates as true, the user can perform the action; if not, the action is prohibited. For instance, you might want to prevent users from inadvertently modifying an invoice that has already been posted. So, you set up limited access for editing records based on the formula `Invoice_Status ≠ "Posted"`. Only records for which the formula is a true statement would be editable.

- **Auto-entry options**—When you're defining text, number, date, time, and timestamp fields, several auto-entry options are available for specifying default field values. One of these options is to auto-enter the result of a calculation formula. For instance, in a contact management database, you might want a default callback date set for all new contact records. The formula you would use for this might be something like `Get ( CurrentDate ) + 14`, if you wanted a callback date two weeks in the future.
- **Calculated replace**—A calculated replace is a way of changing the contents of a field in all the records in the current found set. It's particularly useful for cleaning up messy data. Say, for example, that your users sometimes enter spaces at the end of a name field as they enter data. You could clean up this data by performing a calculated replace with the formula `Trim ( FirstName )`.

Exploring the Specify Calculation Dialog

Now that you know something about how and where calculation formulas are used, it's time to turn to the layout of the calculation dialog itself. The various calculation dialogs you find in particular areas in FileMaker Pro have some small differences. We focus our attention on the dialog used for defining calculation fields because it's the most complex.

Writing the Formula

The large box in the middle of the Specify Calculation dialog is the place where you define the formula itself. If you know the syntax of the functions you need and the names of the fields, you can simply type in the formula by hand. In most cases, though, you'll want to use the lists of fields and functions above the text box. Double-clicking an item in those lists inserts that item into your formula at the current insertion point.

Every calculation formula is made up of some combination of fields, constants, operators, and functions. All the following are examples of formulas you might write:

```
2 + 2
FirstName & " " & LastName
Get ( CurrentDate ) + 14
Left ( FirstName; 1 ) & Left ( LastName; 1 )
"Dear " & FirstName & ":"
$loopCounter = $loopCounter + 1
LastName = "Jones"
```

FileMaker assumes that any unquoted text in a formula is a number, a function name, or a field name. If it's none of these, you get an error message when you attempt to exit the dialog.

In these examples, `FirstName` and `LastName` are fields. `$loopCounter` is a variable by virtue of being prefixed with a dollar-sign character. `Get ( CurrentDate )` and `Left` are functions. The only operators used here are the addition operator (+) and the concatenation operator (&). (*Concatenation* means combining two text strings to form a new text string.) There are also numbers and text strings used as constants (meaning that they don't change), such as `14`, `"Dear"`, and `"Jones"`. Text strings are the only things you have to place within quotation marks.

*To learn about variables, **see** Chapter 16, "Advanced Scripting Techniques."*

Selecting Fields

In the calculation dialog, above the formula box and to the left is a list of fields. By default, the fields in the current table are listed. You can see the fields in a related (or unrelated) table by making a selection in the pop-up menu above the field list. Double-click a field name to insert it into your formula. You can also type field names directly.

*If you're having difficulty with field name syntax in formulas within ScriptMaker, **see** "Formulas in Scripts Require Explicit Table Context" in the "Troubleshooting" section at the end of this chapter.*

caution

Be aware that the only fields you can use from an unrelated table are those with global storage. There's no way FileMaker could determine which record(s) to reference for nonglobally stored fields. You get an error message if you attempt to use a nonglobal field from an unrelated table in a formula.

Choosing Operators

In between the field and function areas in the Specify Calculation dialog is a list of operators you can use in your formulas. *Operators* are symbols that define functions, including the math functions addition, subtraction, raising to a power, and so on.

There is often some confusion about the use of &, +, and the `and` operator. The ampersand symbol (&) is used to concatenate strings of text together, as in the previous example where we derive the `FullName` by stringing together the `FirstName`, a space, and the `LastName`. The + symbol is a mathematical operator used, as you might expect, to add numbers together. The `and` operator is a logical operator used when you need to test for multiple Boolean conditions. For instance, you might use the formula `Case (Amount Due > 0 and Days Overdue > 30, "Overdue")`. Here, the `and` indicates that both conditions must be satisfied for the test to return true.

note

Strictly speaking, not all the symbols listed here are operators. The ¶ paragraph symbol (or pilcrow), for instance, is used to represent a literal return character in strings. The symbols and concepts available in the Operators section are common to many programming and scripting languages; there are no FileMaker-specific concepts in this part of the dialog.

The other operators are quite intuitive, with the exception of `xor`. `xor`, which stands for *exclusive or*, tests whether either of two statements is true, but not both of them. That is, `(A xor B)` is the same thing as `(A or B) and not (A and B)`. The need for such logic doesn't come up often, but it's still handy to know.

Selecting Functions

The upper-right portion of the Specify Calculation dialog contains a list of the functions you can use in your formulas. By default, they are listed alphabetically, but you can use the View pop-up menu above the list to view only formulas of a certain type. The `Get` functions and `External` functions, in fact, will display only if you change to View by Type.

Double-clicking a function inserts the function into your formula at the current insertion point. Pressing Control and the spacebar (Macintosh) or the Insert key (Windows) while highlighting the

function also adds it to your formula. The guts of the function—the portion between the parentheses—is highlighted so that you can begin typing parameters immediately.

To learn more about how to read and use functions, see "Parts of a Function," p. 229.

Writing Legible Formulas

Whether you're typing in a formula by hand or are using the selection lists to insert fields and functions, here are a few general comments about how to make your functions easy to read. First, when you're writing functions, spacing, tabs, and line returns don't matter at all except within quotation marks. You can put spaces, tabs, and returns just about any place you want without changing how the formula evaluates. For legibility, it's therefore often helpful to put the parameters of a function on separate lines, especially when you have nested functions. In its own formatting, FileMaker leaves spaces between values and parentheses; this can make for more easily read code.

You can also add comments to calculation formulas. You can prefix a comment with two forward slashes (`//`) and anything following until the end of that line will not be evaluated. To comment a block of multiple-lined text, begin with `/*` and close with `*/`.

Figure 8.3 shows a calculation from the Time Billing Starter Solution with a few modifications. As you can see in the lower left of the dialog, the result of the calculation will be a Text value. This function fills the `Customer Name` field with a value selected by a `Case` statement.

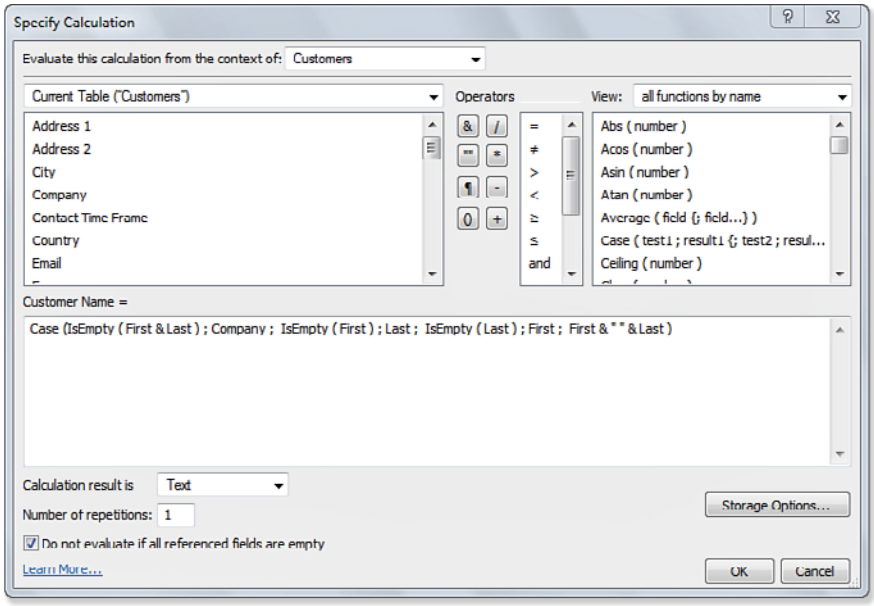


Figure 8.3
A complex formula written without adequate spacing can be very difficult to understand and troubleshoot.

About `Case` Statements and Functions

If you are familiar with programming languages, this is the same construct you know in that context. If you are not familiar with programming languages, see the section on "Using Conditional Functions," later in this chapter.

This is a typical calculation field. It evaluates several statements based on data fields and returns a value for the `Customer Name` field. Because it is spaced out, it is clear to see the following:

- If both `First` and `Last` (name) are empty, set `Customer Name` to the `Company` field.
- Otherwise, if `First` is empty, set `Customer Name` to `Last`.
- Otherwise, if `Last` is empty, set `Customer Name` to `First`.
- Otherwise, set `Customer Name` to `First`, a space, and then `Last`.

note

Lining up sections of code by indenting them helps to make it much easier to see errors and logical flaws.

That is the code you see in Figure 8.3. Some people would argue that it is quite clear, and because the statement appears on a single line, it is easy to see what is happening.

In Figure 8.4, that same formula has been reformatted with extra spacing to make it more legible. Legibility isn't merely an idle concern; it has real value. If you, or someone else, ever need to debug or alter one of your formulas, it will take much less time and effort if you've formatted your formula well in the first place. And that is why the actual calculation field in the Time Billing Starter Solution looks as shown in Figure 8.4.

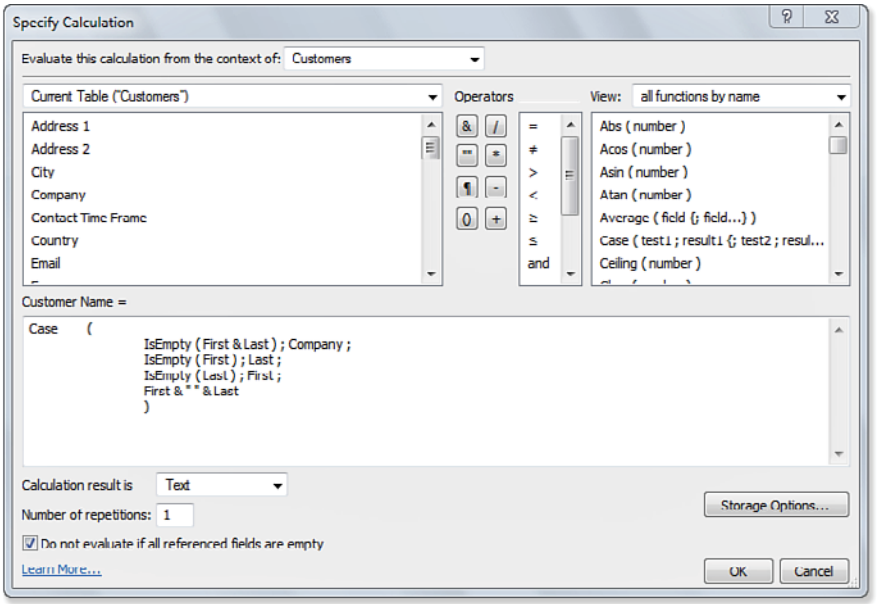


Figure 8.4
Adding spaces, returns, and comments to a formula can make it much more legible, and hence easier to maintain in the future.

Options

At the bottom of the Specify Calculation dialog, you can see a variety of miscellaneous options. These options pertain only to defining calculation fields; you don't see them in any of the other calculation dialogs, such as those used to evaluate an `If` statement.

Data Type

The first of these miscellaneous options is to specify the type of data the calculation will return. Usually, it's obvious. If you're concatenating the `FirstName` and `LastName` fields to form the `FullName` field, your calculation result will need to be a text string. If you're adding the `SalesTax` to an `InvoiceSubTotal` to generate the `InvoiceTotal`, the expected result will obviously be a number. Adding 14 days to the current date to generate a callback date should result in a date. Simply ask yourself what type of data the formula should produce and select the appropriate result.

If you choose the wrong data type for a calculation field, you might experience some unexpected results. ***See*** *"Errors Due to Improper Data Type Selection" in the "Troubleshooting" section at the end of this chapter.*

Number of Repetitions

The only time you'll ever have to worry about the number of repetitions in a calculation field is when your formula references one or more repeating fields. If it does, you'll typically define your calculation to have the same number of repetitions as the fields it references. The formula you define is applied to each repetition of the source fields, resulting in different values for each repetition of your calculation field.

If you reference nonrepeating fields in your calculation, they affect only the first repetition of output. You can, however, use the `Extend()` function to allow a nonrepeating field to be applied to each repetition of output.

Do Not Evaluate

By default, for new calculation fields, the Do Not Evaluate If All Referenced Fields Are Empty box on the Specify Calculation dialog is checked. This means that the calculation returns a null (empty) value as long as all the fields it refers to are empty. If this box is unchecked, the formula will be evaluated using the empty values in the referenced fields. For instance, say that you had a `StatusCode` field in an invoice database and wanted to use it to generate a status message, the formula of which was `If ( StatusCode = "P"; "Paid"; "Not Paid" )`. If you left the Do Not Evaluate... box checked, invoices with no status code would have no status message. If it were unchecked, their status message would be `Not Paid`.

Another example draws from this feature's most common use: financial calculations. If you have a field that calculates, say, a price total based on quantity and sales tax fields, it's often helpful to return an explicit zero rather than leaving the calculation field null or blank. Consider a calculation field that calculates a discount based on a `transactionAmount` field:

```
If ( transactionAmount >= 1000; 50; 0 )
```

If the check box is unchecked, this evaluation will return a zero if either `transactionAmount` is less than 1000 or the field is empty. In this way, the zero is explicit and demonstrates for the user that the calculation was performed. If the check box is left checked and `transactionAmount` is empty, this `discount` field will be empty as well, leading to possible uncertainty on the part of users.

Storage Options

The last things in this anatomy lesson are the storage options available when you're defining calculation fields. Be aware that the output of your calculation formula may differ depending on the storage method selected. The Storage Options dialog is shown in Figure 8.5.

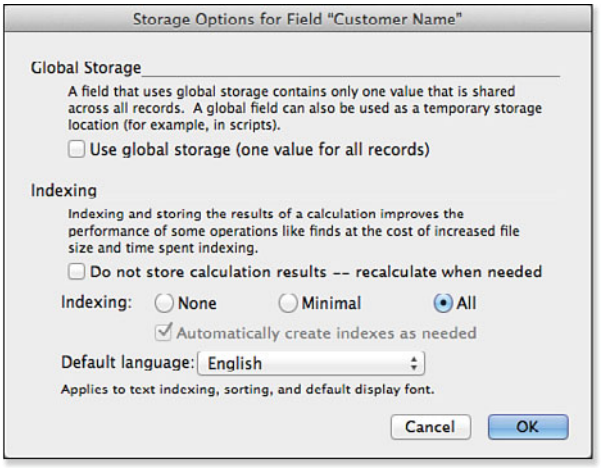


Figure 8.5
The Storage Options dialog enables you to set calculation fields so that they have global results and to specify indexing options.

note

There's no simple rule as to when you want to check or uncheck this option. You need to look at your formula and determine whether the inputs to the formula—those fields referenced in the formula—could all ever be blank, and if so, whether you would still want the formula to evaluate. Typically, if your formulas have default results (as in the `StatusCode` example), rather than using explicit logic for determining results, you probably want to uncheck the box.

In the top portion of the dialog, you can specify global storage as an option. This is a concept introduced in FileMaker Pro 7 and one perhaps not immediately intuitive even for longtime FileMaker developers. Global storage for regular fields (that is, text, number, date, time, timestamp, and container) is typically used when you need a temporary storage location for a value or for infrequently changing, solutionwide values such as your company's name and address. For instance, globally stored text fields are often used in scripts as a place to hold users' preferences or selections as they navigate through your interface.

*For more information on global storage of field data, **see** "Storage and Indexing," **p. 104**.*

If you set a calculation field to be stored globally, the results of the calculation formula will be available to you from any record and, indeed, any table in your system without having to establish a relationship to a table occurrence tied to its source table. The formula isn't evaluated for each record in the system; it is evaluated only when one of the inputs of the formula changes or when you modify the formula.

Consider a scenario involving a sales commission calculation. You might create a utility table containing the fields necessary to calculate a daily sales commission (based on market values or whatever variable data affected the business in question) in which a manager could modify the data in the formula on demand. A global calculation then would provide the system with its current sales commission without requiring a series of relationships.

The bottom half of the Storage Options dialog enables you to specify indexing options. Indexing a field speeds up searches based on that field, but it results in larger files. FileMaker also uses field indexes for joining related tables.

note

Note that the sales commission example assumes there to be one record in the utility table in question. If there were multiple records, it would be possible to include the concept of an active/inactive status into the calculation or simply rely on the fact that the last edited record will be that from which the calculation will draw its source information.

*For more detailed information on indexing, **see "Storage and Indexing," p. 104.***

In most cases, the default indexing option for a calculation field will be set to None, and the Automatically Create Indexes as Needed box will be checked. For most calculations you write, this configuration is perfect. FileMaker determines whether an index is needed and creates one if it is. Performing a find in the field and using the field in a relationship are both actions that trigger the automatic indexing of a field.

For some calculation formulas, the default storage option is to have the Do Not Store Calculation Results option checked and for everything else to be grayed out. This is an indication that the field is unindexable. Calculation fields that return text, number, date, time, or timestamp results can be indexed as long as they are stored. Calculations can be stored as long as they don't reference any unstored calculations, globally stored fields, related fields, or summary fields. Not saving a calculation means that finds or sorts using the field will be slower than if it is stored. For a field that is frequently used for finds, this is a serious consideration; for other fields, it might be irrelevant.

There are a few circumstances in which you'll want to explicitly turn off storage. For instance, when you use any of the `Get` functions in a calculation, you should make sure that the calculation result is unstored. `Get` functions typically return information relating to the state of a user session. By definition, that information changes on a second-by-second basis, and formulas based on it should not be stored so that they continue to reflect present reality. If you do so, the calculation is forced to evaluate based on the current environment each time it's evaluated (as opposed to always "remembering" the environment at the time the record was created or modified). Imagine you defined a calculation to return the number of records in the current found set by using the `Get ( FoundCount )` formula. If you don't explicitly set the results to be unstored, for a given record, the formula evaluates once and keeps that value, regardless of changes to the size of the found set. The

count of found records the first time the calculation is triggered is the value that will be stored. As their name implies, unstored calculations do not make your files larger, but because they must evaluate each time you view them, they can slow down a system if they're based on complex formulas.

Specifying Context

Across the top of the Specify Calculation dialog, you're asked to specify the context from which to evaluate this calculation. This choice is necessary only when the source table you are working with appears in your Relationships Graph more than once as several table occurrences. And even in those cases, it really matters only when your calculation formula involves related fields. In such cases, the calculation might return different results, depending on the context from which it's evaluated. To make this point clear, consider the example of a database that contains transactions for buyers and sellers.

There are two tables in the database: Persons and Transactions. Figure 8.6 shows the Transactions table in Table view.

In this example, a person can act as either a buyer or a seller for a given transaction. This means then that a Persons record will have potentially two sets of related transactions: those for which that person is a seller and those for which she is a buyer.

note

As a rule of thumb, you should stick with the default storage options unless you know for sure that you need the result to be unstored. You'll almost never need to explicitly turn indexing on; let FileMaker turn it on as necessary. Very seldom should you uncheck the option to have FileMaker turn on indexing as needed. Be aware that indexing increases the size of your files, sometimes by a great deal. By unchecking the option to have FileMaker turn on indexing as needed, you can ensure that certain fields won't be indexed accidentally just because a user performs a find on them.

note

This database, CalculationDemo, can be downloaded from the author's website as described in the Introduction.

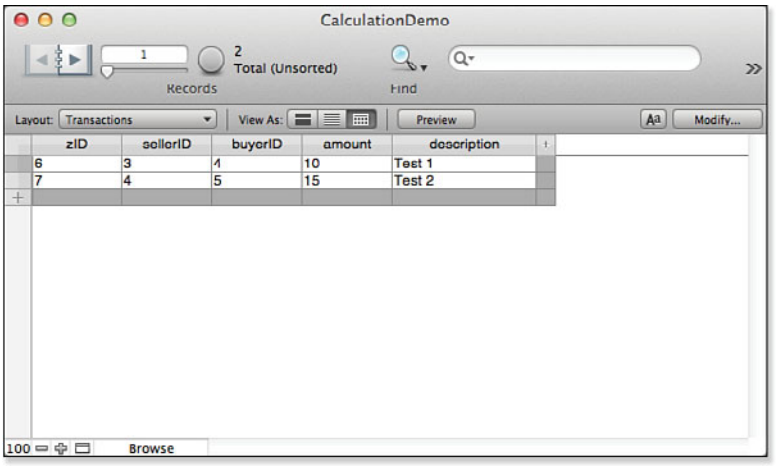


Figure 8.6
Transactions are stored with the IDs of buyer and seller as well as amount and description.

Figure 8.7 shows the Relationships Graph of the database. Note that there is a single Transactions table and three occurrences of the Persons table: one is named Buyers, one is named Sellers, and one has the default name Persons. Buyers and Sellers are related to Transactions by relationships between sellerID or buyerID in Transactions and the zID in the appropriate table occurrence of Persons (Buyers or Sellers).

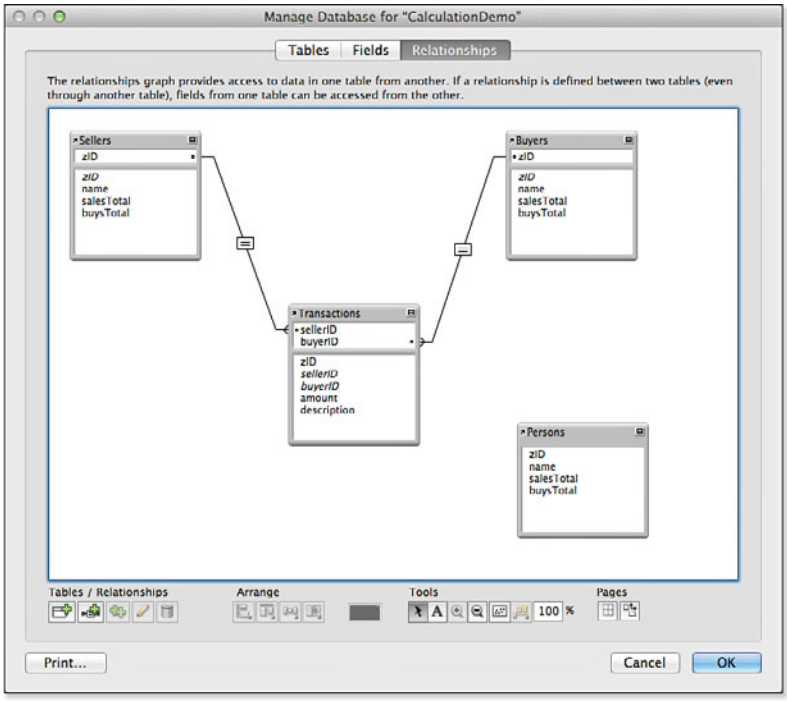


Figure 8.7
The two tables (Persons and Transactions) are the basis for four table occurrences.

Note, too, that in the Persons table (as well as the Sellers and Buyers occurrences based on it) are two additional fields: `salesTotal` and `buysTotal`. These calculation fields contain the total amount of sales and purchases for that person.

The relationship between Buyers, Sellers, and Transactions enables you to construct a layout such as the one shown in Figure 8.8. It is based on the Sellers table occurrence, and it has a portal showing the Transactions table to which the seller is related. A copy of this layout, based on the Buyers table occurrence, has a portal for Transactions based on the relationship between Buyers and Transactions.

To create the `salesTotal` field in the Persons table, you need to use a function that sums up all sales—that is, all records in the Transactions table that are found via the relationship between Sellers and Transactions. Likewise, to create the `buysTotal` field, you need a calculation that sums up all records in the Transactions table found via the relationship between Buyers and Transactions.

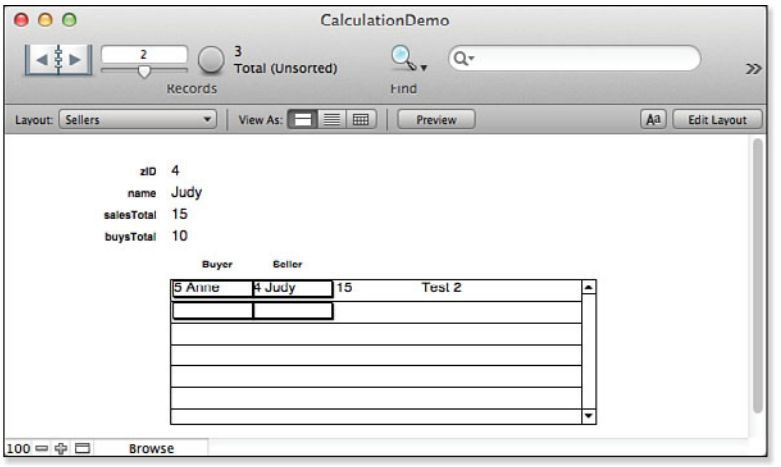


Figure 8.8
A Sellers layout contains a portal to Transactions.

The problem is that these two calculation fields are in the Persons table. You need a way to specify which relationship from a table occurrence of the Persons table is to be used. You do so by setting the *context* for the calculation using the pop-up menu at the top of the Specify Calculation dialog, as shown in Figure 8.9.

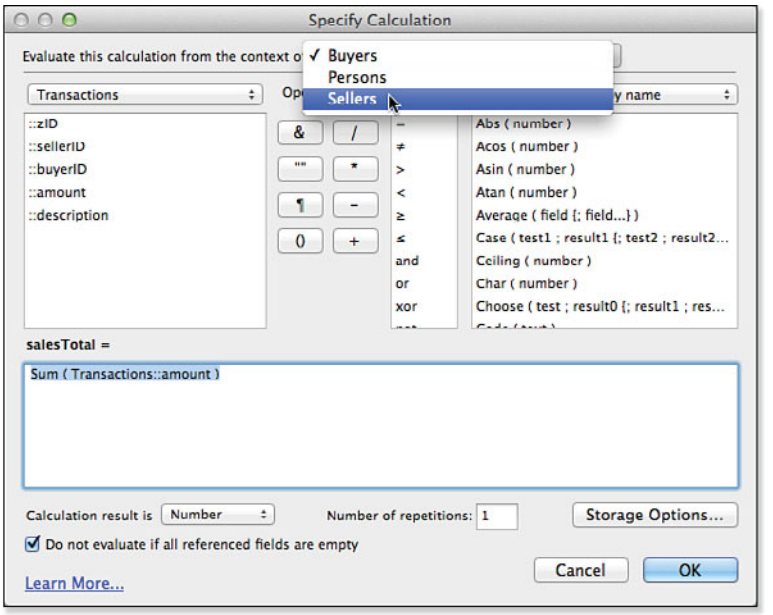
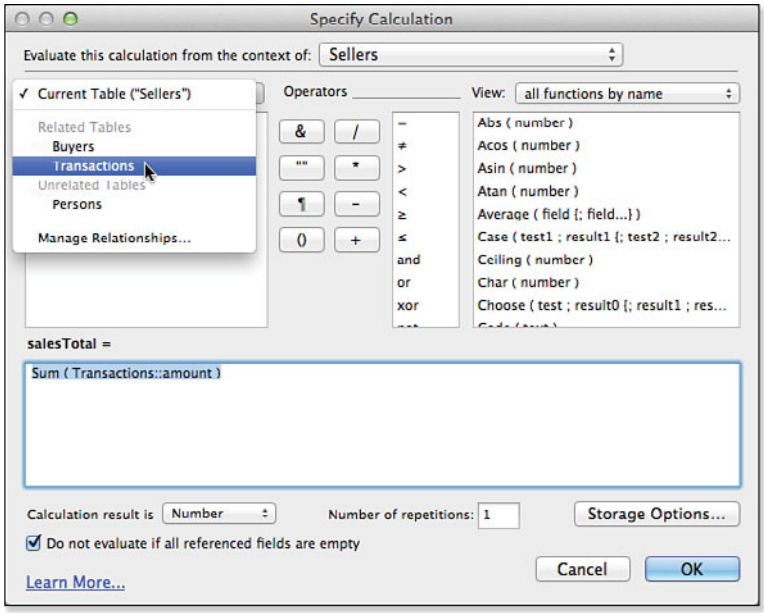


Figure 8.9
Specify the context for an ambiguous relationship.

Having specified the context, you then add the `Sum` function and, for the field to be summed, you select the table as shown in Figure 8.10 and then select the field.

Figure 8.10
Select the table and field to be used.



As you saw from the same layout in Figure 8.8, this calculation correctly computes the totals needed. You only need to specify a context for a calculation when there is ambiguity. The ambiguity arises if the table in which you are creating a calculation has more than one relationship to the table in which a related field resides.

Essential Functions

Now that you know your way around the Specify Calculation dialog itself, it's time to start learning more about particular calculation functions. Here is an in-depth tutorial on the most essential functions and techniques. These will form a solid base for your own work and for assembling complex formulas. As a reminder, Chapter 15 covers advanced calculation formulas and techniques.

Parts of a Function

Let's begin with a general discussion about what functions do and how to learn about them. Their sole mission in life is to act on some set of inputs and produce an output. The inputs are usually referred to as *parameters*; the function's syntax specifies the number of parameters it expects to be fed and provides a clue about what the nature of each of those parameters is.

An example will help clarify this point. Look at the syntax of the `Position` function as it's taken directly from the function list in the calculation dialog:

```
Position ( text ; searchString ; start ; occurrence )
```

A function's parameters are always placed in parentheses directly after the name of the function itself. They are separated from one another by semicolons.

note

In English versions prior to FileMaker 7, the parameter separator was a comma. In fact, if you use commas now, they are transformed into semicolons for you.

You can see that the `Position` function has four parameters. The function reference will tell you that the first parameter should be a text string in which you want to search, and the second should be a text string you want to find within it. The third parameter is a number that specifies the character number at which to begin searching. The final parameter is also a number; it specifies which occurrence of the search string to find.

Besides knowing what to feed a function (here, two text strings and two numbers), you also need to know what type of output the function produces. Again, you first learn this by consulting some reference source or the help system. There, you'd learn that the `Position` function returns a number—not just any number, of course, but the character number where the search string was found within the initial text string. If the string was not found at all, it returns a `0`. So, for example, if you had the function

```
Position ( "Mary had a little lamb"; "a"; 1; 1 )
```

the function would return `2` because the first occurrence of the letter *a* is at character 2 of the input string. If you change the function slightly, to

```
Position ( "Mary had a little lamb"; "a"; 1; 2 )
```

you'd now expect a value of `7` because that's the position of the second occurrence of the letter *a*.

In these examples, all the parameters were hard-coded with constant values. More typically, the parameters that you feed a function will be either fields or the outputs of other functions. For instance, if you have a field called `PoemText` and another called `SearchCharacter`, you might end up using the `Position` function as shown here:

```
Position ( PoemText; SearchCharacter; 1; 1 )
```

Now, each record in your database will contain a different result, dependent on the contents of those two fields.

Using functions as parameters of other functions is called *nesting*. In those cases, the inner functions evaluate first, and their results are used as the inputs for the outer functions. For instance, you might have the following function:

```
Position ( PoemText; SearchCharacter; Length( PoemText ) - 5; 1 )
```

Notice that the third parameter of the `Position` function here is the expression `Length( PoemText ) - 5`. The `Length` function (which we discuss in more detail shortly) takes a single parameter, a text string, and returns the number of characters in the string. So, in the preceding function, the length of the `PoemText` field will be determined, and that value less 5 will be used as the third parameter of the `Position` function. No practical limit exists on the number of layers you can use to nest functions within one another. Just remember that readability becomes very important as your calculations become more complex.

At this point, you know quite a bit about the `Position` function. You know about its inputs and outputs; you've worked with a few examples. Eventually, you'll likely want to memorize the inputs and outputs of a core set of functions. For lesser-used functions, you can look up the parameters and usage on an as-needed basis. There's still a difference between proficiency with a function and a complete understanding of it. For instance, to truly master the `Position` function, you'd need to know such things as whether it's case sensitive (it's not), and what happens if you supply a negative number for the occurrence (it searches backward from the specified start character). Over time and with use, you'll learn about the subtle and esoteric usage of various functions, thereby moving from mere proficiency to mastery.

Let's turn now to a close look at those functions and techniques that should form the core of your calculation knowledge.

Text Operations

Text functions enable you to interrogate and manipulate text strings. If you haven't done much programming before, the concept of a string might require some explanation. Essentially, a *string* is a series of characters. Think about threading characters on a string like you do popcorn to make holiday decorations, and you'll have a good mental image of a text string. The characters can be anything from letters and numbers to spaces and punctuation marks.

note

In versions of FileMaker prior to 7, the size limit for text strings was 64,000 characters. It has now been expanded to a whopping 2GB.

Typically in FileMaker, text strings are found in text fields, but be aware that you can treat any numeric, date, and time data as a text string as well. When you do that, it's called *coercing* the data. FileMaker automatically coerces data into the type expected for a given operation. If you ever need to override the automatic coercion for any reason, you can use the `GetAs` functions. They include `GetAsDate()`, `GetAsNumber()`, `GetAsTime()`, and `GetAsText()`.

The simplest text operation you can perform is concatenation. *Concatenation* means taking two or more text strings and placing them beside each other to form a new, longer text string. As an example, consider the following formula:

```
FirstName & " " & LastName
```

Here, we're taking three strings, two of which happen to be field data, and we're concatenating them into a full name format.

Let's look next at several functions that can be used to interrogate text strings. By *interrogate*, we mean that we're interested in answering a specific question about the contents of a text string. For

the examples in this section, assume that you have a field called `fullName` with the string `"Fred Flintstone"` and the field `someString`, which contains `"The quick brown fox jumped over the lazy dog"`. The following is a list of some of the core calculation functions with examples that apply to the `fullName` and `someString` fields:

- `Length ( text )`—The `Length` function takes a single argument and simply returns the number of characters in the string. Remember that spaces and return characters are considered characters. So, `Length ( fullName )` would return `15`.
- `PatternCount ( text; searchString )`—The `PatternCount` function tells you the number of times a search string occurs within some string. As an example, `PatternCount ( someString; "the" )` would return `2`. Note that this function is *not* case sensitive. If the search string is not found, the function returns `0`. Although the function returns an integer, it's often used as a true/false test when you just want to know whether something is contained in a string. That is, you don't care where or how many times the string is found; you just care that it's there somewhere. Recall that any nonzero value represents "true" when being used as a Boolean value.
- `Position ( text; searchString; start; occurrence )`—You've already looked in depth at the `Position` function. To recap, it returns an integer that specifies the place where one string is found in another. The `start` argument specifies where to begin the search; the `occurrence` argument specifies whether you want the first occurrence, the second, and so on. Much of the time, you'll simply use `1` for both the `start` and the `occurrence` parameters.
- `WordCount ( string )`—`WordCount` is similar to the `Length` function, except that instead of counting every character, it counts every word. So, `WordCount ( someString )` would return `9` because there are nine words in the phrase. Be careful if you use `WordCount` that you have a good understanding of what characters FileMaker considers as being word delimiters.

What's in a Word?

Several FileMaker functions, such as `WordCount()`, `LeftWords()`, `RightWords()`, and `MiddleWords()`, treat text strings as collections of words rather than as collections of characters. But how does FileMaker determine what constitutes a word? It's actually quite simple. FileMaker recognizes a handful of characters as word separators. Spaces and carriage returns are both word separators, as you'd probably expect. Additionally, just about every punctuation symbol and other special character is considered a word separator. The two exceptions are worth knowing: Neither a period (.) directly after a number nor an apostrophe (') is a word separator. Also, in versions of FileMaker prior to 7, hyphens (-) were *not* considered word separators, but they are now. If you have multiple word separators right next to each other, they're considered together as a single delimiter. For instance, the string `" hello ,-, world "` is considered to have two words, even though there are a total of nine word separators in the string.

- `Exact ( originalText, comparisonText )`—The `Exact` function takes two strings as its inputs, and it compares them to see whether they are exactly the same string. It returns a `1` if they are, a `0` if not. By "exactly," we mean *exactly*; this function is case sensitive. The order of the two input arguments is irrelevant.

The other broad category of text operators consists of those functions that enable you to manipulate a string. Whereas the interrogatory functions returned a number, these functions all return a string. You feed them a string; they do something with it and spit back another string. The text operators that fall into this category are explained in the following sections.

Trim()

The simplest of these functions is the `Trim ( text )` function. `Trim()` takes a string and removes any leading or trailing spaces from it. Spaces between words are not affected; no other leading or trailing characters other than a space (that is, return characters at the end of a field) are removed.

There are two common uses of `Trim()`. The first is to identify data-entry problems. Imagine you have a field called `FirstName` and that some users have been accidentally typing spaces after the first name. You might want to display a message on such records, alerting users to that error. You'd define a new calculation field, called something like `SpaceCheck`. Its formula could be one of the following:

```
Case ( FirstName ≠ Trim ( FirstName ), "Extra Space!" )
Case ( not Exact( FirstName, Trim ( FirstName )), "Extra Space!" )
Case ( Length( FirstName ) > Length( Trim( FirstName )), "Extra Space!" )
```

To review the use and syntax of the `Case()` *function,* ***see "Using Conditional Functions," p. 240.***

The other common usage of `Trim()` is in a calculated replace to clean up fixed-length data that has been imported from another application. *Fixed length* means that the contents of a field are padded with leading or trailing spaces so that the entries are all the same length. After importing such data, you'd simply replace the contents of each field with a trimmed version of itself.

Substitute()

The next text manipulation function we'll explore is the `Substitute()` function. `Substitute ( string; searchString; replacementString )` is used to replace every occurrence of some substring with some other substring. So `Substitute( fullName; "Fred"; "Wilma")` would return the string `"Wilma Flintstone"`. If the initial substring were not found, the `Substitute` function would simply return the original string. You should be aware that the `Substitute()` function is case sensitive.

One common use of `Substitute()` is to remove all occurrences of some character from a string. You just substitute in an empty string for that character. For instance, to remove all occurrences of a carriage return from a field, you could use `Substitute ( myString; "¶"; "")`. If there are multiple substitutions you want to make to a string, you simply list them all as bracketed pairs in the

order in which they should be performed. Let's say you have a PhoneNumber field from which you want to strip out any parentheses, dashes, or spaces that users might have entered. One way to do this would be to use the following formula:

```
Substitute (PhoneNumber; ["("; ""] ; [")"; ""] ; ["-"; ""] ; [" ", ""])
```

Be aware when performing multiple substitutions like this that the substitutions happen in the order in which they are listed, and that each subsequent substitution happens on an altered version of the string rather than on the original string. Say you had the string "xxyz" and you wanted to put *z*'s where there are *x*'s, and *x*'s where there are *z*'s. The formula Substitute ("xxyz"; ["x"; "z"]; ["z"; "x"]) incorrectly returns "xxyx". First, the two leading *x*'s are turned to *z*'s, yielding "zzyz"; then all three *z*'s are turned into *x*'s. If you ever want to swap two characters like this, you need to temporarily turn the first character into something you know won't be found in your string. So to fix this example, we could use the formula Substitute("xxyz"; ["x"; "**TEMP**"]; ["z"; "x"]; ["**TEMP**", "z"]). That would correctly yield "zzyx".

Case-Altering Functions

You can use a few text functions to alter a string's case. These are Lower (text), Upper (text), and Proper (text). It's quite intuitive how these act. Lower ("Fred") returns "fred"; Upper ("Fred") returns "FRED". Using Proper() returns a string in which the first letter of each word is capitalized. For instance, Proper ("my NAME is fred") returns "My Name Is Fred".

Text-Parsing Functions

The final category of text operators we'll look at here is text-parsing functions. Text-parsing functions enable you to extract a substring from a string. The six text-parsing functions are Left(), Middle(), Right(), LeftWords(), MiddleWords(), and RightWords(). The first three operate at the character level; the other three operate at the word level.

The Left() function extracts a substring of length N from the beginning of some string. For example, Left ("Hello"; 2) returns the string "He"; it simply grabs the first two characters of the string. If the number of characters you ask for is greater than the length of the string, the function simply returns the entire string. A negative or zero number of characters results in an empty string being returned.

The Right() function is similar, except that it grabs characters from the end of the specified string. Right ("Hello"; 2) would return "lo". Middle(), as you might expect, is used to extract a substring from the middle of a string. Unlike the Left() and Right() functions, which require only a string and a length as parameters, the Middle function requires a starting position. The syntax is Middle (text; startCharacter; numberOfCharacters). For example, Middle ("Hello"; 2; 3) yields "ell".

The LeftWords(), MiddleWords(), and RightWords() functions all operate exactly as Left(), Middle(), and Right() functions, except that they operate at the word level. One typical use of these functions is to extract names or addresses you've imported as a lump of data from some other application. Say that your import resulted in contact names coming in as full names. You might want

to create a `LastName` calculation field so that you could sort the records. If you knew that the last name was always the last word of the `FullName` field, you could use the formula `RightWords ( FullName; 1 )`.

Nested Functions

The text operators we discussed often appear nested within each other in formulas. Writing nested formulas can be tricky sometimes. One thing that helps is to think of a particular example rather than trying to deal with it abstractly. For instance, let's say that you have a big text field, and you need a formula that extracts just its first line—that is, everything up until the first carriage return. So, imagine that you had the following text:

```
The quick
brown fox
jumped over the
lazy dog
```

Think first: What text-parsing formulas would potentially yield `"The quick"` from this text? Well, there are several of them:

```
Left (myText; 9)
LeftWords (myText; 2)
Middle (myText; 1; 9)
```

Of course, at this point these formulas apply only to this particular example. Think next: Could one of these be extended easily to *any* multiline text field? If there were a constant number of words per line, the `LeftWords()` formula would work. And if not? What do the text interrogation formulas tell us about this field? `Length ( myText )` is 44. Not particularly helpful. `PatternCount ( myText; "¶" )` is 3. This indicates that there are four lines total. Interesting, but not obviously helpful for extracting the first line. `WordCount ( myText )` is 9. It's just coincidence that this is the number of characters in the first line; be careful not to be misled. `Position ( myText; "¶"; 1; 1 )` is 10. Finally, something interesting. In this example, the length of the first line is one less than the position of the first carriage return. Is that true in all cases? At this point, if you write out a few more examples, you'll see that indeed it is. Therefore, a general formula for extracting the first line of text is

```
Left ( myText; Position( myText, "¶"; 1; 1 ) - 1 )
```

How about extracting the *last* line from any multiline text field? You should approach this problem the same way, working from a specific example. Counting characters by hand, assemble a list of options:

```
Middle ( myText; 36, 8 )
Right ( myText; 8 )
RightWords ( myText; 2 )
```

What clues do the interrogatory functions yield? If you spend a few minutes thinking about it, you'll realize that 36 is the position of the last return character. You can derive that by using the number of returns as the occurrence parameter in a `Position()` function, like this:

```
Position ( myText; "¶"; 1; PatternCount( myText; "¶" ))
```

After you have the 36 figured out, recall that the length of the string is 44 characters, and notice that 44 – 36 = 8. Given these discoveries, you'll soon see that a simple and elegant generalized formula for grabbing the last line of a text field is

```
Right (myText; Length ( myText ) - Position( myText; "¶"; 1;
➥PatternCount( myText; "¶" )))
```

Number Functions

In general, most people find working with math functions simpler and more intuitive than working with string functions. Perhaps the reason is that they remind us of various high-school math courses. Or it could be they typically have fewer parameters. Regardless, you'll find yourself using number functions on a regular basis. This chapter focuses not so much on what these functions do, but rather on some interesting applications for them.

The first set of functions we'll look at includes `Int()`, `Floor()`, `Ceiling()`, `Round()`, and `Truncate()`. Each of these can be thought of as performing some sort of rounding, making it sometimes difficult to know which one you should use. You can look up these functions in the help system for complete syntax and examples, but it's helpful to consider the similarities and differences of these functions as a set. Here's a rundown:

- `Int ( number )`—The `Int()` function returns the integer portion of the number that it's fed—that is, anything before the decimal point. `Int ( 4.5 )` returns `4`. `Int ( -2.1 )` returns `-2`.
- `Floor ( number )`—`Floor()` is similar to `Int()`, except that it returns the next lower integer of the number it's fed—unless that number is an integer itself, of course, in which case `Floor()` just returns that integer. For any positive number, `Floor ( number )` and `Int ( number )` return the same value. For negative numbers, though, `Floor ( number )` and `Int ( number )` don't return the same value unless number is an integer. `Floor ( -2.1 )` returns `-3`, whereas `Int ( -2.1 )` returns `-2`.
- `Ceiling ( number )`—The `Ceiling()` function is complementary to the `Floor()` function: It returns the next higher integer from the number it's fed (unless, again, that number is already an integer). For example, `Ceiling ( 5.3 )` returns `6` and `Ceiling ( -8.2 )` returns `-8`.
- `Round ( number; precision )`—`Round()` takes a number and rounds it to the number of decimal points specified by the precision parameter. At the significant digit, numbers up to 4 are rounded down; numbers 5 and above are rounded up. So, `Round ( 3.6234; 3 )` returns `3.623`, whereas `Round ( 3.6238; 3 )` returns `3.624`. Using a precision of `0` rounds to the nearest whole number. Interestingly, you can use a negative precision. A precision of `-1` rounds a number to the nearest 10; `-2` rounds to the nearest 100, and so on.

- `Truncate ( number, precision )`—`Truncate()` is similar to `Round()`, but `Truncate()` simply takes the first `n` digits after the decimal point, leaving the last one unaffected regardless of whether the subsequent number is 5 or higher. `Truncate ( 3.6238; 3 )` returns `3.623`. For any number, `Truncate ( number; 0 )` and `Int ( number )` return the same value. Just as `Round()` can take a negative precision, so can `Truncate()`. For example, `Truncate ( 258; -2 )` returns `200`.

Which function you use for any given circumstance depends on your needs. If you're working with currency and want to add an 8.25% shipping charge to an order, you'd probably end up with a formula such as `Round ( OrderTotal * 1.0825 ; 2 )`. Using `Truncate()` might cheat you out of a penny here or there.

`Floor()`, `Ceiling()`, and `Int()` have some interesting uses in situations where you want to group numeric data into sets. For instance, imagine you have a list report that prints ten records per page and that you have a found set of 57 records to print. If you want, for whatever reason, to know how many pages your printed report would be, you could use `Ceiling ( Get( FoundCount )/10 )`. Similarly, if you want to know what page any given record would print on, you would use the formula `Floor ( (Get( RecordNumber )-1 )/10 ) + 1`. The `Int()` function would yield the same result in this case.

Another common use of these functions is to round a number up or down to the multiple of some number. As an example, say you had the number 18, and you want to know the multiples of 7 that bounded it (...14 and 21). To get the lower bound, you can use the formula `Floor ( 18/7 )* 7`; the upper bound is `Ceiling ( 18/7 )* 7`. These generalize as the following:

```
Lower bound:  Floor ( myNum / span ) * span
Upper bound:  Ceiling ( myNum / span ) * span
```

The `span` can be any number, including a decimal number, which comes in handy for rounding currency amounts, say, to the next higher or lower quarter.

You should know a few other number functions as well:

- `Abs ( number )`—The `Abs()` function returns the absolute value of the number it's fed. One interesting use of the function is to determine which platform FileMaker Pro is running on. The three possible return values are –1 for PowerPC-based Macs, 1 for Intel-based Macs, and –2 for Windows. `Abs ( Get ( SystemPlatform ) )` returns 1 for both flavors of Macs.
- `Mod ( number; divisor )`—The `Mod()` function returns the remainder when a number is divided by a divisor. For instance, `Mod ( 13; 5 )` returns `3` because 13 divided by 5 is 2, remainder 3.
- `Div ( number; divisor )`—The `Div()` function is complementary to the `Mod()` function. It returns the whole-number result of dividing a number by a divisor. For instance, `Div ( 13; 5 )` would return `2`. In all cases, `Div ( number; divisor )` and `Floor ( number/divisor )` return exactly the same value; it's a matter of personal preference or context which you should use.
- `Random()`—The `Random()` function returns a random decimal number between 0 and 1. Usually, you'll use the `Random()` function when you want to return a random number in some other range, so you'll need to multiply the result of the function by the span of the desired range. For

instance, to simulate the roll of a six-sided die, you use the formula `Ceiling ( Random * 6 )`. To return a random integer between, say, 21 and 50 (inclusive), the method would be similar: First, you generate a random number between 1 and 30, and then you add 20 to the result to translate it into the desired range. The formula would end up as `Ceiling ( Random * 30 ) + 20`.

Character Functions

In their simplest form, these character functions let you convert a character to its numeric code, and vice versa. This capability is particularly useful in dealing with special characters such as the Return key. Because FileMaker Pro supports Unicode, these functions actually work on Unicode code points, but many people refer to ASCII codes, which are a subset of Unicode code points:

- `Char ( number )`—This function returns a character for a Unicode code point. It may return more than one character if the `number` string describes multiple characters.
- `Code ( text )`—This is the reverse of the preceding function: It provides a text string (often a single character), and the function returns its numeric value. This function is often used with `Get ( TriggerKeystroke )`, which is described in "`Get` Function," later in this chapter.

Working with Dates and Times

Just as there are functions for working with text and numbers, FileMaker Pro provides functions that enable you to manipulate date and time fields. This section introduces you to the most common and discusses some real-world applications you'll be likely to need in your solutions.

The most important point to understand at the outset is how FileMaker itself stores dates, times, and timestamps. Each is actually stored as an integer number. For dates, this integer represents a serialized number beginning with January 1, 0001. January 1, 0001, is `1`; January 2, 0001, is `2`; and so on. As an example, October 19, 2013, is stored by FileMaker as `735160`. FileMaker understands dates from January 1, 0001, until December 31, 4000.

Times are stored as the number of seconds since midnight. Midnight itself is `0`. Therefore, times are typically in the range of `0` to `83999`. It's worth knowing that time fields can contain not only absolute times, but also elapsed times. That is, you can type `46:18:19` into a time field, and it will be stored as `166699` seconds. Negative values can be placed in time fields as well. FileMaker doesn't have the capability to deal with microseconds; however, it can manage fractional elements:

`10:15:45.99` is a valid time within FileMaker and `10:15:45.99 - 10:15:44 = 00:00:01.99`. Note that this is not hundredths of a second, but rather simply a case of using a decimal instead of an integer.

Timestamps contain both a date and time. For example, "10/19/2013 8:55:03 AM" is a timestamp. Internally, timestamps are converted to the number of seconds since midnight on January 1, 0001. You could derive this number from date and time fields with the formula `(( myDate - 1 ) * 86400 ) + myTime`.

The easiest way to begin learning date, time, and timestamp functions is to split them into two categories: those that you feed a date or time and that return a "bit" of information back, and those

that are *constructors*, in which you feed the function bits and you get back a date, time, or timestamp. These aren't formal terms that you'll find used elsewhere, but they're nonetheless useful for learning date and time functions.

The "bit" functions are fed dates and times, and they return numbers or text. For instance, say that you have a field `myDate` that contains the value `10/19/2013`. Here's a list of the most common "bit" functions and what they'd return:

```
Month ( myDate ) = 10
MonthName ( myDate ) = October
Day ( myDate ) = 19
DayName ( myDate ) = Saturday
DayOfWeek ( myDate ) = 7
Year ( myDate ) = 2013
```

Similarly, a field called `myTime` with a value of `9:23:10 AM` could be split into its bits with the following functions:

```
Hour ( myTime ) = 9
Minute ( myTime ) = 23
Seconds ( myTime ) = 10
```

You need to know only three constructor functions. Each is fed bits of data and returns, respectively, a date, time, or timestamp:

```
Date ( month; day; year )
Time ( hours; minutes; seconds )
TimeStamp ( date; time )
```

For example, `Date ( 10; 20; 2013 )` returns `10/20/2013`. `TimeStamp ( myDate; myTime )` might return `10/19/2013 9:23:10 AM`. When using these formulas in calculation fields, be sure to check that you've set the calculation result to the proper data type.

note

One very interesting and useful fact to know about these constructor functions is that you can "overfeed" them. For example, if you ask for `Date ( 13; 5; 2013 )`, the result will be `1/5/2014`. If the bits you provide are out of range, FileMaker automatically adjusts accordingly. Even zero and negative values are interpreted correctly. `Date ( 10; 0; 2013 )` returns `9/30/2013` because that's one day before 10/1/2013.

Using Date and Time Functions

There are many practical uses of the date and time functions. For instance, the "bit" functions are often used to generate a break field that can be used in subsummary reports. Say that you have a table of invoice data, and you want a report that shows totals by month and year. You would define a field called `InvoiceMonth` with the formula `Month ( InvoiceDate )` and another called `InvoiceYear` with a formula of `Year ( InvoiceDate )`.

A common use of the constructor functions is to derive a date from the bits of a user-entered date. Say, for example, that a user entered `10/19/2013` into a field called `myDate`, and you wanted a calculation formula that would return the first of the next month, or `11/1/2013`. Your formula would be `Date ( Month( myDate ) +1; 1; Year( myDate ))`.

If you're importing dates from other systems, you might have to use text-manipulation functions in conjunction with the constructor functions to turn the dates into something FileMaker can understand. Student information systems, for example, often store students' birth dates in an eight-digit format of MMDDYYYY. To import and clean this data, you first bring the raw data into a text field. Then, using either a calculated replace or a looping script, you would set the contents of a date field to the result of the formula:

```
Date (
    Left ( ImportedDate; 2 );
    Middle( ImportedDate; 3; 2 );
    Right( ImportedDate; 4 )
)
```

Using Timestamps

Timestamps are quite useful for logging activities, but sometimes you'll find that you want to extract either just the date or just the time portion of the timestamp. The easiest way to do this is via the `GetAsDate()` and `GetAsTime()` functions. When you feed either of these a timestamp, it returns just the date or time portion of that timestamp. Similarly, if you have a formula that generates a timestamp, you can set the return data type of the calculation result to date or time to return just the date or just the time.

Using Conditional Functions

Conditional functions are used when you want to return a different result based on certain conditions. The most basic and essential conditional function is the `If()` function. If takes three parameters: a test, a true result, and a false result. The test needs to be a full equation or expression that can evaluate to true or false.

Let's look at an example. Suppose you have a set of records containing data about invoices. You'd like to display the status of the invoice—"Paid" or "Not Paid"—based on whether the `AmountDue` field has a value greater than zero. To do this, you'd define a new field, called `InvoiceStatus`, with the following formula:

```
If ( AmountDue > 0, "Not Paid", "Paid")
```

For each record in the database, the contents of the `InvoiceStatus` field will be derived based on the contents of that record's `AmountDue` field.

The test can be a simple equation, as in the preceding example, or it can be a complex test that uses several equations tied together with `and` and `or` logic. For the test

```
If ( A and B; "something"; "something else")
```

both A and B have to be true to return the true result. However, for the test

```
If ( A or B; "something"; "something else" )
```

if either A or B is true, it will return the true result.

The true or false result arguments can themselves be `If()` statements, resulting in what's known as a *nested* `If()` statement. This allows you to test multiple conditions and return more than two results. For instance, let's revise the logic of the `InvoiceStatus` field. Say that we wanted invoices with a negative `AmountDue` to evaluate as `Credit Due`. We could then use the following field definition:

```
If ( Amount Due > 0; "Not Paid"; If (Amount Due < 0; "Credit Due"; "Paid" ))
```

The other commonly used conditional function is the `Case()` statement. The `Case()` statement differs from the `If()` statement in that you can test for multiple conditions without resorting to nesting. For instance, say that you have a field called `GenderCode` in a table that contains either `M` or `F` for a given record. If you wanted to define a field to display the full gender, you could use the following formula:

```
Case ( GenderCode = "M"; "Male"; GenderCode="F"; "Female" )
```

A `Case()` statement consists of a series of tests and results. The tests are conducted in the order in which they appear. If a test is true, the following result is returned; if not, the next test is evaluated. FileMaker stops evaluating tests after the first true one is discovered. You can include a final optional result that is returned if none of the tests comes back as true. The gender display formula could be altered to include a default response as shown here:

```
Case ( GenderCode = "M"; "Male"; GenderCode="F"; "Female"; "Gender Unknown" )
```

Without the default response, if none of the tests is true, then the `Case()` statement returns a null value.

Aggregate Functions

Another important category of functions includes those known as *aggregate* functions. They include `Sum()`, `Count()`, `Min()`, `Max()`, and `Avg()`. These functions all work in similar, quite intuitive ways. Each operates on a set of inputs (numeric, except for the `Count()` function) and produce a numeric output. The name of the function implies the operation each performs. `Sum()` adds a set of numbers, `Min()` and `Max()` return (respectively) the smallest and largest items of a set, `Avg()` returns the arithmetic mean of the numbers, and `Count()` returns the number of non-null values in the set. `List()` returns a text field with the inputs concatenated together and separated by carriage returns.

The inputs for an aggregate function can come from any one of three sources:

- **A series of delimited values**—For example, `Sum ( 6; 4; 7; 2 )` yields `19`. `Average ( 6; 4; 7; 2 )` yields `4.75`. An interesting use of the `Count()` function is to determine the number of fields in a record into which a user has entered values. For instance, `Count ( FirstName; LastName; Phone; Address; City; State; Zip )` returns `2` if the user enters values into only those two fields.

- **A repeating field**—Repeating fields enable you to store multiple values within a single field within the same record. For instance, you might have repeating fields within a music collection database for listing the tracks and times of the contents of a given disc. The functions `Count ( Tracks )` and `Sum ( Times )` produce the number of tracks and the total playing time for a given disc.
- **A related field**—By far, this is the most common application for aggregate functions. Imagine that you have a Customer table and an Invoices table and you want to create a field in Customer that totals up all the invoices for a particular customer. That field would be defined as `Sum ( Invoices::InvoiceTotal )`. Similarly, to tell how many related invoices a customer had, you could use the formula `Count ( Invoices:: CustomerID )`.

note

When you are using the `Count()` function to count related records, it usually doesn't matter what field you count, as long as it's not empty. The count will not include records where the specified field is blank. Typically, you should count either the related primary key or the related foreign key because these, by definition, should contain data.

Learning About the Environment: Introspective Functions

FileMaker has two categories of functions whose job it is to tell you information about the environment—the computing and application environment, that is. These are the `Get()` functions and the `Design()` functions. There are more than 70 `Get()` functions and 20 `Design()` functions. Here, our goal is to give you an overview of the types of things these functions do and some of the most common uses for them.

`Get` Function

A `Get()` function provides a broad array of information about a user's computing environment and the current state of a database. Each takes a single parameter that identifies the type of information you want.

As an example, the `Get ( TotalRecordCount )` function returns the total number of records in some table. One typical use for this is as the formula for a calculation field. If you have hidden the Status Area from users, this field could be used as part of constructing your own "Record X of Y" display. If you're using this function in a script—or any `Get()` function, for that matter—be sure that you're aware that the active layout determines the context in which this function is evaluated.

Whenever you use a `Get()` function as part of a field definition, you need to be acutely aware of the storage options that have been set for that field. For `Get()` functions to evaluate properly, you must explicitly set the calculation to be unstored. If it is not set this way, the function evaluates only once when the record is created; it reflects the state of the environment at the time of record creation, but not at the current moment. Setting the calculation field to unstored forces it to evaluate every time the field is displayed or used in another calculation, based on the current state of the environment.

Frequently Used `Get` Functions

Although you don't need to memorize all the `Get()` functions, a handful of them are used frequently and should form part of your core knowledge of functions. To remember them, you might find it helpful to group them into subcategories based on their function.

The first subcategory includes functions that reveal information about the current user:

```
Get ( AccountName )
Get ( ExtendedPrivileges )
Get ( PrivilegeSetName )
Get ( UserName )
Get ( UserCount )
```

Another subcategory includes functions used frequently in conditional tests within scripts to determine what actions should be taken:

```
Get ( ActiveModifierKeys )
Get ( LastMessageChoice )
Get ( LastError )
Get ( ScriptParameter )
```

There are four functions for returning the current date and time:

```
Get ( CurrentDate )
Get ( CurrentHostTimeStamp )
Get ( CurrentTime )
Get ( CurrentTimeStamp )
```

Many `Get()` functions tell you where the user is within the application and what the user is doing:

```
Get ( FoundCount )
Get ( LayoutNumber )
Get ( LayoutName )
Get ( LayoutTableName )
Get ( PageNumber )
Get ( PortalRowNumber )
Get ( RecordNumber )
```

Another group of functions reveals information about the position, size, and name of the current window:

```
Get ( WindowName )
Get ( WindowTop )
Get ( WindowHeight )
Get ( WindowWidth )
```

The last set of functions lets you find two file paths. You can use them in scripts when you need to create an exported file from FileMaker Pro or a PDF file. The desktop path is just that: the path to the user's desktop. On both OS X and Windows it is a fully qualified path name, which means that

it starts at the drive (a letter like C: in Windows or a name in OS X). You can use the path as a prefix for a file that you will create on the user's desktop. Alternatively, you can use text functions in FileMaker to drop `Desktop/` from the end of the path so that you can save a file in another location inside the user's home folder.

The new `Get ( TemporaryPath )` function in FileMaker Pro 10 and later gives you access to a folder created by FileMaker Pro (or by FileMaker Server) for each session (FileMaker Pro) or schedule (FileMaker Server). This folder is automatically deleted when the program ends. You can place files in this folder confident that they will not clutter up the user's hard disk once FileMaker Pro has terminated.

```
Get ( DesktopPath )
Get ( TemporaryPath )
```

To see the list of `Get()` functions in the Specify Calculation dialog, you have to toggle the view to either All Functions by Type or to just the Get functions. They don't show up when the view is All Functions by Name. Be aware that there are a number of functions with `Get` in their name that aren't `Get()` functions. They include functions such as `GetRepetition()`, `GetField()`, `GetAsText()`, and `GetSummary()`. These are not functionally related in any way to the `Get()` functions that have just been discussed.

Design Functions

The Design functions are used to get information about the structure of a database file itself. With just two exceptions—specifically, `DatabaseNames()` and `WindowNames()`—not one of the Design functions is session dependent. That is, the results returned by these functions won't differ at all based on who is logged in or what that user is doing. Unlike the `Get()` functions, Design functions often take parameters.

Fully half of the Design functions simply return lists of names or IDs of the major structural components of a file. These include the following:

```
FieldIDs ( fileName; layoutName )
FieldNames ( fileName; layout/tableName )
LayoutIDs ( fileName )
LayoutNames ( fileName )
ScriptIDs ( fileName )
ScriptNames ( fileName )
TableIDs ( fileName )
TableNames ( fileName )
ValueListIDs ( fileName )
ValueListNames ( fileName )
```

Six other Design functions return information about a specified field:

```
FieldBounds ( fileName ; layoutName ; fieldName )
FieldComment ( fileName ; fieldName )
FieldRepetitions ( fileName ; layoutName ; fieldName )
FieldStyle ( fileName ; layoutName ; fieldName )
```

```
FieldType ( fileName ; fieldName )
GetNextSerialValue ( fileName ; fieldName )
```

The `DatabaseNames()` function returns a list of the databases that the current user has open. The list doesn't include file extensions, and it doesn't distinguish between files that are open as a host versus those that are open as a guest.

Similarly, the `WindowNames()` function returns a list of the window names that the current user has open. The list is ordered by the stacking order of the windows; it includes both visible and hidden windows across all the open database files.

Typically, the `DatabaseNames()` and `WindowsNames()` functions are used to check whether a user has a certain database file or window open already. For instance, if you have a navigation window that you always want to be open, you can have a subscript check for its presence and open it if the user closed it. To do this, you use the formula `PatternCount ( WindowNames; "Nav Window" )`. This formula returns a `0` if there was no open window whose name included the string `"Nav Window"`.

The final Design function is `ValueListItems ( fileName ; valueList )`. This function returns a list of the items in the specified value list. As with most of the Design functions, the primary purpose of this function is to help you catalog or investigate the structure of a file. There's another common usage of `ValueListItems()` that is handy to know. Imagine that you have a one-to-many relationship between a table called Salespeople and a table called Contacts, which contains demographic information about all of a salesperson's contacts. For whatever reason, you might want to assemble a list of all the cities where a salesperson has contacts. You can do this by defining a value list based on the relationship that shows the `City` field and then creating an unstored calculation field in Contacts with the formula `ValueListItems ("Contacts"; "CityList")`. For any given salesperson record, this field will contain the "sum" of all the cities where the salesperson has contacts.

Device Identification Functions

FileMaker's built-in security mechanism lets you determine and control who is using your database. The device identification functions let you determine what device they are using. These functions return either

- **UUID**—This is the Universally Unique Identifier of the device. It is a text string containing letters and numbers separated by hyphens.
- **PersistentID**—This is an MD5 hash value of the UUID.

The UUID is the actual identifier value. If you display it or store it in a database, you have identified the user's device, and that may be an invasion of privacy. The MD5 has the value of the PersistentID and is a one-way encryption of the UUID.

This means that you can store the hash value in your database safely. At another time, you can check to see whether the PersistentID of the current device matches that of the previous device. If so, the same device is being used. However, because these values are hashed, you do not directly

check whether the device is the same, which could reveal the UUID. You only check to see if the current and former PersistentID values are the same.

Mobile Functions

Scripts that can be run on mobile devices can interrogate the location of the device.

You Can't Control How the Location Is Obtained

The location of a mobile device is determined using a built-in GPS unit or by triangulating based on cell towers or Wi-Fi locations. You cannot determine which technique is used.

`Location` returns the latitude and longitude of the device. It takes a single required parameter, `accuracy`. This is the accuracy of the distance in meters of the returned result. You can optionally provide `timeout`, which is the number of seconds after which to stop the query. The latitude and longitude are returned as two comma-separated values. If the timeout or accuracy prevent the location from being returned, you get and empty string.

This function is only available on mobile devices; on FileMaker Pro, it is a no-op (that is, not an error, just no operation).

For more precise location information, you can use `LocationValues`, which takes the same parameters. It returns six values you can parse using `GetValue`:

- Latitude in decimal
- Longitude in decimal
- Altitude in decimal
- Horizontal accuracy in integer meters
- Vertical accuracy in integer meters
- Age of value in decimal minutes

Again, you might get an empty string returned if the accuracy or timeout prevent completion of the request.

Container and Window Functions

Specialized functions let you interrogate containers and window styles.

The container functions refer to the contents of the container:

- `GetWidth`
- `GetHeight`
- `GetThumbnail`
- `VerifyContainer`

For windows, you can interrogate the window style with `GetWindowStyle`.

Troubleshooting

Formulas in Scripts Require Explicit Table Context

I'm used to being able to type field names into calculation formulas rather than selecting them from the field list. Sometimes, even if I've typed the field name correctly, I get a `Field not found` *message when trying to leave the calculation dialog. It seems that sometimes calculations need the table occurrence name before the field name, and sometimes they don't. What are the rules for this?*

When you define calculation fields, any fields within the current table can be entered into the formula without the table context being defined. For instance, you might have a `FullName` field defined to be `FirstName & " " & LastName`.

All formulas you write anywhere within ScriptMaker require that the table context be explicitly defined for every field, even when there's only a single table in the file. For instance, if you wanted to use a `Set Field` script step to place a contact's full name into a field, you wouldn't be able to use the preceding formula as written. Instead, it would need to be something like `Contact::FirstName & " " & Contact::LastName`.

If you're used to being able to manually type field names into formulas, be aware that the table context must be included for every field referenced in the formula. The reason for this is that the table context for a script is determined by the active layout when the script is executed. `Contact::FirstName` might have a very different meaning when evaluated on a layout tied to the Contact table than it would, say, on one tied to an Invoice table.

Errors Due to Improper Data Type Selection

I've heard that the data type selection for calculation fields is important. What kind of problems will I have if I select the wrong data type, and how do I know what type to choose?

Every time you define a calculation field, no matter how simple, be sure to check the data type that the formula is defined to return. The default data type is Number unless you're defining multiple calculations in a row, in which case the default for subsequent fields will be the data type defined for the previous calculation.

A number of errors can result from selecting the improper data type. For instance, if your formula returns a text string but you leave the return data type as Number, any finds or sorts you perform using that field will not return the expected results.

Be especially aware that formulas that return dates, times, and timestamps are defined to have date, time, and timestamp results. If you leave the data type as Number, your field displays the internal serial number that represents that date and/or time. For instance, the formula `Date ( 4 ; 26 ; 2013 )` returns `734984` if the date type is set to number.

FileMaker Extra: Tips for Becoming a Calculation Master

As mentioned at the outset of this chapter, mastering the use of calculation formulas takes time. We thought it would be helpful to compile a list of tips to help you get started on the path:

- **Begin with a core**—Don't try to memorize everything at once; chances are you'll end up frustrated. Instead, concentrate on building a small core of functions that you know inside and out and can use without having to look up the syntax or copy from examples. Then gradually expand the core over time. As you have a need to use a new function, spend a few minutes reading about it or testing how it behaves in various conditions.
- **Work it out on paper first**—Before writing a complex formula, work through the logic with pencil and paper. This way, you can separate the logic from the syntax. You'll also know what to test against and what to expect as output.
- **Search for alternative methods of doing the same thing**—It's uncommon to have only one way to approach a problem or only one formula that will suit a given need. As you write a formula, ask yourself how else you might be able to approach the problem, and what the pros and cons of each method would be. Try to avoid the "if your only tool is a hammer, all your problems look like nails" situation. For instance, if you always use `If()` statements for conditional tests, be adventurous and see whether you could use a `Case()` statement instead.
- **Strive for simplicity, elegance, and extensibility**—As you expand your skills, you'll find that it becomes easy to come up with multiple approaches to a given problem. So how do you choose which to use? We suggest that simplicity, elegance, and extensibility are the criteria to judge by. All other things being equal, choose the formula that uses the fewest functions, has tightly reasoned logic, or can be extended to handle other scenarios or future needs most easily. This doesn't mean that the shortest formula is the best. The opposite of simplicity and elegance is what's often referred to as the *brute-force* approach. There are certainly situations in which that's the best approach, and you shouldn't hesitate to use such an approach when necessary. But if you want to become a calculation master, you'll need to have the ability to go beyond brute-force approaches as well.
- **Use comments and spacing**—Part of what makes a formula elegant is that it's written in a way that's logical and transparent to other developers. There might come a time when someone else needs to take over development of one of your projects, or when you'll need to review a complex formula that you wrote years before. By commenting your formulas and adding whitespace within your formulas, you make it easier to expand on and troubleshoot problems in the future.
- **Be inquisitive and know where to get the answer**—As you write formulas, take time to digress and test hunches and learn new things. Whip up little sample files to see how something behaves in various conditions. Also, know what resources are available to you to get more information when you get stuck or need help. The Help system and online discussion groups are all examples of resources you should take advantage of.

- **Use your keyboard**—Entering a less-than character followed by a greater-than character (<>) equates to the "not equal to" operator (≠) within an expression. The following expressions are functionally identical:

```
1 <> 2
1 ≠ 2
```

  This is true for >= and <= for > and <, respectively.

- **Use tabs to improve clarity**—To enter a tab character into an expression (either as literal text or simply to help with formatting), use (Option-Tab) [Ctrl+Tab].

- **Learn the exceptions**—FileMaker allows for a shorthand approach to entering conditional Boolean tests for non-null, nonzero field contents. The following two expressions are functionally identical:

```
Case ( fieldOne; "true"; "false" )
Case ( (IsEmpty (text) or text = 0); "false"; "true" )
```

  Note that the authors do not recommend this shortcut as a best practice. We tend to believe you should write explicit (and, yes, more verbose) code, leaving no room for ambiguity, but if you ever inherit a system from another developer who has used this approach, you'll need to be able to grasp it.

- **Use defaults with conditionals**—FileMaker allows for optional negative or default values in both the `Case()` and `If()` conditional functions. The following expressions are both syntactically valid:

```
Case (
     fieldOne = 1; "one";
     fieldOne = 2; "two"
)

Case (
     fieldOne = 1; "one";
     fieldOne = 2; "two";
     "default"
)
```

  We strongly recommend you always provide a default condition at the end of your `Case` statements, even if that condition should "never" occur. The next time your field shows a value of "never happens," you'll be glad you did.

- **Remember that `Case` short-circuiting can simplify logic**—The `Case()` function features a "short-circuiting" functionality whereby it evaluates conditional tests only until it reaches the first true test. In the following example, the third test will never be evaluated, thus improving system performance:

```
Case (
     1 = 2; "one is false";
     1 = 1; "one is true";
     2 = 2; "two is true"
)
```

- **Repeating value syntax**—Note that fields with repeating values can be accessed either using the GetRepetition() function or via a shorthand of placing an integer value between two brackets. The following are functionally identical:

```
Quantity[2]

GetRepetition ( Quantity; 2 )
```

- **Common calculations**—Create custom functions for common calculations, particularly those that are specific to your solution or to a set of solutions. For example, you can create a custom function that takes a date and returns the fiscal year or the quarter in which it occurs. Depending on the chances of your company changing its accounting year, you might choose to incorporate the start date of the fiscal year into the custom function itself or to place it in a more changeable location such as a custom-built table for such settings.

9

GETTING STARTED WITH SCRIPTING

Scripts in FileMaker Pro

Scripts are sets of stored instructions that specify a series of actions FileMaker should perform when they're initiated; they're programs that run within FileMaker Pro or FileMaker Go solutions. They can be just one command in length, or they can be hundreds of commands long.

Scripts do two important things in FileMaker Pro and FileMaker Go: They automate internal processes, and they add interactivity to custom user interfaces. Internal processes might consist of creating a batch of monthly invoices, setting the status of sales leads, or exporting data for an aggregated report, for example. By *adding interactivity*, we refer to the capability to create interface elements (such as buttons or icons) that will do something in response to user actions. (This is particularly important in FileMaker Go on mobile devices where there is not a menu bar.)

Scripts can also run in response to *triggers*—actions (usually on files, layouts, or layout objects) that automatically cause the scripts to run. It is one thing to click a button that causes something to run; it is a far different thing to enter text in a field and—incidentally—have a script run to automatically handle some necessary action. There has long been the ability to specify validation rules and lookup routines that run when data is entered, but now that any script can be triggered, you can specify any action at all.

Scripts are written using the Manage Scripts command from the Scripts menu in FileMaker Pro or FileMaker Pro Advanced. It provides a point-and-click interface that minimizes syntax errors. Scripts can perform tasks ranging from simple things (such as simply entering Find mode) to com-

plex automated import/export processes, multitable reporting, data reconciliation, and anything that can be expressed as a programmed series of FileMaker steps.

A script usually runs in sequence from its first step to the last, exiting or ending after it is complete. Here's a simple example:

```
Show All Records
Go to Record/Request/Page [ First ]
Beep
Show Custom Dialog [ Title: "First Record";
➥Message: "This is your first record."; Buttons: "OK" ]
```

note

Although you can only create scripts on FileMaker Pro (or FileMaker Pro Advanced), they can run on FileMaker Go as well as in web publishing solutions, where they can be important enhancements to the interface. Not all script steps are available, however. See "Script Editing," later in this chapter, for a discussion of compatibility.

As you can see from this short example, FileMaker Pro scripts are easy to read and comprehend. This script resets the found set of the current layout/window to consist of all the records in a given table and then takes the user to the first record in that set, beeps, and shows a dialog with an OK button. Each step of the script executes in order: `Show All Records` is completed and then `Go to Record/Request/Page` is dealt with.

It's possible to create branching scripts by using logical `If` statements, and it's also possible to construct scripts that execute other scripts (hereafter referred to as *subscripts*). We get into both such techniques later in the chapter.

Script writing is one of the areas where FileMaker Pro Advanced differs from FileMaker Pro. Some of these features are the script debugger, data viewer, the Database Design Report, tools to optimize databases and to create standalone solutions, as well as the ability to copy and paste scripts. Note that it is the *writing* of scripts that is expanded; scripts written using the authoring features in FileMaker Pro Advanced run perfectly well in FileMaker Pro.

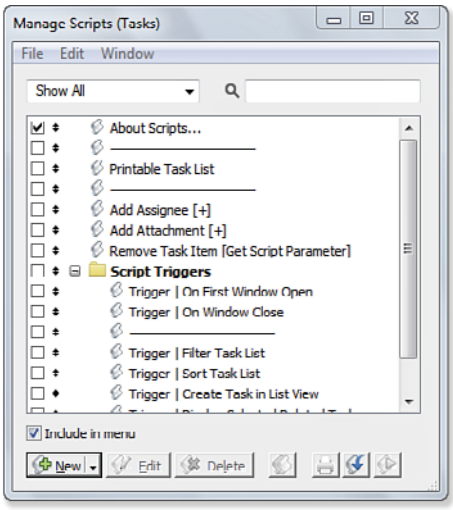
For more information on FileMaker Pro Advanced, **see** *Chapter 16, "Advanced Scripting Techniques." That chapter also provides information about the more programmer-like features of FileMaker scripting such as parameters, variables, and the like.*

Creating Scripts

Creating and editing scripts in FileMaker is straightforward. Simply choose File, Manage, Scripts (or Scripts, Manage Scripts) and the Manage Scripts window opens, as shown in Figure 9.1. You can also use the keyboard shortcut of (Command-S) [Ctrl+S]. Keep in mind you'll need to have signed in with an account that allows script access (the default Admin account for databases allows this access).

When you open the Manage Scripts window, you'll see a list of existing scripts and can manage all the scripts in your file (you can delete, reorder, and so on). You can use the search box at the upper right of the dialog to search for script titles.

Figure 9.1
The Manage Scripts window enables you to create, edit, and organize your scripts, and decide which ones to display in FileMaker's Script menu.



Writing an actual script requires first that you have a goal in mind: What purpose is the script intended to accomplish? A script steps through a series of instructions, one at a time, until the script either reaches its last instruction or reaches some exit condition. Exit conditions can vary, and this chapter covers many of their implementations.

One common use for scripts is to arrange windows, global values, and other settings when FileMaker opens its first file—as well as when it ends. This means that users opening a database will be presented with a known condition of the database, windows, and data; if a companion script runs when the file is closed, everything will be set to an appropriate condition for the next restart.

These are not inconsistent goals: For example, the closing script might store the specific state of the current record and so forth, and, if that value is not found when it is opened the next time, the database will be opened to a known condition.

One particularly important setting that a startup script can manage is a setting for a global variable when you are running in a shared environment on FileMaker Server. FileMaker Server carefully manages separate copies of global fields for each user, and that is true even from one session to another. If user A changes a global value, because it is global only to user A, user B will not see user A's value, which, in most cases, is correct behavior. However, if the global value is a setting such as the current semester for a school, you want it to change for all users at the same time. In that case, you can set the global variable (or a global field) in a startup script. You modify the script once, and then, each time a user runs the startup script that same value is available in a global field or variable.

tip

Note that during the period between the change of the startup script and the time users open the database again, the previous value may be used. This type of change to a script is typically implemented over a weekend or other downtime.

Before FileMaker 12, you could set both a startup script and a layout. Beginning with FileMaker 12, the startup layout (if you want to set it) is accessed via File, File Options. The resultant dialog is shown in Figure 9.2.

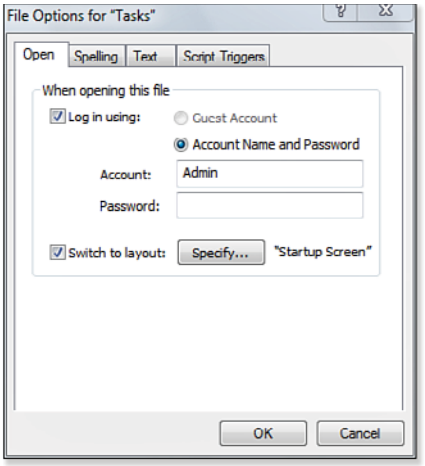


Figure 9.2
You can set a layout to appear automatically when a file opens.

To automatically run a script when a file opens, use the Script Triggers tab in File Options, as shown in Figure 9.3.

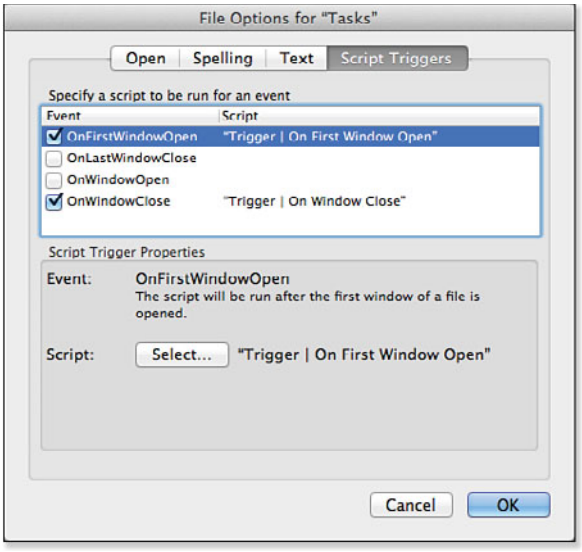


Figure 9.3
Set a script trigger to run a script at file open.

Write the Script First

This example is from the Tasks Starter Solution; you can open it to see how it is set up. Note that if you are building this type of structure yourself, you must first write the script you want to run and then select it with a script trigger, as shown in Figure 9.3.

Scripts to run when a file is opened are becoming more and more important with the advent of FileMaker Go and mobile devices. One of the purposes of such a script is to determine what device is being used. Based on that information, a layout for the appropriate device is displayed. Once that layout is displayed, its navigation tools generally move to other layouts appropriate for that device, so that first layout that is automatically selected in the startup script is critical.

Figure 9.4 shows the script that runs with the On First Window Open trigger event.

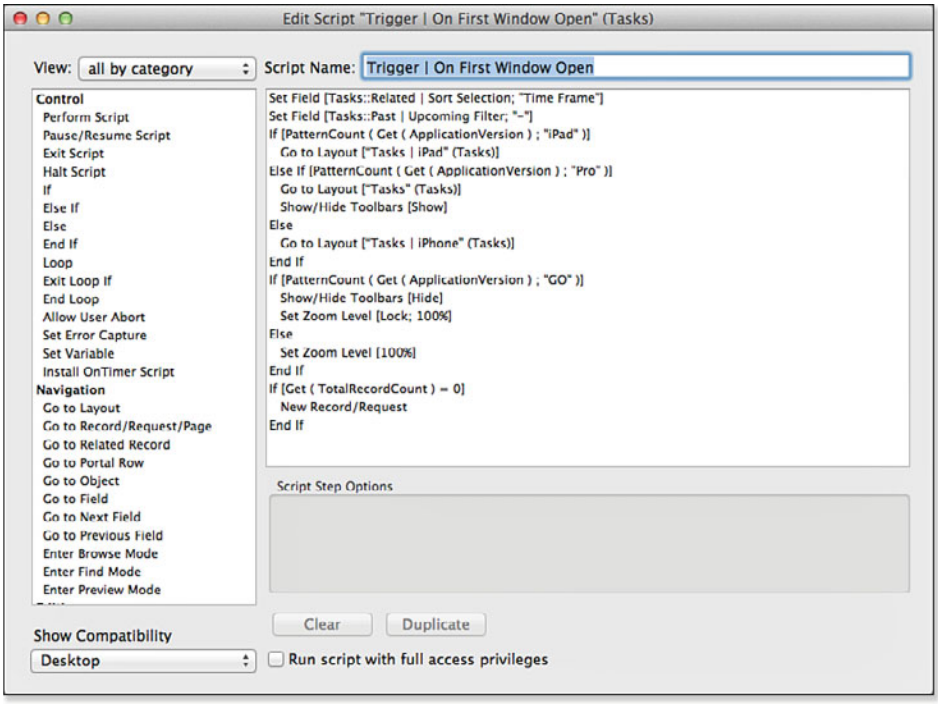


Figure 9.4
A startup script can display a layout for the current device.

Similarly, in Figure 9.5 you see the script that runs when a window is closed.

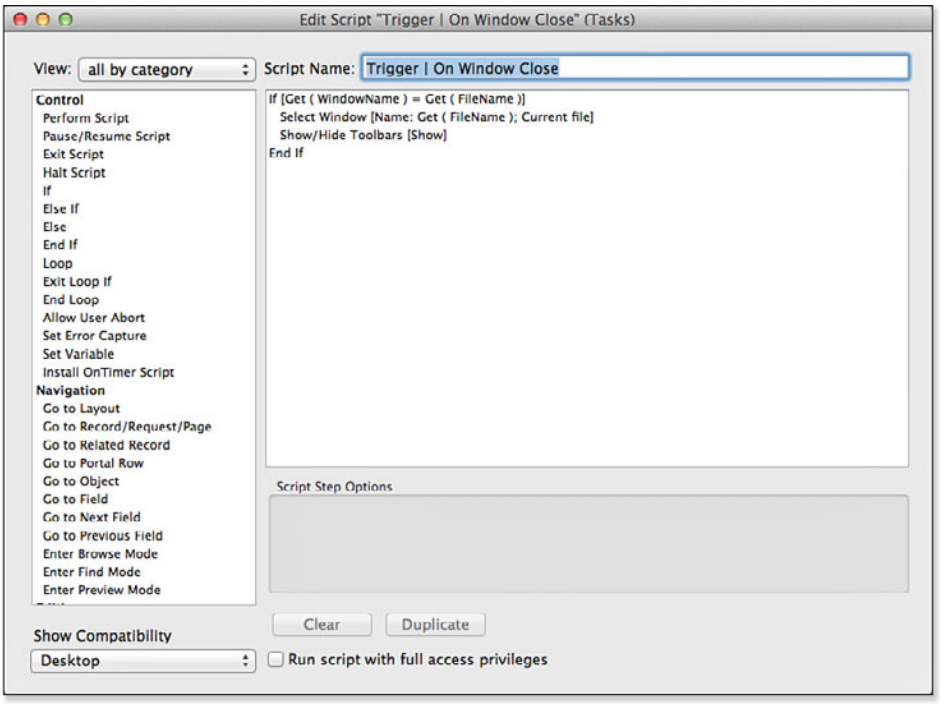


Figure 9.5
A script can be set to run when a window is closed.

The names of the scripts shown in Figure 9.3 include the | character. It is a regular text character and not a part of FileMaker Pro syntax that has any special meaning.

For some guidance on using characters such as these and other naming conventions for scripts, **see** *"Naming Scripts,"* **p. 278**.

Many of the syntax elements of these scripts are described later in this chapter, as well as in Chapter 16, "Advanced Scripting Techniques."

The Scripting Interface

The Manage Scripts window shown previously in Figure 9.1 allows you to manage all the scripts in your current file. As you saw in Figure 9.1, the Manage Scripts window lets you organize scripts into folders and divide them with separators.

As you move separators or scripts into folders, they automatically indent. You can create folders within folders to provide further organization of the scripts. To reorder scripts, folders, or separators, simply drag the individual items up or down the list.

tip
Use (Command-up/down arrow) [Ctrl+up/down arrow] to move scripts via your keyboard.

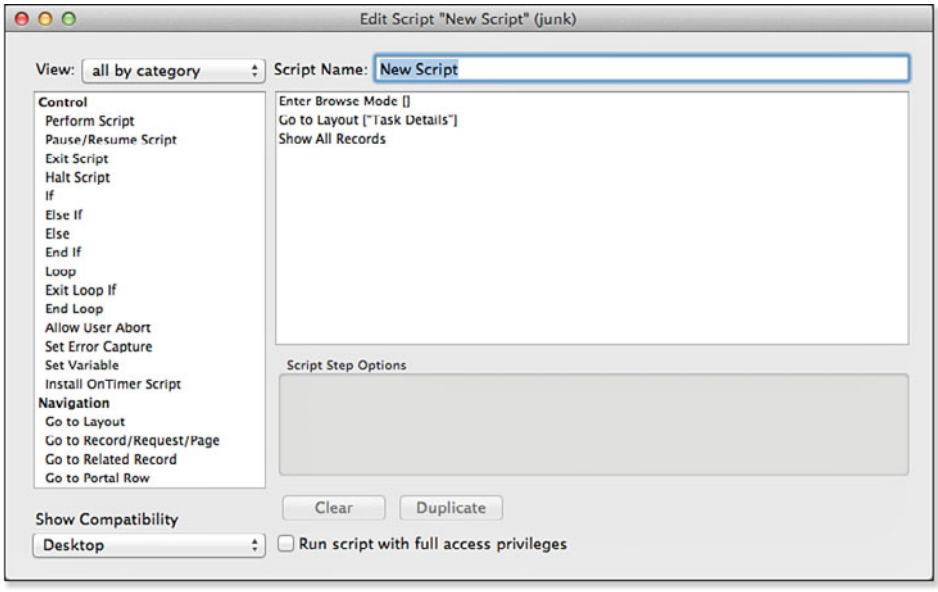
You can use the pop-up menu at the top of the dialog to select whether you want to see all scripts or only the scripts in a given folder.

Buttons at the bottom of the Manage Scripts window let you perform a number of tasks. From the left to right, as shown previously in Figure 9.1, these buttons let you perform the following functions:

- You can create new scripts, folders, or separators with the plus button at the left, together with the down-pointing arrow to its right.

  You also have an option to create a default script, as shown in Figure 9.6. The default script enters Browse mode, goes to the layout you currently have open, and then shows all records. You can use it as a starting point for your scripts.

Figure 9.6
Start from a default script.



- If you select a script in the list, you can edit it with the Edit button. If you select a group, the Edit button lets you change its name just as you can change the name of a script.
- If you select a script, Delete lets you delete it. You receive a warning before the deletion occurs in case you change your mind or clicked the button by accident.
- If you select a script, the Duplicate button makes a copy of it with the word *Copy* appended to the name. You might want to make one or more standard scripts and then copy and customize them as needed. As part of editing a script, you can change its name.
- You can use the Print button to print a script. Printing a script is a good way to create documentation and to spot problems more easily.

tip

Notice that by using Command-click [Ctrl+click], you can select multiple, noncontiguous scripts and then delete, duplicate, or print as you need. Shift-click selects multiple contiguous scripts.

- You can import scripts from one file to another. This feature works similarly to copying scripts.
- By selecting a script and clicking Perform, you can execute scripts directly from this dialog.

Script Editing

After you create a new script or edit an existing script, the Edit Script window opens (as you saw previously in Figure 9.6). Here, you construct the actual script by inserting script commands from the list on the left into the window on the right. Nearly every script step has additional options you need to specify, such as the name of a layout to go to, or the name of a file from which to import. These options appear under your script when you highlight a given step in it.

tip

You can (Command-click) [Ctrl+click] multiple script steps at once and insert the batch into a script in one move.

As an example, `Go to Layout` is a common step you'll use quite often. Notice that when you insert it into a script, a menu appears in the Script Step Options area at the lower right, from which you can choose an existing layout, the layout on which the script began, or one determined by calculation.

To reorder script steps, simply drag them by the two-headed arrow icon located to the left of the step.

You can open as many script-editing windows as you want at the same time: Just select the script(s) you want to edit and click Edit in the Manage Scripts window. If you try to close a script window with unsaved changes, you receive a prompt with a dialog to save the changes. In addition, when you are working with scripts, the Scripts menu contains commands to let you save or revert the script in an editing window or to save all scripts.

In the lower left of the Edit Script window, the Show Compatibility pop-up menu lets you see which commands are compatible with the various environments in which a script might run:

- Desktop
- Server
- iOS
- Instant Web Publishing
- Custom Web Publishing

Any script step that is not compatible with the selected environment will be grayed out. You can still use these script steps, but they will have no effect (and, in fact, may cause your script to execute in unexpected ways). Rather than relying on these script steps simply not executing, it is better to actually check the environment in which you are running and skip over or modify such script steps. The startup script shown previously in Figure 9.3 provides good examples of this practice.

Full Access Privileges

Notice the Run Script with Full Access Privileges check box at the bottom of the Edit Script dialog. Designating that a script run with full access privileges simply means that for the duration of that script, FileMaker overrides all security restrictions. When this option is *not* enabled, scripts run subordinate to whatever privilege set the currently signed-in user has. For instance, if a script makes a call to delete a record and the user running that script cannot do so based on his current security privileges, the script usually presents an alert message to the user and ignores that step of the script. The rest of the script is still performed.

Note that when this option is checked, the security privilege set for the current user actually does change for the duration of the script: If you use the calculation function `Get ( PrivilegeSetName )`, it returns `[Full Access]` as long as the script is running. If your script contains logic in which you need to check a user's assigned privilege set, you'll need to capture the user's privilege set information elsewhere before running the script and refer to it however you've stored or captured the information.

➥ *Error management in scripts is an important element in all scripting. For more detail,* **see** *"*`Set Error Capture`*,"* **p. 265.**

➥ *To understand FileMaker security and privilege sets,* **see** *Chapter 12, "Implementing Security."*

Commenting Scripts

Keeping track of what scripts do is a difficult task. What seemed perfectly intuitive at the time you wrote a given script might become hopelessly obscure a few weeks—or sometimes even hours—later. Although developers vary in how they use comments, nearly all developers recognize the value of commenting their work.

In addition to describing the purpose of the script, it is particularly helpful to note if the script changes from one layout to another. Entry assumptions (such as the layout from which it is called) and exit assumptions (such as the layout that it leaves open) are important to people who are calling scripts. In this case, as well as many others, documenting the script in a standard way can force you to consider important issues such as these.

Listing 9.1 shows a simple example of a commented script. Notice that the # symbol prefixes comments in FileMaker Pro.

note

Remember that you're not coding in a vacuum. We can virtually guarantee that although you might never intend that a given database be seen by someone else's eyes, if it stands the tests of time and proves useful, at some point you'll crack it open with the infamous words, "Let me show you how I did this...." Likewise, professional-grade systems are nearly all collaborative efforts. Comments exist to help your peers understand what your caffeine-sodden brain was thinking at the time you wrote a particular routine.

Listing 9.1 Script with Comments

```
#   Purpose: initiate the running of a report while allowing users
#     to choose what sort order they want
#   History: sl 2007 02 04; bb 2008 02 05
#               jf 2012 04 2
#   Dependencies: Invoices: Monthly Report layout
#   Entry assumptions: none
#   Exit assumptions: layout Monthly Report unless cancelled
#
#          prompt user for sort order
Show Custom Dialog [ Title: "Sort Order"; Message: "Do you want to sort by
➥amount or date?"; Buttons: "Date", "Amount", "Cancel" ]
#
#          check for cancel first
If [ Get (LastMessageChoice) = 3 ]
Go to Layout [ original layout ]
Exit Script
#
#          sort by Amount
Else If [ Get (LastMessageChoice) = 2 ]
Go to Layout [ "Monthly Report" ]
Perform Script [ "Sort by Amount" ]
#
#          sort by Date
Else If [ Get (LastMessageChoice) = 1 ]
Go to Layout [ "Monthly Report" ]
Perform Script [ "Sort by Date" ]
 #End If
```

Exiting a Script

You can exit a script with an explicit call to `Exit Script`; you can also exit the script just by executing the last line of code. If you are passing back a result value from the script, you must use the call. Also, if you are leaving the script at any location other than the last line (perhaps as a result of an error you have encountered), you must use `Exit Script`.

The `Exit Script` step is a script step like any other. That means that if you are stepping through a script with the debugger, you will pause just before the script step is executed—that is, just before you exit the script. During that pause, you can inspect the values of the variables in the script.

Because `Exit Script` is required in some cases and is helpful in debugging, it is a good idea to always use it in writing your scripts.

For more information on script results, **see** *Chapter 16, "Advanced Scripting Techniques."*

For more details on script debugging, **see** *Chapter 19, "Debugging and Troubleshooting."*

Using a Script Template

It is often helpful to create a template script that you can duplicate when you need to create a new script. In our templates, we include several comment lines at the top where we record information about the purpose and revision history of the script. A template script looks something like this:

```
# purpose:
# dependencies:
# history:
# entry assumptions:
# exit assumptions:
# parameters in:
# result value out:
#
#   set error handling
Allow User Abort [ Off ]
Set Error Capture [ On ]
#
#   establish context
Go to Layout [ Original Layout ]
#
#
```

Although it is simple, this template does save time and promote good code. If you don't need a particular piece of it, you can delete it easily enough.

Parameters and result values are discussed in Chapter 16, "Advanced Scripting Techniques."

tip

Adding the `Go to Layout` step to your template can help ensure that the script begins on the correct layout and thus is associated with the proper base table attached to that layout. Including this step in the template prompts developers to make a conscious decision and reminds you that context needs management.

Using Subscripts

One of the most useful features in FileMaker Pro scripting is the `Perform Script` step itself. One FileMaker script can call another script, which is then commonly known as a *subscript*. This allows you to divide scripts into smaller logical blocks and to break out discrete scripts for anything you are likely to want to use again. This degree of abstraction in your system is something we very much recommend. Abstraction makes scripts easier to read, easier to debug, and modular in that a subscript can be generic and used in a variety of scripts. Here's an example:

```
Sales_Report
# purpose: to run the Sales Report, weekly or monthly
# history: scl 2-5-2009
#
Perform Script [ "CheckPermission_forSales" ]
Perform Script [ "Find_CurrentSales" ]
#
Show Custom Dialog [ Title: "Run Report"; Message: "Would you like this
➥report broken out by Weekly or Monthly subtotals?"; Buttons:
```

```
"Monthly", "Weekly", "Cancel" ]
#
If [ Get (LastMessageChoice) = 1 ]
  Perform Script [ "Monthly_Report" ]
#
Else If [ Get (LastMessageChoice) = 2 ]
  Perform Script [ "Weekly_Report" ]
#
End If
```

Notice that the script actually doesn't do much on its own. It first runs a permission check script and then runs another script to establish a found set. It then prompts the user to make a choice and runs one of two report subscripts based on what choice the user makes. This approach is quite common and demonstrates a flexible approach to programming. The `Find_CurrentSales` subscript could have other uses elsewhere in the database. Creating separate routines for weekly and monthly reports makes the script more readable. Imagine seeing all the logic for those two reports embedded here as well.

As another example of script abstraction, imagine sorting a contacts database by `last_name` and then by `first_name` for a given report. If you write a script to produce that report, sorting is a step in the process. However, odds are that you'll want to be able to sort by `last_name`, `first_name` again—perhaps for a different report, perhaps as a function that lives on a List view or in a menu, or perhaps before running an export script (or perhaps all the above). Whenever reasonable, we recommend looking for ways to abstract your code and foster reuse. Doing so saves time and complexity if your client (or boss) suddenly comes to you and says now you have to present everything by first name. If that logic lives in one place, it's a one-minute change. If you have to hunt for it, the change could take days and require extensive debugging.

tip

Even if you're not planning to reuse blocks of code, it's still a good idea to break scripts into subscripts. They're easier to read, they're easier to enable and disable during testing, and they allow you to name them in logical ways that are comprehensible even at the Define Scripts dialog level.

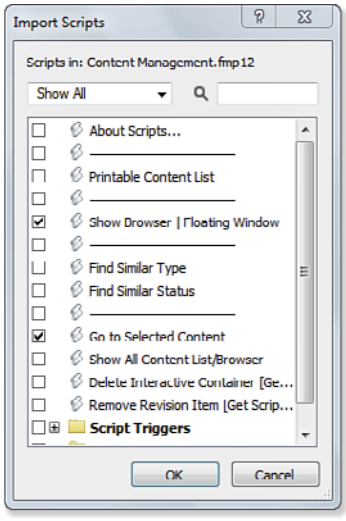
Some other good candidates for subscripts are sort and find routines; these are often reusable by a wide range of scripts or by users as standalone functions. Other uses of subscripts might be for the contents of a loop or `If` function. Sometimes it's easier to separate logic into separate paths by dividing logical groups into separate scripts, as in the example we gave a little earlier. When you have a branching script (covered later in the chapter), it's helpful to encapsulate a single branch in a subscript. This allows you to see the flow of logic in the parent script and cover each branch in its own respective subscript.

Finally, consider separating scripts by whether or not they have user interactions. Often a script interacts with the user, collecting options and values it passes on to a subscript that does not interact with the user.

Importing Scripts

You can import scripts from another database file by clicking the Import button at the bottom of the Manage Scripts window. This lets you select a file; after you have done so, the Import Scripts dialog shown in Figure 9.7 opens.

Figure 9.7
You can import scripts from another file.



Use the check boxes to select as many scripts as you want to import. After the import completes, you receive a warning about any conflicts or errors encountered.

Managing the Scripts Menu

The Scripts menu in Browse mode shows the available scripts organized into the groups you have created. The check box column to the left of a script, folder, or separator controls whether it appears in FileMaker's standard Scripts menu. If you hide a script by unchecking its check box, you need to provide the user with another means of performing, or executing, the script. Typically, this entails either associating the script with one or more button objects that appear on various layouts or tying the script to a custom menu item. And, of course, subscripts often should not appear in the Scripts menu because only other scripts call them.

Figure 9.8 shows the Scripts menu with the groups and scripts visible, as indicated previously in Figure 9.1. If a script is visible but its group is not, it appears on its own in the menu as if it were not part of a group. In addition, even if a group is supposed to be shown, it will be grayed out if there are no visible scripts within it.

note

In addition to controlling whether the Scripts menu shows scripts, note that FileMaker Pro Advanced allows developers to create custom menu sets. Quite often you might want to have a menu item run a certain script. You might want to use custom menus to hide the regular Scripts menu altogether and attach your scripts to other menu items throughout other menus.

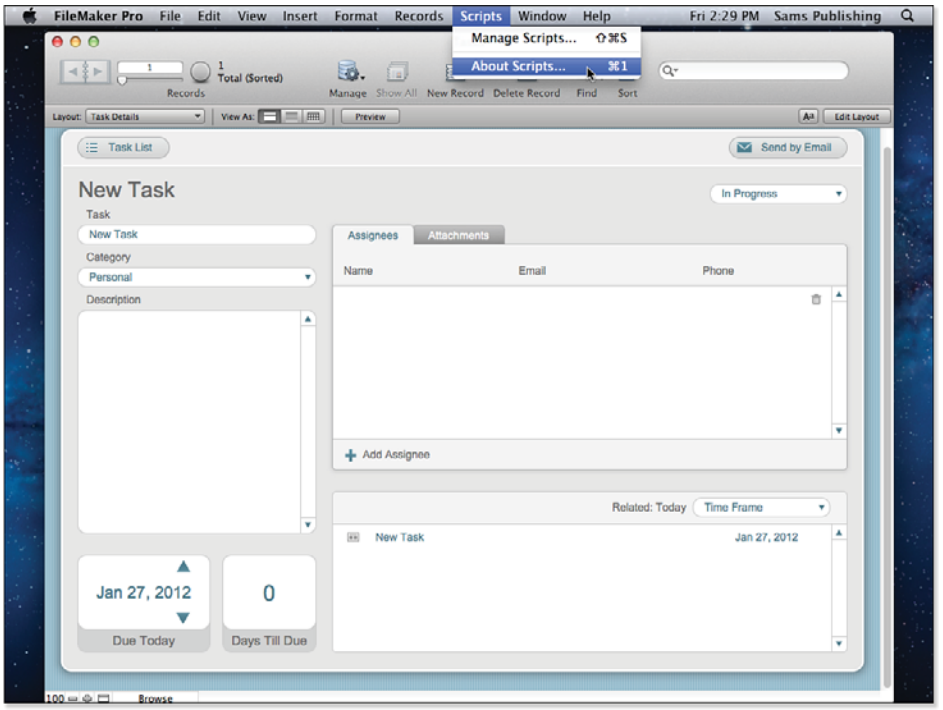


Figure 9.8
You can control visibility of the Scripts menu items.

To learn how to implement custom menus in a solution, **see** *Chapter 14, "Advanced Interface Techniques."*

Common Scripting Topics

We will now delve into some useful and common scripting techniques and discuss topics that are germane to a wide range of scripts. They will help you establish a solid foundation in scripting.

Error Management

Error management is an important part of the scripting process. Frequently scripts make assumptions about the presence of certain data or the existence of certain objects, or depend on a layout to establish context. If any one of a given script's assumptions is not met, the script either might not work or might produce unintended results. Error management involves identifying these assumptions and creating ways of dealing with them. You can bank on users finding odd, unpredictable ways to break your system. Applying some thought to how to manage such situations will serve you well in the long run.

Note that FileMaker Pro Advanced has the capability to enable and disable individual script steps. This facilitates testing significantly: You can turn off sections of your script that aren't finished and run discrete sections of your logic.

For further discussion of error handling, **see** *"Handling Errors in Scripts,"* **p. 490.**

For more ideas on error management, **see** *Chapter 19, "Debugging and Troubleshooting."*

Allow User Abort

`Allow User Abort` enables and disables a user's ability to press (Command-period) [Esc] to cancel a script in midstream. Generally speaking, it's the rare script that's designed for graceful cancellation at any time in its process. There's really no reason to turn on `Allow User Abort` unless you're testing a loop script or some other long-running process. Any script that doesn't have `Allow User Abort` disabled allows users to cancel a script in progress, with consequences you might not intend. Note that this is true for scripts users run, but the opposite is true for developers: If you're in the midst of writing a script and need to test a loop, for example, you should leave this setting turned on to halt your script if necessary.

The other thing `Allow User Abort` does is take away the Cancel button when a script pauses, giving users only the option to continue. There are many cases in which canceling a script would leave the user stuck on a report layout or stranded midstream in some extended process.

*To learn more about how to deal with incomplete script completion (*atomicity *in database lingo),* **see** *"Unfinished Scripts" in the "Troubleshooting" section at the end of this chapter.*

Set Error Capture

The `Set Error Capture` script step either prevents or allows the display of FileMaker's default error messages to the user. When error capturing is off, FileMaker displays its own alert dialogs to the user if, for example, a record fails validation or a user runs a search without any find criteria. When error capture is on, the script in question captures errors and doesn't present them to the user. This allows you, the developer, to present your own, customized error messages but imposes a greater burden in terms of checking for and managing errors yourself.

Handling errors well in scripts is a black art because it's difficult to anticipate what errors will crop up. For more information on using the `Set Error Capture` *script step,* **see** *Chapter 19, "Debugging and Troubleshooting."*

When doing your own error checking and managing, you'll want to use the `Get ( LastError )` calculation function to programmatically deal with errors within your script. Use the `If` function to test `Get ( LastError )` and present dialogs to the user, as appropriate. Refer to FileMaker Pro's online help system for a list of error codes.

To explore problems with error messages you think are being wrongly suppressed in scripts, **see** *"Lost Error Messages in Scripts" in the "Troubleshooting" section at the end of this chapter.*

caution

Be careful with the `Set Error Capture` script step. It certainly doesn't prevent errors from happening; it simply doesn't show the user a message about one that did. An error might happen, but FileMaker won't interrupt the user's experience to deal with it. This allows you to control how you manage errors within your script itself. You should not turn on error capture unless you also add steps to identify and handle any errors that arise.

Here's an example of a script segment that tests for an error—in this case a find request that results in zero found records:

```
Find_BirthdaysThisMonth
Enter Find Mode [ ]
Set Field [ Person::birthMonth[Month ( Get (CurrentDate)
➥)] ]
Perform Find []
If [ Get( LastError ) ≠ 0 ]
Show Custom Dialog [ Title: "No Birthdays Found";
       Message: "There are no birthdays listed for this
       ➥month.";
       Buttons: "OK" ]
End If
```

note

This segment reports that no birthdays are found if any error occurs. You might want to refine the test so that if the database is not accessible or some other error occurs, the report is more specific. However, the message "No Birthdays Found" is, indeed, correct no matter what the error is.

Setting and Controlling Data

Some of the primary uses of scripts lie in manipulating, moving, and creating data. The Fields category contains most of the script steps for manipulating field data.

Essentially, these field category steps allow you to insert data into a given field programmatically, just as a user otherwise would. This could mean setting the field contents to the result of a calculation, copying the contents of one field into another, or simply inserting into a field whatever is on the user's Clipboard.

As an example, imagine that you wanted to give users a button that inserts their name, the current date, and the current time into a comments field, and then places the cursor in the proper place for completing their comment:

```
# purpose: To insert user and date/time data into a comment field, preserving
#      the existing information, and place the cursor in the correct position
#      for the user to begin typing.
# dependencies: Need to be on the Main_Info layout, with the Comment field
#      available. The script takes the user there.
# history: sl 2009 jan 25
#
#
Allow User Abort [ Off ]
Set Error Capture [ On ]
#
#
Go to Layout [ "Main_Info" (Movie) ]
#
#    this next step applies the comment info in italics.
Set Field [ Movie::Comment; TextStyleAdd (
Movie::Comment & "¶¶" & Get ( AccountName ) & " " &
Get ( CurrentDate) & " " & Get ( CurrentTime);
Italic)
& "¶" ]
#
```

note

This script includes the full commenting approach described in this chapter and the two `Allow User Abort` and `Set Error Capture` steps. From here on out, we'll forgo those details in the interest of brevity.

```
Go to Field [ Movie::Comment ]
Commit Records/Requests [ No dialog ]
```

When using a `Go to Field` step, FileMaker Pro places the cursor at the end of whatever content already exists in the field unless the Select/Perform option is enabled, in which case FileMaker Pro selects the entire field. If you wanted, you could use the `Set Selection` script step to place the cursor somewhere within the body of text.

Notice that the comment info is nested within a `TextStyleAdd()` function so that it displays in italics.

For more information on calculation functions, including text formatting, **see** *Chapter 8, "Getting Started with Calculations," and Chapter 15, "Advanced Calculation Techniques."*

`Set Field` is by far the most used of the field category steps. Nearly all the other functions in this category depend on the field in question being on the layout that performs the script. You should get into the habit of using the `Set Field` command whenever possible, in preference over the others. It doesn't depend on a field being on a specific layout—or any layout, for that matter—and it can usually accomplish what you're trying to do with one of the other steps.

You'll generally need the `Insert` script steps only when you expect user input. For example, you might place a button next to a field on a given layout called "index" that then calls up the index for a given field and waits until the user selects from its contents. That script could often be a single step: `Insert from Index (table::fieldname)`. As always, you would use your template for clarity, but this script would open the index for a given field and wait for the user to select a value. Again, you should tend to think of scripts as evolutionary. Consider writing a script even for a one-step process because you might want to attach that script to multiple buttons or extend its operation in the future.

To manage cases in which your script seems to be affecting the wrong portal row or related record, **see** *"Editing the Correct Related Records" in the "Troubleshooting" section at the end of this chapter.*

For more discussion on indexes, **see** *"Storage and Indexing,"* **p. 104.**

For further discussion of layout dependencies, as well as other types of dependencies that can get your scripts into trouble, **see** *"Context Dependencies,"* **p. 497.**

Another example of using the `Set Field` script step concerns totaling child record data calculations and saving the results in a new record (presumably to track the growth of some quantity over time). Often a simple calculation field with a `Sum ( related field )` function works, but consider that with a large related data set, the performance of such calculations can become a problem. Furthermore, you cannot index that sort of a calculation field—which might prove problematic for users performing find requests

caution

You might also discover the `Copy`, `Cut`, and `Paste` script steps. They work as you would expect. `Copy` and `Cut` place data onto the user's Clipboard, and `Paste` inserts from it. `Cut` and `Copy` overwrite anything already on the user's Clipboard. Furthermore, `Copy`, `Cut`, and `Paste` depend on having access to the specified fields and are therefore layout dependent. If, for some reason, you remove those fields from the specific layout in the future, your script will stop working. You should almost never use `Copy` and `Paste` for these reasons and should defer instead to `Set Field`.

or for your needs as a developer. Consider instead creating a script to calculate and store your totals and calling that script only on demand:

```
Go to Layout [ "Customer" (Customer) ]
Set Field [ Customer::storedTotal; Sum ( OrderbyCustomer::Amount ) ]
Set Field [ Customer::storedDate; Max ( OrderbyCustomer::Date ) ]
Commit Records/Requests
[ No dialog ]
Go to Layout [ original layout ]
```

The preceding script is a typical example of drawing data from related records, of moving from layout to layout to establish proper context, and finally of using `Set Field` to populate data.

The `Set Field` script step lets you specify the field to be set as well as the value to be used with two Specify buttons in the lower right of the options area. In FileMaker Pro, a new script step was added: `Set Field By Name`. Both of these script steps open a calculation dialog to let you specify the value to be placed in the field. However, the Set Field Specify button for the field opens a list of fields in the database tables, whereas the new `Set Field By Name` script step opens another calculation dialog in which you can construct a field name based on any conditions. This script step allows you much more flexibility in developing scripts, and it is one of the most valuable additions to scripting in FileMaker Pro.

For examples of the use of `Set Field By Name`, **see** *Chapter 14, "Advanced Interface Techniques."*

Providing User Navigation

You might have noticed a section of the script steps list devoted to navigation in FileMaker's Edit Script dialog. One of the most common uses of scripts is to provide a navigation scheme to users whereby they can navigate from layout to layout, record to record, or window to window by using buttons or some other intuitive means.

There's not too much magic here: By using the `Go to Layout` script step, you'll get the fundamentals. Consider placing buttons along the top of each layout to offer a means of navigating to all user-facing layouts in your solution with a `Go to Layout` script attached.

By building complete navigation scripts, you can control the entire user experience of your solutions and can opt to close the Status toolbar if you want. Armed with find routines, sort buttons, reporting scripts, and a navigation interface, you are able to build a complete application with a look and feel all its own.

Script Context and Internal Navigation

Consider that FileMaker uses layouts to determine script context: For any script step that depends on a specific table, you need to use `Go to Layout` steps to provide that context. Review the script we introduced at the beginning of the chapter:

```
Go to Layout [ "zdev_GlobalAdmin" (Globals) ]
Set Field [ Globals::gAccountName; Get (AccountName) ]
```

```
Set Field [ Current_User::LastLoginDate; Get (CurrentDate) ]
Set Field [ Globals::gUserNameDisplay; Current_User::Name_First ]
Set Field [ Globals::gUserMessage; "Welcome Back, "
       & Globals::gUserNameDisplay & "." ]
Go to Layout [ "Main Menu" (Globals) ]
```

This script takes itself to a `zdev_GlobalAdmin` layout, executes some steps (in this case sets data into fields), and then brings the user to a `Main Menu` layout. All the users see (presumably when they log in) is that they've landed on a `Main Menu` layout. They'll never see, or interact with, the `zdev_GlobalAdmin` layout, but the system will have done so. Had we written the script without the initial `Go to Layout` step, the routine would have had quite unexpected results.

Notice that the script makes use of a `Current_User` table occurrence. As related data, that information would likely be very different depending on the perspective from which a user views it. The purpose of navigating internally to a specific layout is to control this context precisely.

The point here is that you'll need to bring a script to a specific layout to establish a different context. The table occurrence associated with a given layout determines context. The user might never see this internal navigation going on, but if you were to walk through the script step by step, you'd see the system go to the `zdev_GlobalAdmin` layout and then to the `Main Menu` layout.

To see an example of how to walk through a script using the Script Debugger, **see** *Chapter 19, "Debugging and Troubleshooting."*

Using object names on layouts provides greater control of navigation. For more information, **see** *Chapter 14, "Advanced Interface Techniques."*

Saved Script Options

Scripts tend to mirror the actions a user could perform manually but, obviously, do so without human intervention. It is possible in FileMaker to save find, sort, export, and other actions in a script (hard-coding them, if you will), or to prompt the user for some input to help perform these steps.

The advantages of hard-coding requests should be obvious. If, for example, you need to prepare a report on active real estate listings, it makes sense to have one of your script steps be a `Perform Find` that returns all the records with a status of "active." The requirements of your report will rarely change, so you'll save users time (and possible errors) if you hard-code the find request.

On the other hand, allowing the user to provide input is a great way to make scripts more flexible. Continuing the example, you might create a real estate listings report and in your script prompt the user for some search criteria. This can be done either by using a dialog that gives the user one or two choices (we cover that later in the chapter, in the "Working with Custom Dialogs" section) or by simply allowing the report to act on the current found set and sort.

You will often find it helpful to build hard-coded find and sort routines. For example, you might want a script for finding overdue invoices, or easy-access buttons for sorting by first name, last name, or company. FileMaker allows you to save complex find, sort, export, and import requests as necessary, and allows you to edit these requests.

Find Script Steps

FileMaker allows you to assemble and store complex requests within scripts using *find requests*. You create and edit them by double-clicking Perform Find in the script editing window. In Figure 9.9, the script finds all records from the Jasmine company and omits those with status Pending; the result replaces the found set.

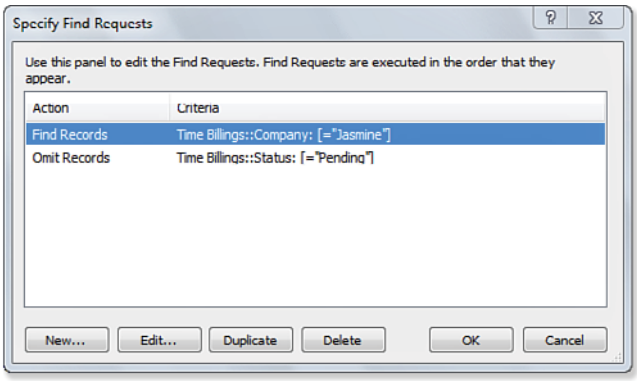


Figure 9.9
Assemble as many find requests as necessary.

You can assemble a single find request via the Edit Find Request dialog; from the Specify Find Requests dialog shown in Figure 9.9, click New or Edit to open the Edit Find Request dialog.

Note in Figure 9.9 that we have opted to omit records that match the second request. You choose this in the Action pop-up menu shown at the top left of Figure 9.10. Setting a request to omit records simply means that FileMaker finds those records that match the overall request and takes out or ignores those that meet the omit criteria. If you create a find request that does nothing but omit records, it replaces your existing found set with all records that don't match your request. The example shown in Figure 9.10 shows a script that combines a find request with an omit request:

Other actions options include `Constrain Found Set` and `Extend Found Set`. Just as though a user had chosen each command from FileMaker's menu-driven interface, `Constrain` reduces the current found set, eliminating any records that don't match the search criteria, and `Extend` adds those records from outside the set that match its criteria to the current found set.

With FileMaker Pro 10 and later, you can now save and manage Find requests from scripts. For more on this, **see** *Chapter 16, "Advanced Scripting Techniques."*

Sort Script Step

Establishing saved sort orders in the Sort dialog works, happily, just as it does for users performing a manual sort (see Figure 9.11).

Figure 9.10
By adding multiple criteria to a single find request, you are performing an And search.

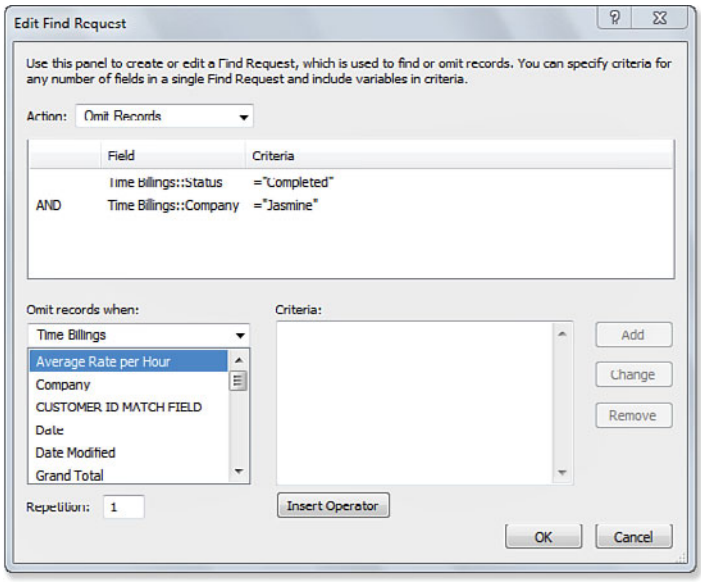
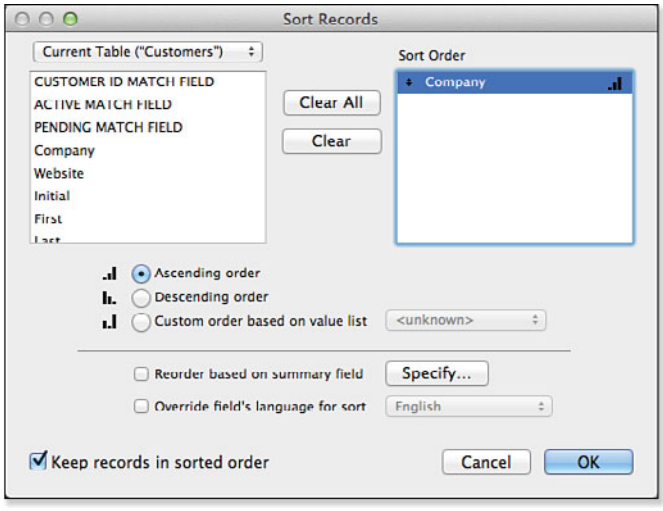


Figure 9.11
Creating sorting scripts for users is generally quite helpful. Sorting needs are usually predictable and always needed more than once.



One of the most common applications of sorting scripts is in building column header buttons. Simply create a series of sorting scripts and apply them to the buttons along the top of a list view. That is what is done in the Time Billing Starter Solution, which is shown in Figure 9.12.

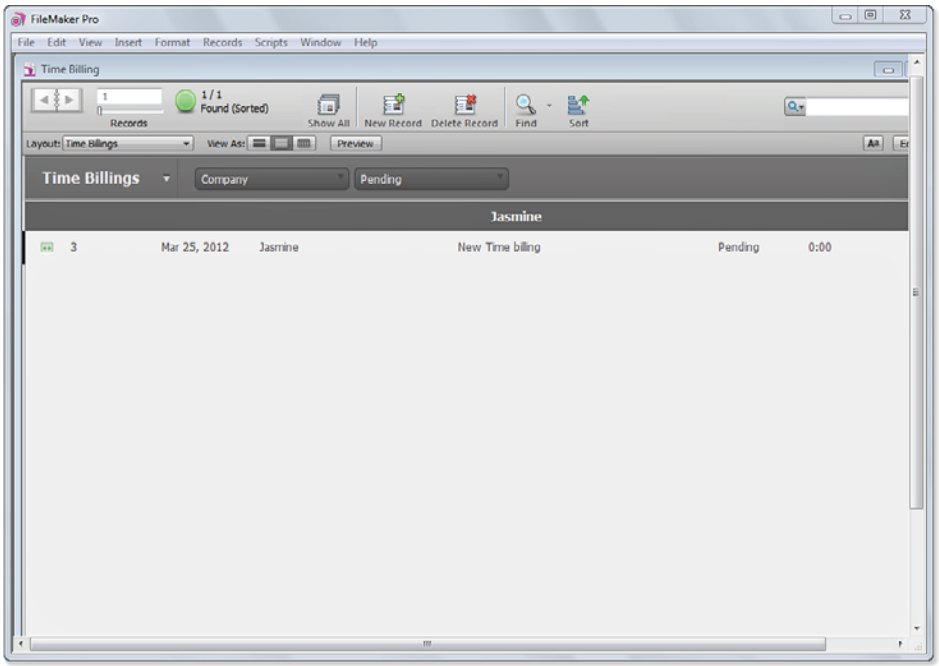


Figure 9.12
Use pop-up menus with triggers to perform multi-level finds and sorts.

For more details on using scripts to sort data, **see** *Chapter 14, "Advanced Interface Techniques."*

Keep in mind that many of your reports depend on sorting, especially as you get into reporting by subsummary data. It's a good idea to create sort scripts for your reports and call them as subscripts, rather than hard-coding sort criteria into your report scripts themselves.

You might well create reports that behave differently depending on different sort orders—by setting up different, multiple subsummary parts on one report layout, for example—so you'll want to factor the sorting logic into its own script or subscripts. A report with both a week-of-year subsummary part and a month subsummary part will display by week, by month, or by week and month, depending on the sort options your script establishes. This technique is handy for reducing the number of layouts you need in a system; with a little bit of scripting, you can use a single layout for three different reports.

For more details on summary reporting, **see** *Chapter 10, "Getting Started with Reporting and Charting."*

Using Conditional Logic

Another important element of scripting is the capability to branch scripts based on various conditions. To manage logically branching scripts, you use the `If`, `Else`, `Else If`, and `End If` script steps. These conditional script steps work by performing a logical test, expressed as a calculation. If

that calculation formula resolves to a true statement, FileMaker executes all the script steps subordinate to (that is, nested within) an `If` or `Else If` statement.

One of the most common applications of conditional logic in FileMaker revolves around `Perform Find` script steps. Because we, as developers, can never guarantee the state of a given table's data—in other words, how many records it contains—we have to test for their existence in scripts that perform find requests and then branch accordingly if no records are found. Here's an example:

```
Find Overdue Orders
Set Error Capture [ On ]
Allow User Abort [ Off ]
#
Go to Layout [ "Order" (Order) ]
#
Perform Find [ Specified Find Requests:
       Find Records; Criteria: Order::Status: ""Overdue"" ][ Restore ]
If [ Get ( LastError ) = 401 ]
Show Custom Dialog [ Title: "Overdue Orders";
       Message: "There are no overdue orders in the system.";
       Buttons: "OK" ]
Show All Records
Else If [ Get ( LastError )≠ 0 ]
Show Custom Dialog [ Title: "Overdue Orders";
       Message: "Unexpected error in finding data [12-3].";
       Buttons: "OK" ]
Show All Records
End If
```

In this simple script, two outcomes are possible: Users see either a set of order records whose statuses have been set to Overdue or a dialog informing them that no overdue orders exist and ending with a full set of all orders.

The entire idea behind conditional logic is to allow the computer to determine which of multiple possible paths to take. Computers aren't terribly smart, so they make these decisions based entirely on Boolean (true/false) tests. At the end of every script step, FileMaker records an internal error that you can retrieve using the `Get ( Last Error )` function. In the preceding script, if that function is storing a value of `401`, the first set of nested steps within the `If` clause will be performed; if the value is not zero (no error) and not `401`, the second set of nested steps will be performed, providing a true error message. If you turn on error capture, it is your job to capture errors; typically, you either capture all of them, or, in cases such as this, you handle some "errors" as normal conditions, passing along other, unexpected errors.

To learn how to bake error checking into your conditional tests, **see** *"Conditional Error Defaults" in the "Troubleshooting" section at the end of this chapter.*

There's no practical limit to the number of branches a script might take. For scripts of particular complexity, we recommend breaking them into subscripts and, when necessary, creating a flowchart of the process before writing these scripts.

Using Loops

Another key scripting technique is looping. Looping allows you to execute a series of script steps repeatedly until some exit condition is met. This is very much the same as an `If/Else If` construct. But this time, instead of performing a new branch of logic, you simply tell the script to perform the same actions over again until a controlling conditional test returns a true value—for example, if the end of a found set is reached or the results of a calculation come to a specific number.

A simple example of this might be stepping through each record in an invoice table's found set and generating a new invoice for any invoice that remains unpaid or should be sent out again for some reason. The logic, without worrying about syntax, might look like this:

```
Go to first record in found set.
Begin Loop.
    Check whether the invoice is closed. (We'll assume
    unclosed invoices are those that need to be resent.)
    If CLOSED
        Go to next record.
        If there is no next record (you're at the end of
        your found set), then exit the loop.
        Else begin loop again (go to the "Begin Loop" step).
    If NOT CLOSED
        Close current invoice.
        Create/duplicate new invoice. (In a real system,
        there would likely be more steps involved here.)
        Go to next record.
        If there is no next record (you're at the end of
        your found set), then exit the loop.
        Else begin loop again (go to the "Begin Loop" step).
End Loop
Exit Script
```

Notice that an exit condition is established. The system tests in both `If` branches whether you're at the end of a found set and exits the script regardless of whether the last record in the set closes. Imagine you're a user doing this manually. You would start at the top of a found set, use the book icon to page through each record one at a time, and then stop the process when you reach the end of your record set.

A loop exit condition is almost always useful if something changes during the course of a script. It's possible you might be sitting in a loop, waiting for something elsewhere to change and checking periodically, but this kind of polling activity is not commonly needed in FileMaker Pro.

Loops become more interesting when combined with conditional logic more complex than checking for an incremented counter. Loops can be exited in various ways. A common technique is to use the `Go to Record/Request/Page [Next, Exit After Last]` script step. It enables you to step through a found set and exit a loop after reaching the last record.

Another way to exit a loop is to exit or halt the script altogether. You have two processes running: the script itself, which you can terminate, and the internal loop.

To cope with endless loop problems, **see** *"Testing Loops" in the "Troubleshooting" section at the end of this chapter.*

Working with Custom Dialogs

One of the most common user interactions necessary for a system is to capture a response to a question. "Are you sure you want to delete all records?" "Do you want to report on all records, or just your found set?" "Would you like fries with that?"

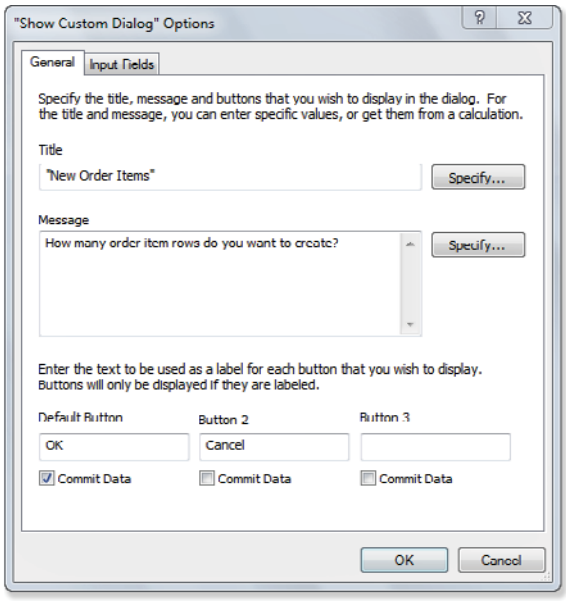
The `Show Custom Dialog` script step is a great, built-in way to capture this sort of interaction. (There are ways to create layouts that act and behave like dialogs, but they're a good bit more work.) Custom dialogs allow you to present some descriptive text or a question to a user and then capture a response.

To learn how to create pop-up layouts that behave as modal dialogs, **see** ***"Multiwindow Interfaces,"*** **p. 473.**

Naturally, after you create a custom dialog, you need to deal with the results. FileMaker Pro stores the user's button choice until the end of the current script or until you present another custom dialog. Think of these dialogs as existing solely within the space of a given script.

To identify which response the user chose to your dialog, use the `Get ( LastMessageChoice )` function. This function returns a `1`, `2`, or `3` based on which button the user clicked. As you can see in Figure 9.13, you assign names to each of the buttons in a dialog. You also can specify whether each button commits data (that is, serves as a standard OK button). Typically, only one of the buttons is a commit button, but in some cases you might want to use two or even three buttons for that purpose; you then would sort out the specific action to take based on the actual button that was clicked. The label you assign to the button is inconsequential.

Figure 9.13
Set buttons for your custom dialog.



Conditional scripting allows you to test the choices a user made and respond accordingly. Here's an example:

```
Report_Revenue_Start
Show Custom Dialog [ Title: "Revenue Report"; Message:
"Do you want to view a Revenue
➥Report by month, year, or a date range?";
➥Buttons: "Range", "Year", "Month"; Input #1:
➥Invoices::Date_Range, "Date Range (e.g., 1/1/2004...2/15/2004)" ]
If [ Get ( LastMessageChoice ) = 1 ]
    Perform Script [ "__Report_DateRange" ]
Else If [ Get ( LastMessageChoice ) = 2 ]
    Perform Script [ "__Report_YearSummary" ]
Else If [ Get ( LastMessageChoice ) = 3 ]
    Perform Script [ "__Report_MonthSummary" ]
End If
```

Custom dialogs are flexible, but they do have limitations. The most obvious limitation is that you cannot alter their appearance or size. In a FileMaker layout, you can apply images, background color fills, and other graphical attributes of the screen. Not so in a custom dialog. You are limited to a system-style dialog.

See *Chapter 14 for other ways to approach this issue.*

The second limitation of the dialog lies in scope: You're limited to three input fields and three buttons. If you need anything more complex, you have to use a standard FileMaker layout to build a custom pop-up layout.

For more ways to handle windows and dialogs, **see** *"Using Styles,"* **p. 385.**

tip

Dialogs of all sorts were quite popular in the early days of personal computers. Today, dialogs are used less and less frequently because they bring processing to a halt until the user deals with them (which is the point). Allowing users to continue with other operations, as you can do by using multiple windows, increases user control and productivity. If you have used versions of FileMaker Pro before FileMaker Pro 9, compare the modal, dialog-based ScriptMaker interface with the current window-based interface.

Starting and Triggering Scripts

One of the most significant features in FileMaker Pro 10 and later is the capability to use script triggers. One of the side effects of this major new feature is that you now need to distinguish between starting a script and triggering a script. In previous versions, there was no distinction, but now it is important to use the terminology carefully.

Starting Scripts

Starting a script means just that—doing something to cause a script to run.

There are several ways to start scripts:

- By selecting a script via the Scripts menu
- By establishing a custom menu item tied to a script
- By opening the Scripts dialog, selecting a script, and clicking Perform
- By calling a script from another FileMaker script (within the same file or externally)
- By calling a script from an external web source
- By attaching a script to a layout element, which a user then clicks (and thus it becomes a button)
- By calling it from AppleScript or VBScript

To start a script, the user generally has to actively click something. You can attach scripts to layout objects so that a user clicking the object triggers them, or they can be activated directly from the Scripts menu, if you choose to make particular scripts visible there. You can also use FileMaker Pro Advanced to create custom menus that run scripts. There are other ways to call scripts externally through web publishing as well.

➥ *To tie a script to a custom menu item,* ***see*** *Chapter 14, "Advanced Interface Techniques."*

➥ *To call scripts externally through Instant Web Publishing,* ***see*** *Chapter 24, "Instant Web Publishing."*

Triggering Scripts

A script trigger fires automatically when certain events occur, such as editing a record or opening a layout. In general, the user's focus is on entering data, going to a new layout, going to a record, or going from one FileMaker mode to another. The developer associates a trigger with specific events and specific objects (such as fields), layouts, or files. The user does whatever she wants to do, and the script is coincidentally triggered.

There are many significant advantages to being able to use script triggers. Perhaps the most common is that users do not have to remember an extra step. Most of the time, if you have to tell a user, "Remember, after you do *X*, make certain you do *Y*," you can create a trigger that will always fire when *X* happens, and FileMaker will remember to do *Y*.

➥ *Triggers are discussed extensively in Chapter 17, "Working with FileMaker Triggers."*

Working with Buttons on Layouts

More often than not, clickable layout objects are graphical buttons, but it is possible to attach a script to anything you can place on a layout: a field, a graphic, even a portal. These layout objects then become button-like so that when a user clicks such an object, the script associated with the object runs.

Creating buttons on FileMaker layouts is straightforward. You can opt to use the Button tool to draw a 3D-esque button, or you can attach a button behavior to any object on a layout (including fields, merge fields, text, images, and even binary files pasted onto layouts).

Apply button behaviors to an object either by right-clicking and choosing Specify Button, or, with a layout object selected, by navigating to the Format menu and choosing Button.

Given that fact, when one is adding interactivity to a button, why use any of the other single script steps other than `Perform Script` itself? Well, we're going to argue that you shouldn't. If you use single script steps, you're out of luck if you ever want a button to do two things. You're out of luck again if you create a bunch of buttons that perform the same step (such as `Go to Layout`) and you need to change them all—it's insufficient abstraction to not allow the same button behavior to be reused elsewhere. Because it's likely that you will want to add steps, or duplicate a button and edit its behavior globally, you should ignore every button behavior other than `Perform Script`.

tip

We've talked about buttons as a tool for starting scripts. This is actually a little inaccurate. A button, when clicked, can perform any single script step: `Go To Layout`, for example, or `Hide Window`. Of course, one of the available script steps you can attach to a button is `Perform Script`. Choose that option, and your button can perform a script of any length or complexity.

Even if a script is one step long and is likely never to be reused, take the few extra seconds to create a script. If the button performs a script, you can easily add steps whenever you need them, and you can change at once all the buttons that need to go to, say, the Invoice layout. After you select the script in question, you can opt to modify the behavior of the script that might or might not be currently running.

For more details on controlling script flow via button attributes, **see** *"Script Parameters,"* **p. 443.**

Naming Scripts

As you saw in Figure 9.1, scripts can include special characters in their names—the | character is a good example. There is much discussion among FileMaker developers about appropriate naming conventions for scripts (as well as tables, layouts, fields, and internal variables). Names that may be exposed to other database management systems must conform to the strictest rules.

tip

You can often work around limitations on the use of special characters with escape characters, but that makes your code difficult to read.

The most basic rule for naming everything in a database environment is to make the name clear: In the case of scripts, it should describe what the script does. As described previously in this chapter in the "Using a Script Template" section, you can structure your scripts so that they contain comments and documentation in a structured fashion so that you know where to look in a script—indeed, in every script that conforms to the template—for detailed information. This normally includes a summary of the script's actions, information on its parameters, and a description of its result. Along with this information, a log of modifications is invaluable.

Beyond that, it can be useful to identify the type of script it is: For example, is it a script that is designed to be run by a trigger as opposed to user interaction? Along those lines, you may want to separate scripts that interact with users from those that have no interaction. You can add this information in a naming convention, but gradually you may find your script names becoming longer and longer.

Some people use single letters to identify aspects of the script as in a convention such as `GetAdjustedValue | i` and `DoAdjustValue |`. This particular convention might indicate that `GetAdjustedValue` has user interaction (the `| i`) whereas `GetAdjustedValue` simply gets a value and has no user interaction. There are many such naming conventions you can consider. If at all possible, be consistent within a single project.

Troubleshooting

Lost Error Messages in Scripts

My script is not working properly, but I'm not getting any error messages. Where do I start?

Be sure that you properly account for potential errors if you turn on error capture. What if a find request returns zero records? What if a user doesn't have access to a given layout needed for a script? To manage debugging, turn off error capture while you're testing. Some developers write scripts that toggle error capture for all scripts in a system. This is a convenient way to turn on and off a debugging mode.

Unfinished Scripts

I need a script to run to completion without fail. I set `Allow User Abort [off]`*, but it appears that a user aborted the script at some point. How can I make sure that users can't muck with my scripts?*

Remember that turning off `Allow User Abort` doesn't always save you from errors in the script itself, power outages, the user closing FileMaker Pro, or other random acts of unpredicted computer wonkiness. You can never absolutely depend on a script completing in FileMaker Pro. If necessary, write a "check conditions" script in your system and run it when appropriate. Another way to deal with this problem is to write a script log that saves a record when a script starts and another when it ends. You can check for incomplete pairs.

Editing the Correct Related Records

My `Set Field` *script step is continually changing the first record in a portal instead of the one I want. How do I get the script to act on the proper row?*

Be careful when setting fields through relationships. It's possible to think that you're pointing to a single record when you're really pointing to the first of many. In that case, FileMaker blithely applies your script steps to the first related record it finds. Either put a button directly in a portal, in which case the script will apply to that row, or use a `Go To Related Record` script step to explicitly control both the context and the record against which a script operates.

Conditional Error Defaults

My `If/Else` *statement isn't returning the proper result. How can I test what's going on?*

Be sure to account for all variations of logic in your conditional scripts. We strongly recommend that you build `If` routines that end with an option you think will never occur. Here's a quick example:

```
If [Invoices::Total > 0]
    Do something
Else If [Invoices::Total = 0]
    Do something
Else If [Invoices::Total < 0]
    Do something
Else
    Handle error conditions here
End
```

This function should *never* return the default error, but you cannot perfectly predict all such behaviors. For example, what if a calculation for Total is wrong and returns a null or empty value? Or if a calculation you expect to be numeric returns text in some cases?

Testing Loops

My loop seems to be stuck endlessly looping. How do I debug the problem?

Rare is the developer who gets everything right the first time, and if you don't, you might find yourself in the middle of an endless loop. A handy trick is to always create an exit condition that tests whether the Shift key is held down by using the `Get( CurrentModifierKey )` function. It's a backdoor out of your loop that's quite handy if you have an error in logic. A much easier way to go if you own FileMaker Pro Advanced is simply to turn on the script debugger the first time you test a new loop.

FileMaker Extra: Creating a Script Library

You might consider having a utility file sitting around your hard drive with all the basic scripts each of your solutions will need. You can then import these scripts into your own solution files as needed. We always have the following in our databases:

- `StartUp`—Here's a script we use to open all the files of a given solution at once, to set default values for globals, to set a login history record if necessary, and so on.
- `ShutDown`—The partner for `StartUp`, the `ShutDown` script can close out your user session by setting any tracking information and can close all the files in a solution so that FileMaker Pro need not be quit.
- `ToggleAllStatusAreas`—This is another critical script for developers working in multiple files or windows. Very often we'll close and lock the Status toolbar to maintain control and keep users from accessing records or layouts we have carefully scripted around. This handy script reopens the Status toolbar for development.
- `ToggleMultiUser`—This script simply turns on or off peer-to-peer sharing. It is useful to use the `Set Multi-User [on/off]` script step when you need to isolate your system during testing.
- `InitializeGlobals`—Often a subscript of `StartUp`, but best abstracted as it is here, this script sets all the initial values of globals and global variables in your system, ensuring that they all start out user sessions in a predictable state. You need to add explicit steps for each global you add to your system as you work, but you'll find it invaluable to have a "global" global initializer.
- `ScriptTEMPLATE`—This is the template we duplicate for new scripts. It has initial comment headers and default script steps as needed.
- `PrintSetUp_landscape` and `PrintSetUp_portrait`—Every printer-bound output of your system needs page properties established. Write them once.

If you find yourself writing certain scripts time and time again, add them to your library. Using the capability to copy and paste scripts in FileMaker Pro Advanced enables you to leverage prior work more easily.

This page intentionally left blank

10

GETTING STARTED WITH REPORTING AND CHARTING

Reporting in FileMaker Pro

You have seen how to use layouts to display data and allow it to be entered. Both Browse and Find modes are designed for interactive use on the screen. You can print a layout, but obviously the interactive parts of the layout such as buttons, pop-up menus, and scrolling portals become static when printed. FileMaker Pro provides another mode, Preview mode, that is used for reporting. Preview mode is not interactive. It is designed primarily for printing, so instead of using scroll bars, which require interactivity, it paginates data that is too big for a printed page. In addition, because it is the heart of FileMaker Pro's reporting mechanism, Preview mode supports summaries that can include calculated fields that total, average, or count data. This means that each mode has its own function: Browse for interactive entry and display on the screen; Find for finding data; Layout for designing layouts; and Preview for printing, pagination, and summarizing data.

Beginning in FileMaker Pro 10, major changes were made to reporting. Everything that was there remains, but, in addition, reports and their calculated summary data can be live. This chapter explores reporting in the traditional sense of producing reports that are displayed in Preview mode and are suitable for printing (and are not interactive). Then it explores the FileMaker Pro features that add features of Preview mode to certain views in Browse mode to provide live reporting.

Finally, you learn how to use FileMaker's built-in charting tools to convert your text-based data to charts. Charting is just another way of displaying your data so that people can understand it. You can think of it as just another method for developing reports.

Deriving Meaning from Data

Reporting is an important component of almost every database project. Indeed, the need to create reports that summarize or synthesize data is often the reason many databases exist in the first place. No matter what your database does, it's a fair bet that you have many reporting needs.

Reports come in many shapes and sizes: There are simple list reports, summarized reports, workflow reports, cross-tabulated reports, variance reports, and graphic reports such as charts (to name but a few). There are standard reports that have to be generated periodically; there are ad hoc reports for which the report criteria must be defined on the fly (FileMaker Pro excels at this). Some reports need to be printed and distributed, whereas others are meant to be viewed onscreen.

Despite the wide range of things that can be classified as reports, most reports tend to have a few characteristics in common:

- Reports are generally used for viewing data rather than creating or editing data.
- Reports generally display (or draw on data contained in) multiple records from one or more tables. They are usually designed to provide an overview or higher-level understanding of a data set than you would obtain by looking strictly at data-entry screens or at a single record's data.
- Reports capture a snapshot in time and reflect the database's current state. Running the same report at different times might yield different results if the data in the system has changed. For this reason, reports are often stored for future reference either on paper or on unchangeable reference versions, such as PDF files.
- Often, but not always, reports are distributed by some means other than FileMaker: on paper, via email, or as an electronic document. FileMaker Pro provides the capability to create PDF documents and Excel spreadsheets from reports.

To generate meaningful reports, you should learn several standard reporting techniques. From there, it's just a matter of coming up with variations that suit your particular needs. This chapter covers working with lists of data and reporting with grouped data (also known as *subsummary reports*).

In FileMaker, layouts are used to display data and, in some cases, to allow for its printing as well as its entry in Browse or Find mode. The distinction between layouts designed for the displaying or printing of data and layouts designed for interaction with the user has to do with their design (page width, for example) and, in many cases, the mode in which they are viewed. Data entry can be done only in Browse mode, and entry of data to be used in a find operation can be done only in Find mode. Preview mode, which allows no interaction, is often used to display reports. It must be used for you to be aware of page breaks (which is the first step in using headers and footers and creating page numbers), as you will see in this chapter. Before FileMaker Pro 10, it was required to view subsummary reports.

Begin with the End in Mind

One of the keys to creating successful reports is beginning with the end in mind. Right at the beginning of a project, you should start thinking about the reports that a system will have to generate. A system's intended outputs can have a profound impact on its design and implementation.

Determine Report Requirements

Just as a system's reporting requirements influence its design, an organization's business needs influence the design of the reports themselves. When thinking about how you'll go about generating any given report, ask yourself (or your client/users) the following types of questions:

- What questions is this report trying to answer? Focus first on the purpose the report will serve, not on its design. Is it trying to monitor progress toward a goal? To be an early warning of potential problems? To help spot business trends? The more you know about how a report will be used, the more effective you can make it.
- Who will read this report? Is it going to be used strictly for internal purposes, or might it be presented to customers or vendors? Should the report be accessible to everyone, or should certain users be prohibited from viewing it?
- How will be it read? Will it be distributed in hard copy, emailed to a group of people, or read onscreen 18 times a day? If the report is distributed, should the document be secured with a password or encrypted?
- What media constraints are there in terms of page size, color, and resolution?
- Is this a one-time report, or will it be used on a regular basis? For one-time or special occasion reports, you probably won't go to the trouble of setting up scripts and/or find screens, but you should do so for reports intended to be run regularly.
- What level of granularity is appropriate? Will the consumers of the report be interested in seeing details or just the big picture?

After you collect answers to questions like these, we strongly recommend writing out a sample report (using whatever tools you choose—pencil and paper and whiteboards are our favorites) and showing it to its appropriate consumers for feedback. Although the report will be implemented in FileMaker, any tool from Excel or Numbers to InDesign can help people visualize what you are thinking of.

Generic Versus Specific Report Structures

Another part of report planning is determining whether the report is to meet a specific or a generic need. That is, should users be able to select a data set to feed into a report shell, or should the search criteria for the report be hard-coded?

For example, say you have a List view layout that displays customer data. If you feed it a found set of customers obtained since a certain date, it becomes a New Customers report. If you feed the same shell a set of inactive customers, it transforms into an Inactive Customers report.

In instances like this, it's often helpful to think of a report as consisting of two distinct components: its format and its content. If you can create a generic multipurpose format, you create different reports simply by sending in different content. The point is that in planning reports, you should have the distinction between format and content in mind. You can sometimes save yourself a lot of work if you recognize when a report can be created by simply feeding new data into an existing format.

The simplest way to do this is to use a script for the data selection and sorting and a second script for the report. Sophisticated users can do the sorting and selection manually, whereas others might choose to use the script. The script can be highly interactive, allowing users to choose the data to find and the way it should search. Alternatively, there can be multiple scripts, each of which does a specific set of sorted data for the same report. This is an excellent model for scripting: One script is interactive, and the other—the reporting script—has no interaction whatsoever.

Working with Reports, Layouts, View As Options, and Modes

FileMaker Pro provides a number of ways of viewing data: Layouts provide the overall format, and the specific view shows that layout as a form (one record per screen), as a list (a scrolling list of layouts on a single screen), or as a table (a spreadsheet-like display that is based on the given layout's fields). Figure 10.1 shows the Inventory Starter Solution's Product Details layout, which is shown as a form.

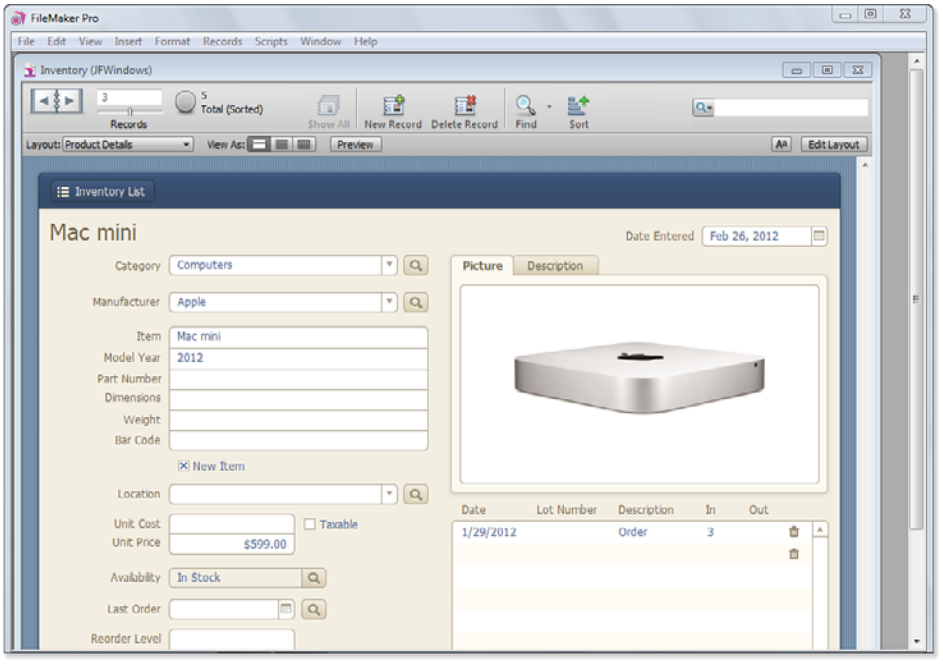


Figure 10.1 The Inventory Starter Solution's Product Details layout is shown with the View As Form option.

You can select a specific layout and its default viewing option by using the Layout pop-up menu in the Status toolbar (you can also use a script to select a layout as well as a viewing option). You can choose the View As option from the buttons on the Status toolbar. These choices can also be made in scripts or by using the View, Go to Layout submenu to select a layout and the View, View As Form/List/Table commands.

If you switch to the Inventory layout (which has the View As List option set as its default), you see the one-line layouts in a scrolling list, as shown in Figure 10.2. The black bar at the left of each row indicates the current record. The List View layout uses a single line, but you can have larger layouts shown in View As List if you want. To experiment, select the Form View and choose View As List.

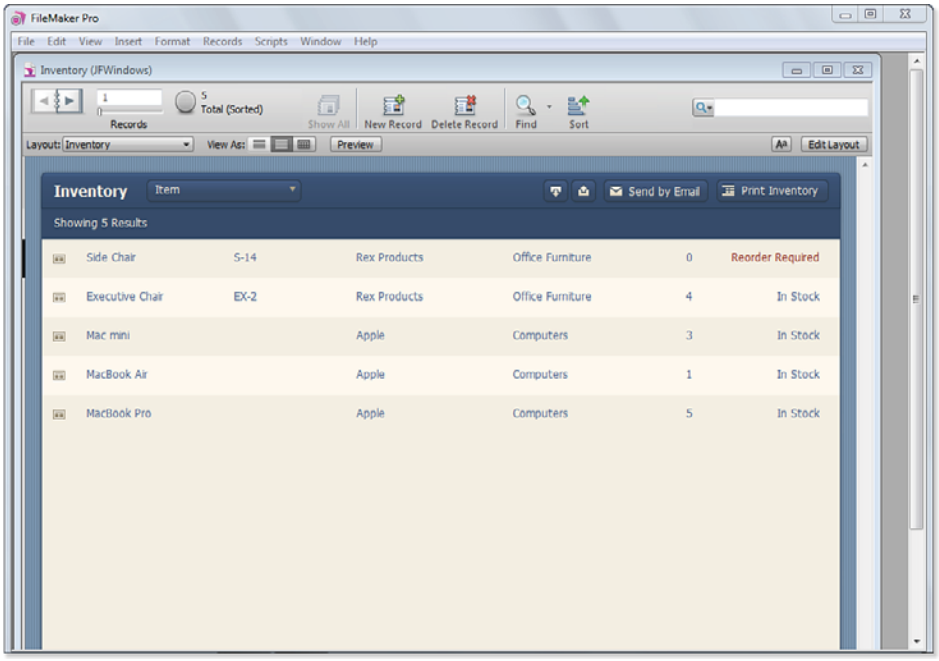


Figure 10.2 Use the Inventory layout with the View As List option.

Both Figures 10.1 and 10.2 show layouts in Browse mode. You can edit the data. If you click the Print Inventory Report button at the upper right of the view shown in Figure 10.1, you will run a script that sorts the data, switches to the Inventory Report, and enters Preview mode, as shown in Figure 10.3. At the center of the Status toolbar, you can see that the script is paused; you can use the Exit Preview button to move on. Very few commands are available at this point. If there are multiple pages to the report, you can use the book in the Status toolbar to move from page to page; you can also print some or all of the pages. You cannot enter data, however.

note

Although this book focuses on FileMaker Pro and the ways in which you can create solutions that run on a variety of devices, it is worthwhile to take a look at Figure 10.4, which shows the same Inventory starter solution running on an iPhone.

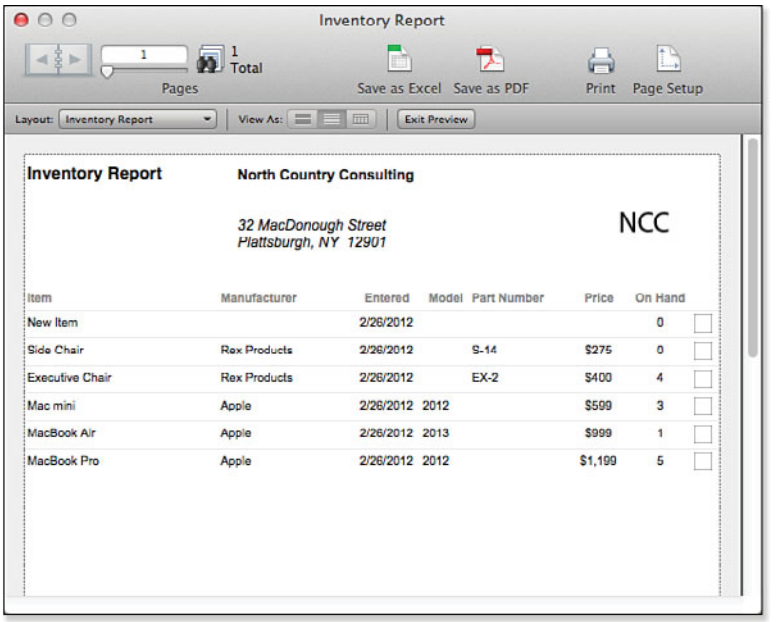


Figure 10.3
The Inventory Summary Report is displayed by a script in Preview mode.

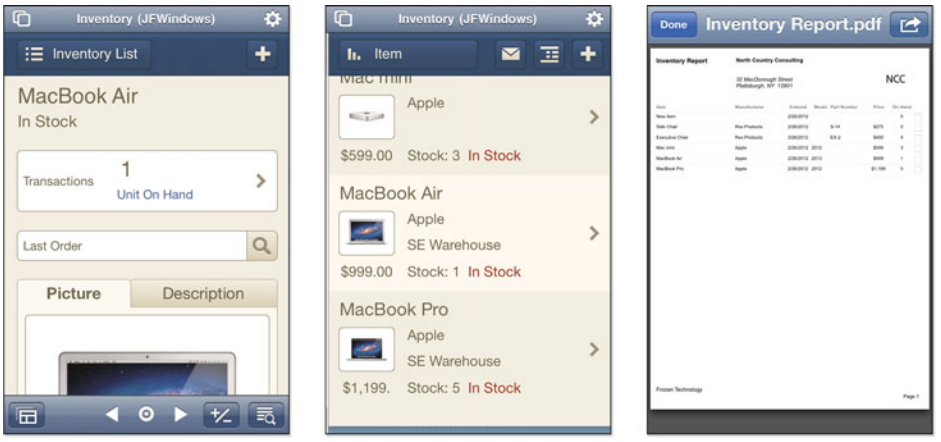


Figure 10.4
Use the same solution on an iPhone.

note

In fact, what is happening in the previous figures is that the Inventory Starter Solution is running in a copy of FileMaker Server (the server is named JFWindows). Throughout this chapter, you see a Mac, a Windows computer, and an iOS device (in this case an iPhone) connecting to the same database. You can enter data on the iPhone and view it on the Mac as soon as you leave the field into which you entered data. Note in Figure 10.4 that the report shown full-size in Figure 10.3 is shown in a compressed version on the smaller screen, but you can print it or email it from the iPhone. The printed or emailed version will be full size.

Working with Lists of Data

List views are easy to create; they make nice reports for several other reasons. The first is that they're very flexible. You can allow users to perform ad hoc finds, or you can write scripts with canned searches and then display the results using your List view.

Users can use list reports while in Browse mode or in Preview mode. We recommend that you consider the final delivery of a report as a separate issue from generating the report for users to view onscreen. We often design systems in which a report displays for a user (in Browse mode), and then the user can, as a second step, send it to a printer, attach it to an email, and so on.

The key benefit of being able to work with a report in Browse mode is that you can place buttons on your report that give the user additional functionality, such as drilling down to additional levels of detail, re-sorting the data without having to regenerate the report, and providing buttons for printing, emailing, and so on.

You might have buttons or other objects, such as navigation buttons, on your layout that you wouldn't want to appear when the report is printed. While you are building the report in Layout mode, select those objects. Then, in the Inspector, choose the Position tab and select Hide when printing in the Sliding & Visibility section toward the bottom.

*To find more information about these controls, **see** "Sliding Objects," **p. 308**.*

*If you have problems printing your reports, **see** "Printed Reports Show Only a Single Record" in the "Troubleshooting" section at the end of this chapter.*

note

If your users are likely to print from a List view, be sure that you constrain your report to the width of the printed page rather than the monitor screen width. You'll also find that although 10- to 12-point fonts generally work well for reports that will be viewed onscreen, 8- to 10-point fonts are more appropriate for printed reports. Be sure to actually print your reports to proof them rather than simply relying on what you see onscreen.

Using the New Layout/Report Assistant

In Layout mode, you can choose Layouts, New Layout/Report to start the interactive tool that helps you design layouts and reports. This section provides an overview of the process. Along the way, you can choose a variety of options. On completion, you can modify the layout itself in a variety of

ways. Many people start with the assistant and then provide their own customization in this way. The distinction between layouts designed for printing and summarization—that is, reports—and other layouts is based on their use. *Reports* are a special type of *layout*, and in this section the terms are used interchangeably. In this section, you learn how to create a new list report. The built-in list shown in the Inventory layout simply lists the items. This one lists them and groups them by location and category.

To begin, enter Layout mode and choose Layouts, New Layout/Report. The dialog shown in Figure 10.5 opens.

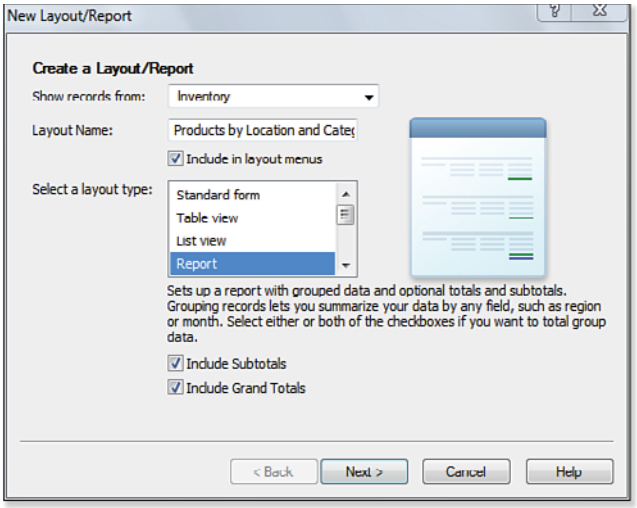


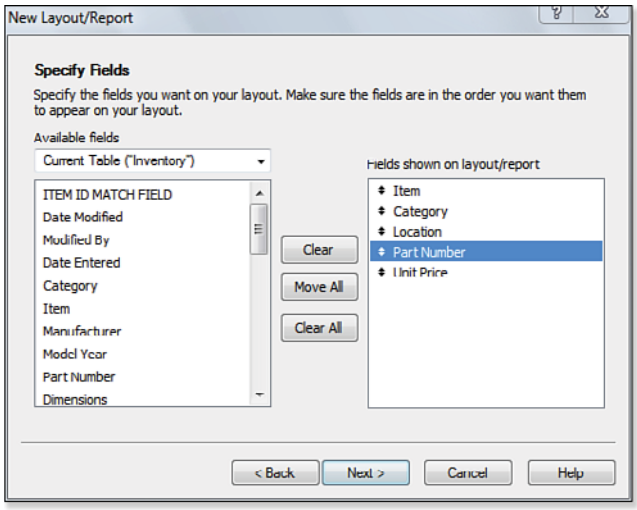
Figure 10.5
Select the base table and name the layout.

You can select from various types of reports. As you select each type, the thumbnail at the right of the dialog changes to show you the basics of the layout you have selected. In this example, a list report is developed. (Note that layouts with "view" in the name are usually designed for the screen, whereas layouts with "report" in the name are usually designed for printing.)

If there are groups, you can opt for a grand total, subtotals, or both. As before, the thumbnail to the right shows a schematic of what the report will look like. Note that this schematic is logical, not actual. Subtotals and totals can be right- or left-aligned; the point is that they exist. In addition, subtotals can precede or follow their detailed data; in the schematic, they are shown as following the detail data, but you can move them during the development process.

Next, specify the fields for the report (see Figure 10.6). You can use fields from the base table or from related tables.

Figure 10.6
Select layout fields.



If you have chosen to group data, the next step lets you select the fields for grouping, as shown in Figure 10.7. At this stage, it is not uncommon to discover that a field you want to use for grouping was not included in the list of fields in the window shown in Figure 10.6. That is no problem: You can go back to a previous step (and frequently may need to do so). As you select grouping fields, the thumbnail shows how the layout is proceeding. If you select multiple grouping fields, they will be indented as subgroups. To change the ordering, move report categories up or down using the two-headed arrow.

The next step is not shown; it is a standard sorting dialog that lets you determine how the data will be sorted. The sorting order includes the grouping fields already selected; they are shown with padlocks because you cannot move them. (If you move them, the grouping will not be done properly. If you change the order of the groups as was shown in Figure 10.5, these locked sort fields will automatically be rearranged.)

You can add additional sorting fields below them so that within a group (or subgroup) data is sorted alphabetically, by date, or any other way you want. After the sort, the Next button lets you add subtotals if you chose them in the window shown previously in Figure 10.5. Figure 10.8 shows this process. You select the summary field for the category, indicate if the summary should be above or below the detail records, and then select a summary field.

*For more details on summaries, **see** "Using Summarized Reports," **p. 295**.*

tip

Groupings are often called *summaries* and *subsummaries*. In those roles, they contain FileMaker summary fields. However, you need not include summary fields in summaries and subsummaries. As is the case in this example, you can sort and group the data and display text in the summary or subsummary (here, it will be the name of a location or category).

tip

If you have forgotten to create a summary field, you don't have to worry. Just click Specify as shown in Figure 10.8. The Specify Field dialog shown in Figure 10.9 lets you select the field. If you haven't created it yet, the Add button in the lower left lets you do so at this point without interrupting your work.

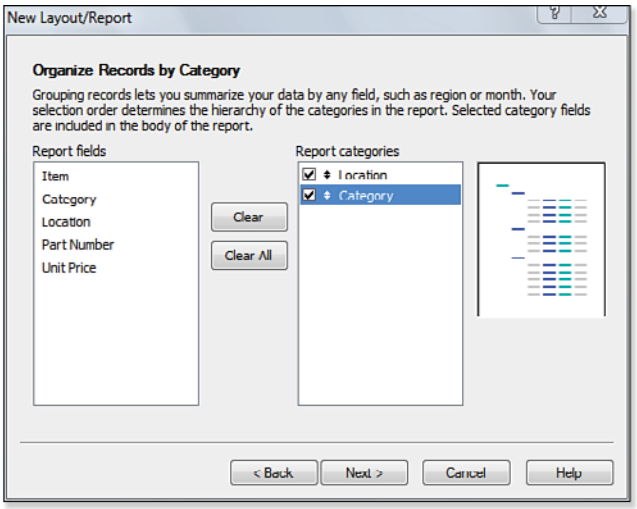


Figure 10.7
Select the fields to use for grouping.

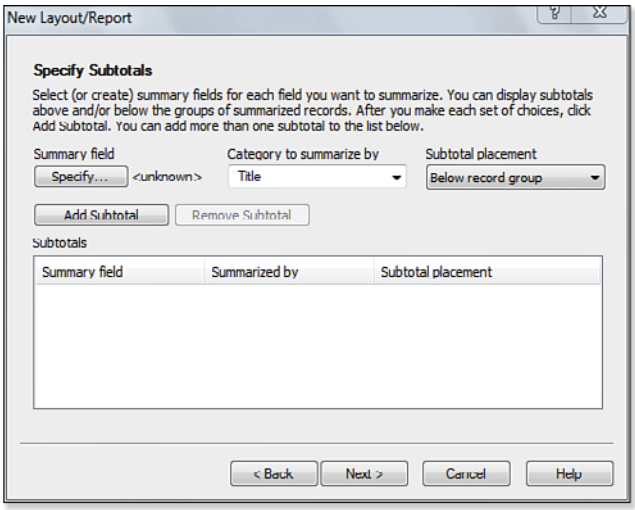


Figure 10.8
Select summary fields for subtotals.

*For more details on creating and using summary fields, **see** "Using Summarized Reports," **p. 295**.*

In the next step, a similar process is used to determine the grand total summary. Following that, you can choose the theme for the layout as shown in Figure 10.10. You are almost finished.

Figure 10.9
Select or add the summary field.

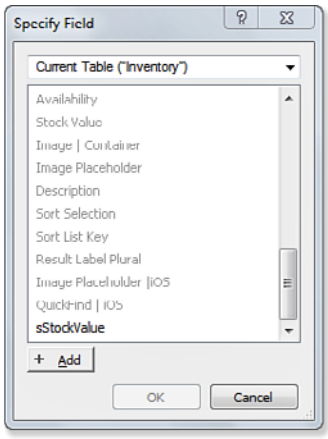
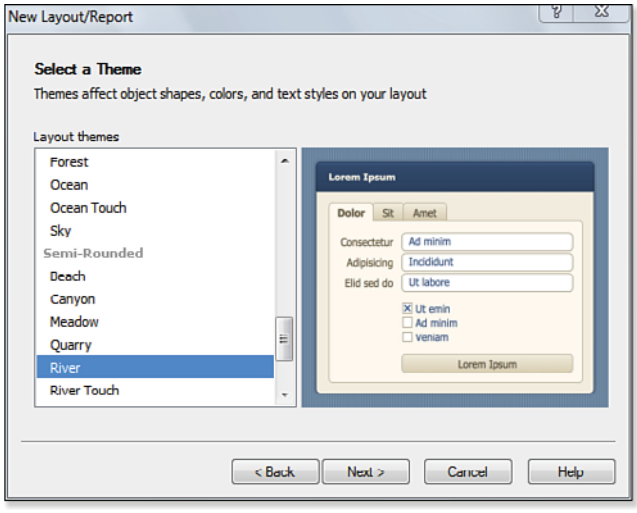


Figure 10.10
Choose a theme for the layout.



In the next step, you can set titles, page numbers, and the like, as shown in Figure 10.11.

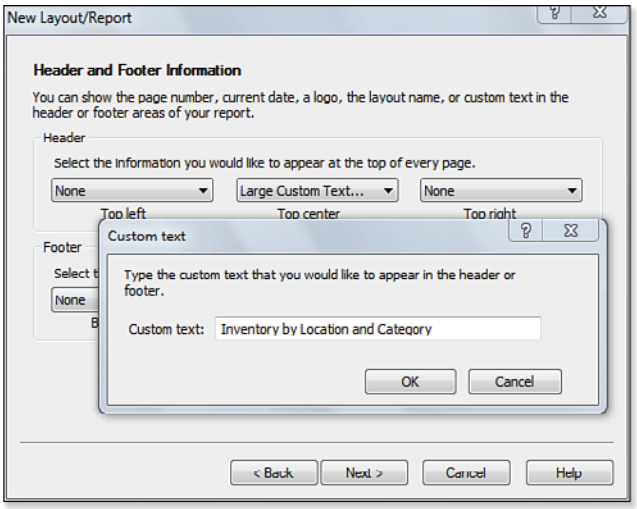


Figure 10.11
Set titles, page numbers, and the like.

In the penultimate step, you can choose to create a script that will sort the appropriate database and go to the layout. Finally, you can also decide whether to view it in Browse or Preview mode. Having completed these steps, you can use your layout/report with its script, or you can customize it.

If you have chosen to create a script, run it and view the report as shown in Figure 10.12.

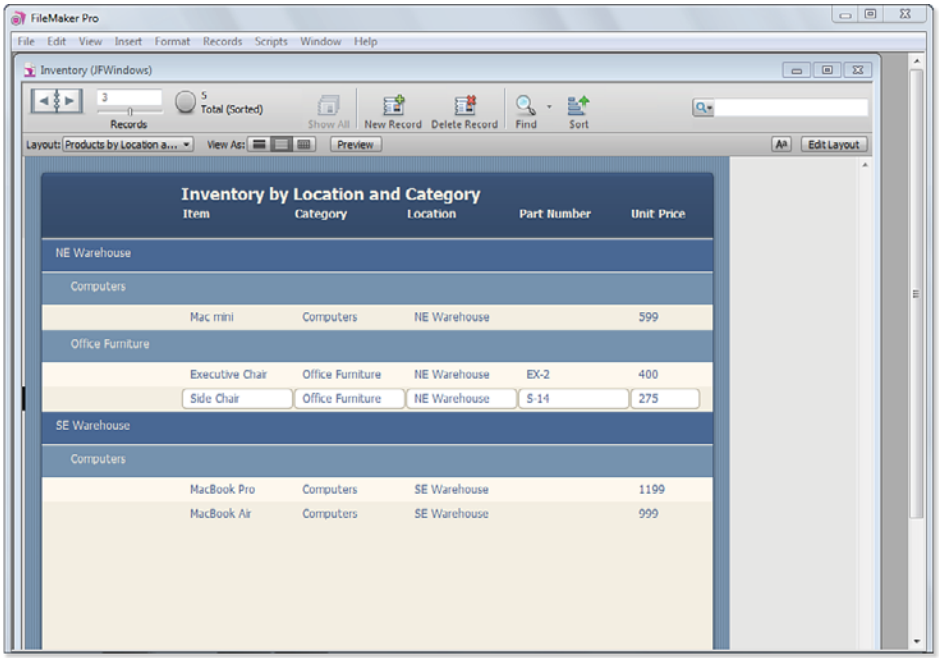


Figure 10.12
View the report.

If you did not create a script, go to the layout and then sort by location and category (in that order). Enter Preview mode to get page numbers if you want.

Using Summarized Reports

As you have seen, you can use the New Layout/Report assistant to create layouts with grouped and summarized data. There are two parts to this. The first is to create summary fields (that is, fields of type Summary, not just an arithmetic or calculation summaries), and the second is to create groups based on a field's values. (This field is called the *break field*.)

The first part—summary fields—is optional. You saw how to create a summary in Figure 10.10. That process lets you select or add a summary field rather than just using group data such as a name. Now you create a summary field.

Creating a Summary Field

A summary field lets you summarize data. The data that you summarize must be from a field in the table you are working with. The Units on Hand calculation uses values from the Stock Transactions table, so you can't summarize it in the Inventory table (see the sidebar, "Summarizing Distant Data" for workarounds). However, there is a Stock Value field in Inventory that can be summarized. This can be a valuable addition to the report: It will show the value of each inventory item, and, because it is a summary field, it will summarize those values for each category and each location.

Summarizing Distant Data

There are several ways to summarize distant data. One way is to simply create a local field that is a calculation and that copies the distant data. This places it in the local table, and you can summarize the calculated field.

In the case of a summarized table such as Inventory, you can use another strategy. In the table as it is shown here, you have a body part that contains records from Inventory. Subsummary parts group that data by location and category.

If you promote the body part to be a subsummary part, the new body part can consist of each transaction from the Stock Transactions table. Then, the body consists of the individual transactions, and you summarize by item (that's the new subsummary part), and then by category and location. With this strategy, you do not even need to display the body part: The subsummary parts show the data, but the body part is summarizing data from its own table.

In this case, there is a further complication, and it is so common that it is worth pointing out. In Stock Transactions, there are separate fields for Units In and Units Out. If you want to summarize them to get a total number of units, you need to create a new calculation field that consists of Units Out prefixed with a minus sign. In other words, you want the sum of Units Out (1) plus Units In (2) to be 1 (2–1) not 3 (2+1).

This happens all the time. Often, people expect to see unsigned numbers in these types of transactions, so that when you add them up, you must adjust the sign of one of the components.

Add the new summary field to summarize Stock Value (see Figure 10.13). Using a common convention, the summary field name begins with *s* (sStockValue).

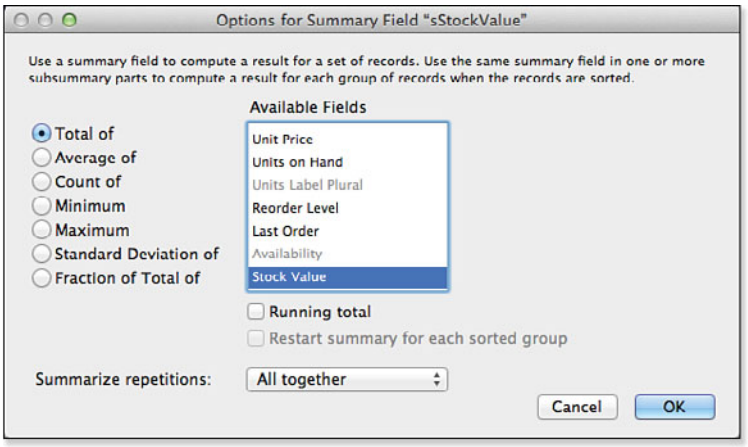


Figure 10.13
Add sStockValue.

Add the field to the layout you have been working on. It can appear in the detail lines of the body as well as in the subsummaries (you will see how it looks in Figure 10.15). Once the field is added, you can rearrange things and use the Inspector to change the appearance of the new field. Figure 10.14 (left) shows you how you can use the Inspector to set the format for a numeric field using the Data Formatting section of the Data tab. Figure 10.14 (right) shows how to right-align the field in the Paragraph section of the Appearance tab. In a column of numbers, aligning them to the right and specifying a fixed number of decimals makes for a better appearance.

Figure 10.15 shows the layout in Browse mode. Note that the summary field is updated automatically in the subsummary part when you select the part or one of its components.

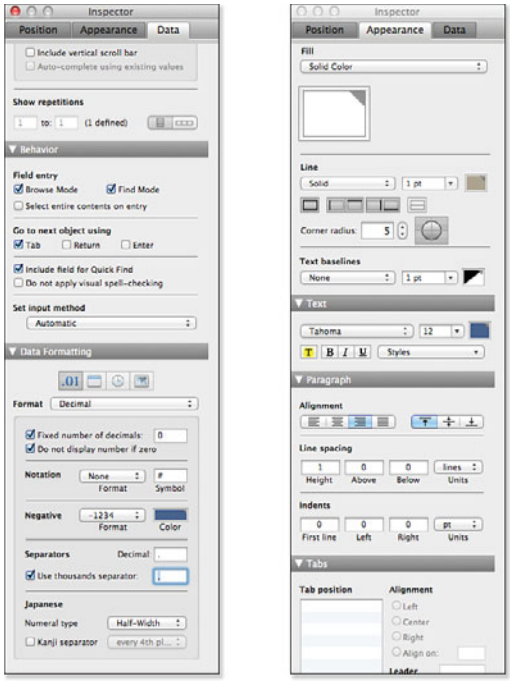


Figure 10.14
Adjust the format for the new field.

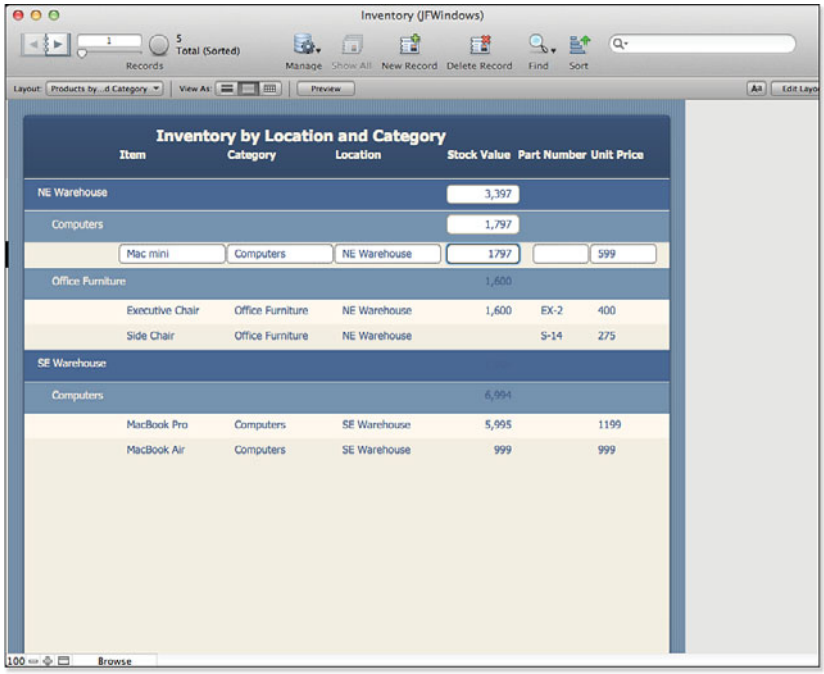


Figure 10.15
Subsummary parts are updated sometimes.

To get the best result, view the layout as Table view, as shown in Figure 10.16. This takes advantage of the dynamic reporting that was introduced in FileMaker Pro 10. You can see that the subtotals are correctly updated.

note
This sequence matters. Create the subsummary parts before switching to Table view.

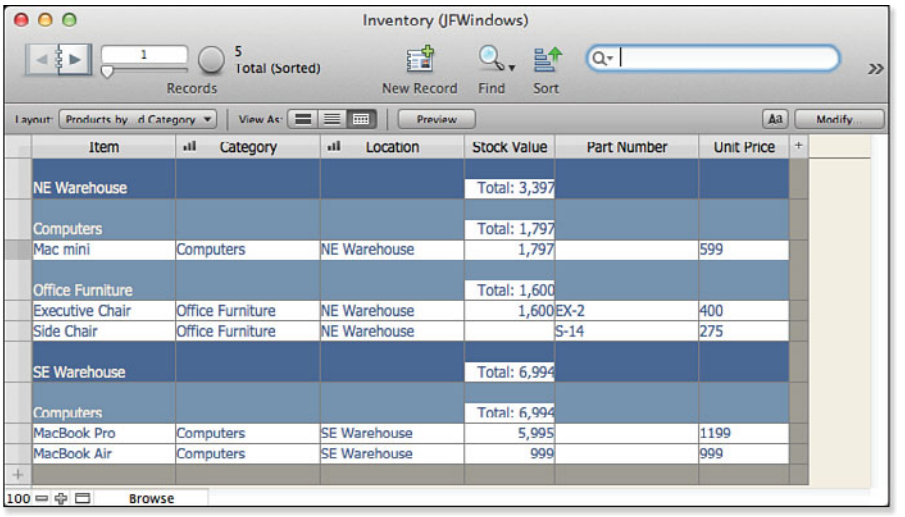


Figure 10.16
Using dynamic reporting in Table view.

Working with Subsummary Parts

Working with subsummary parts is easy when you use the assistant, as shown in the previous section, but you can also do it on your own. Often, you start from an existing report and modify it to add additional break or summary fields. You can use the Part tool, as shown in Figure 10.17, or you can choose Insert, Part to open the Part Definition dialog. You can also use Layout, Part Setup and click Create to open the Part Definition dialog. Whichever method you use, you will open the Part De8finition dialog shown in Figure 10.18.

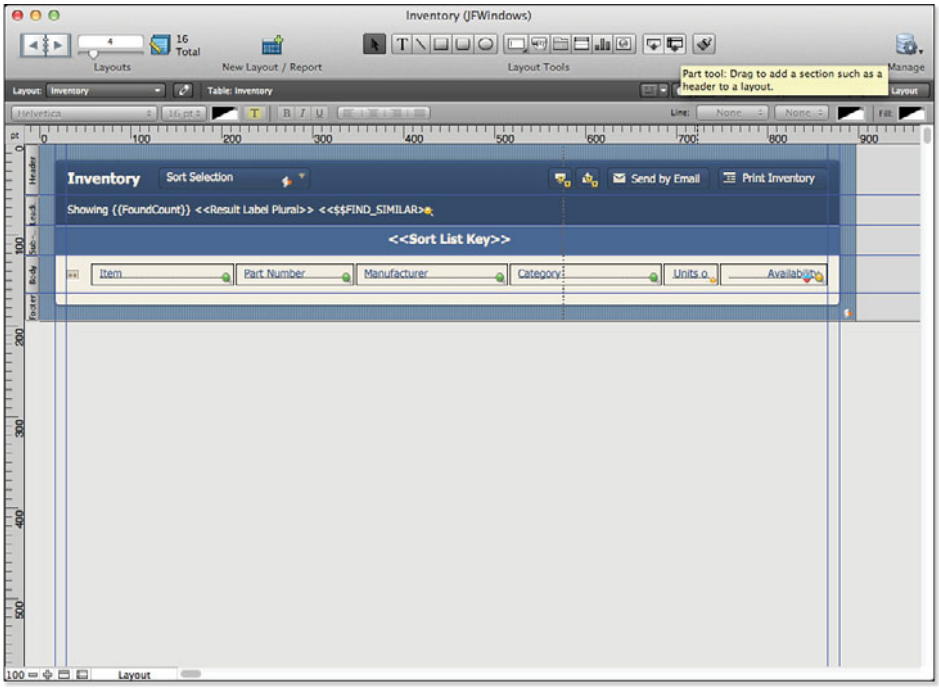


Figure 10.17
Drag the Part tool into a layout to add a part.

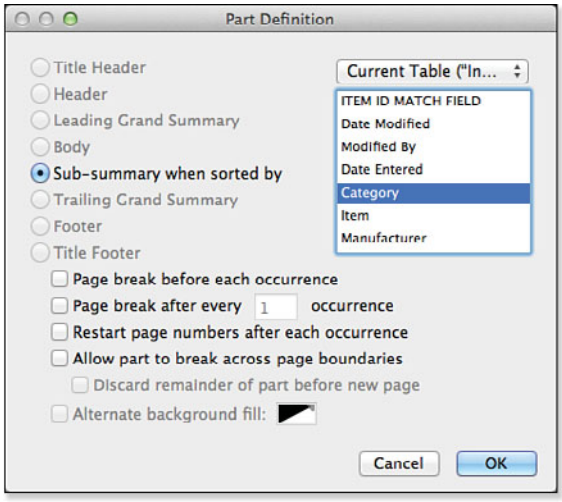


Figure 10.18
Open the Part Definition dialog.

You have seen how this dialog is used for headers and footers, but now you will see how to use it for subsummary parts. You click the radio button to select a subsummary part, and then you select the break field from the list at the right (notice that you can use the pop-up menu at the top to choose another table). The part will be created, and you can set the appropriate pagination options. FileMaker Pro has two ways of identifying parts. The first, shown in Figure 10.17, identifies the parts vertically in the left-hand margin of the layout. You can use the fifth button from the left in the lower frame of the window to switch the labels to a horizontal orientation, as shown in Figure 10.19. Vertically, the labels obscure no data, but they are hard to read; horizontally, they are legible, but they may obscure data. You will probably switch back and forth.

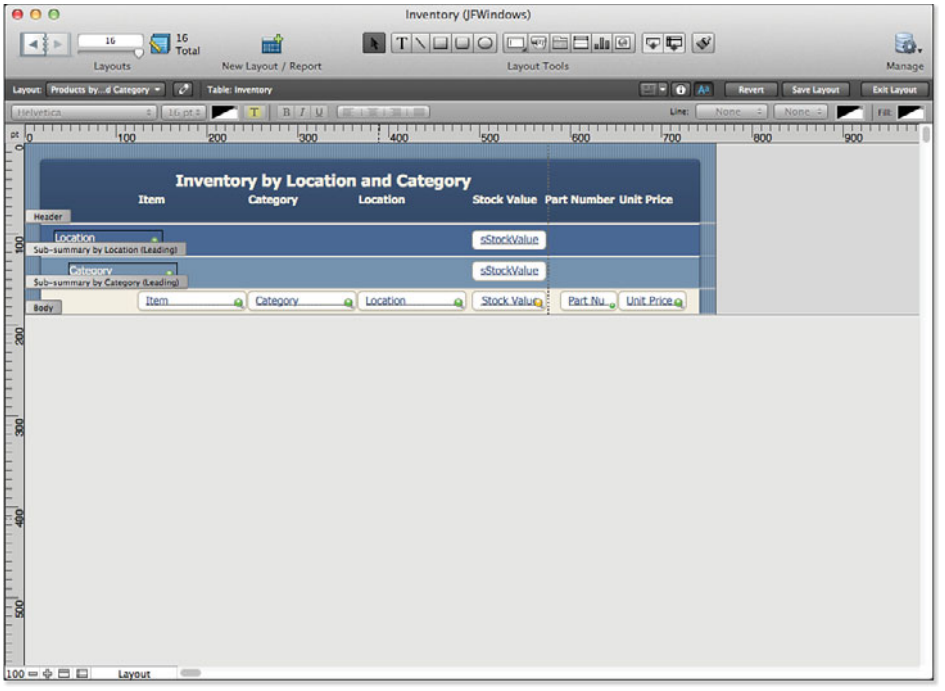


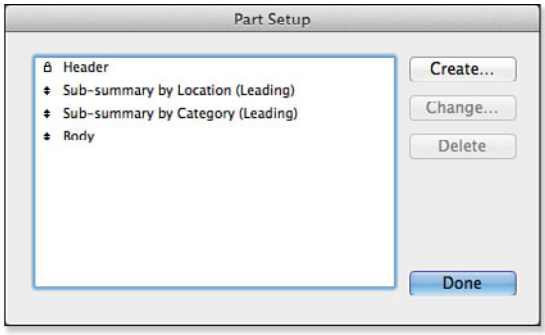
Figure 10.19 You can use horizontal part labels.

When placed in subsummary or summary parts, summary fields behave as you would expect: They provide the appropriate summary for the data in that part. Therefore, it is common to place the same summary field in several parts to provide different but appropriate information. In Figure 10.19, for example, the summary field sStockValue appears three times:

- First, it provides the total for a subsummary based on location.
- Then, it provides the total for a subsummary based on category.
- Finally, it provides the value for a single record in the body part.

To create or rearrange parts, choose Layouts, Part Setup, which will open the dialog shown in Figure 10.20. Some parts cannot be moved (you cannot put a footer before a header, for example). The core of the report is the body part, but you may not need it for summary reports without details; in that case, you can delete it.

Figure 10.20
Create and rearrange parts.



tip

What you must remember about the Part Setup dialog is that the sort order in your script or manual sort must match the implied sort order. The sequence is down toward the body part. Thus, in the example shown in Figure 10.20, the data must be sorted by location and, within that, by category. It does not matter how the data is sorted (ascending, descending, or a custom order based on a value list), but it must be sorted. When you set up the subsummary parts, the sorting sequence must be reversed below the body part.

If you display data for locations and then within locations by categories, the subsummaries are for categories and then for locations. You can use the dialog shown in Figure 10.20 to create a grand summary part that will include `sStockValue`.

When you sort records for a report, you can use a summarized field in the sort. In the example shown in Figure 10.20, the report must be sorted by location and category to work properly. But you can sort based on a summary value to modify that report. As Figure 10.21 shows, you can sort the category field based on the `sStockValue` summary field by clicking Reorder Based on Summary Field toward the bottom of the dialog box.

If you do that, the report will appear with the categories sorted in ascending order by their total cost field.

Calculations Involving Summary Fields

After you begin using summary fields on reports, you're likely to come across situations in which you need to perform some sort of calculation involving a summary field. For instance, in a Student Quiz database, imagine that Quiz 1 is a pre-test for a unit and that Quiz 3 is a post-test for the same unit. You might want to find out the change in scores from the pre-test to the post-test.

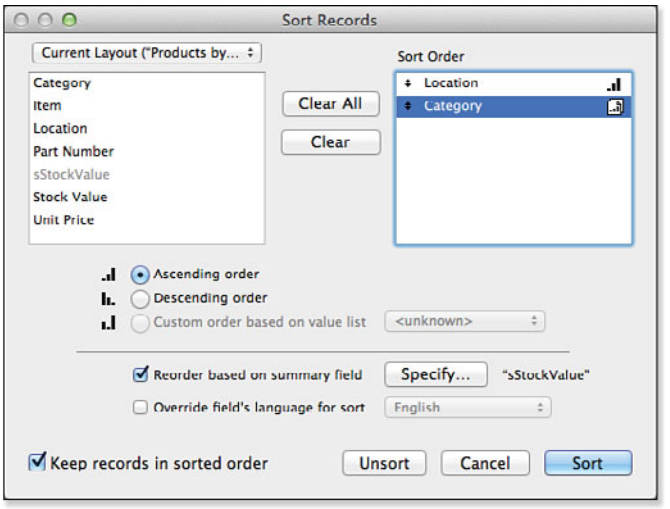


Figure 10.21
Sort on summary fields.

For an individual student, you could generate this information simply by adding a calculation field called something like ScoreIncrease, defined as `Quiz3 - Quiz1`. But what if you wanted to find out the average increase for each class? Can you do math with summary fields?

The answer to the preceding question is both yes and no. Summary fields can be, but should not be, used directly in calculation formulas. There's nothing to prevent you from doing so, but it's usually nonsensical to do so. Inside a calculation formula, a summary field is evaluated as the aggregate result of the entire found set. Thus, if you were to define a field called Average_ScoreIncrease as `Average_Quiz3 - Average_Quiz1`, the result would be `-0.84` no matter what record you were viewing or in what layout part you placed the field. This formula doesn't properly generate subsummary values.

The solution to the problem is to use the `GetSummary` function. `GetSummary` takes two parameters: a summary field and a break field. When the current found set is sorted by the break field, this function returns the same value that would appear if the summary field were used in a subsummary layout part (based on the same break field, of course). If the found set is *not* sorted by the break field, the function returns the value of the summary field over the entire found set, which the astute reader might recall is the same value returned by simply putting a summary field in a calculation without the `GetSummary` function.

In the current situation, to produce a summary `ScoreIncrease` at the teacher level, the following calculation (called Average_ScoreIncrease_Teacher) would be necessary:

```
GetSummary (Average_Quiz3 ; Teacher) - GetSummary (Average_Quiz1 ; Teacher)
```

This field could then be placed in the trailing subsummary part to display the results for each teacher.

The fact that you must explicitly name a break field means that calculations involving summary fields aren't as reusable as summary fields themselves. If you were making another report showing quiz scores by gender, you would need a new calculation field called Average_ScoreIncrease_Gender that specifies Gender as the break field instead of Teacher. Similarly, for use in a trailing grand summary, you would need yet another version of the formula that didn't use `GetSummary` at all.

If this lack of reusability is a problem for you, there actually is a way around the break field problem. The solution is to make a new field—a global text field—that you set (either manually or via script) to be the name of the break field that you need. Then you can dynamically assemble an appropriate `GetSummary` function and use the `Evaluate` function to return the proper value. Using this technique in the present example, you would just define a single Average_ScoreIncrease field with the following formula:

```
Evaluate ( "GetSummary(Average_Quiz3; " & gSortValue & ")") -
Evaluate ( "GetSummary(Average_Quiz1; " & gSortValue & ")")
```

Although the purpose of using a `GetSummary` function is to produce a value appropriate for display in a subsummary part, the values also display properly when placed in a body part. That is, each record of the subgroup knows the aggregate value for its particular set. This is distinctly different from the result of simply placing a summary field into a body part, in which case the value displayed represents an aggregation of the entire found set.

Modifying Table Views

When you choose View As Table in Browse mode, the current layout is shown in a spreadsheet-like display. The fields in the Table view are the fields from the current layout. You can rearrange the columns, but, initially, the columns in the Table view are the current view's fields from top to bottom and left to right.

With the advent of live reporting, you now can modify the fields shown in a Table view rather than using all the fields in the current layout. As you can see in Figure 10.22, you can do this when you are in Table view by clicking the Modify button at the right of the Status toolbar and choosing new fields for the Table view with the Add button at the lower-right of the Modify Table View dialog box. You can use the check boxes to remove fields from the Table view. The initial state—all fields top-to-bottom and left-to-right—remains.

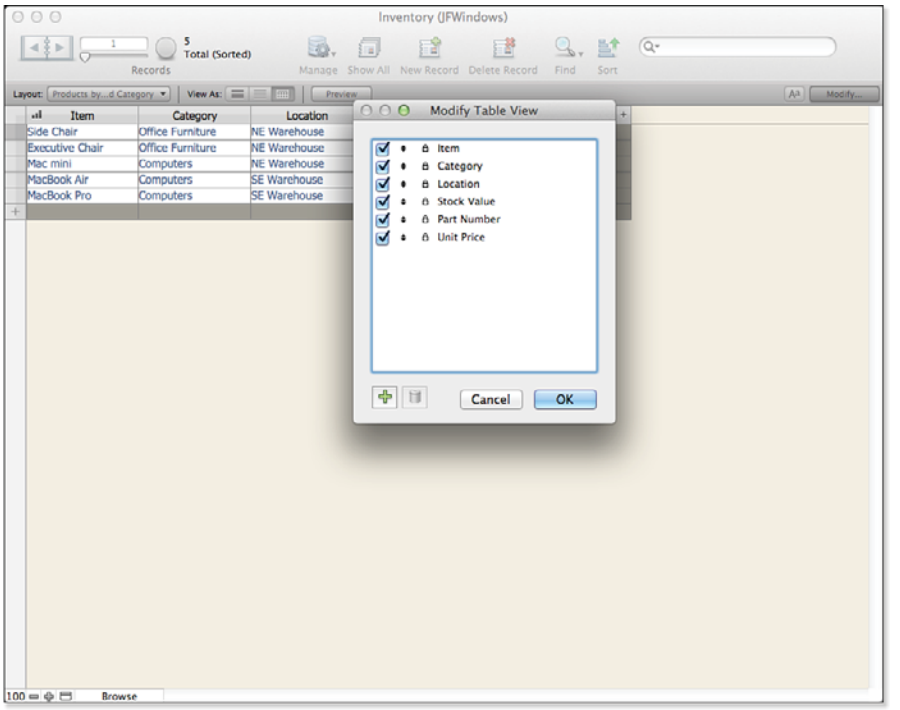


Figure 10.22
Manage fields in a Table view.

Customizing Layouts and Reports

The methods of customizing layouts described in Chapter 4, "Working with Layouts," apply to all layouts. When you are creating layouts designed for displaying data in report formats, several specific customizations are often used:

- Alternating row colors
- Column sorting
- Sliding objects

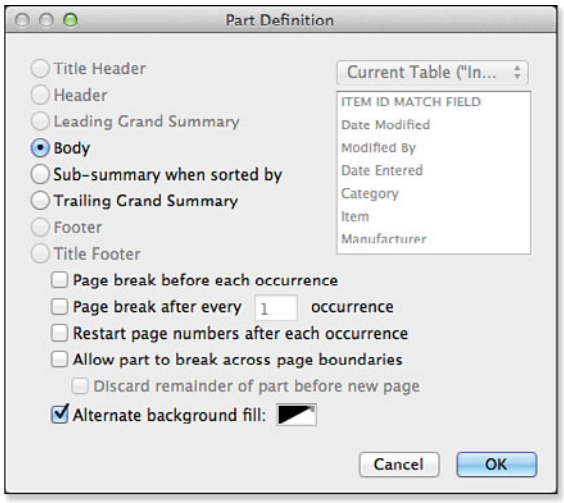
Alternating Row Color

An enhancement you might want to make to a list report is to alternate the row color. The option to alternate row color is found in the Part Definition dialog, shown earlier in Figure 10.21; the quickest way to get there is by double-clicking the body part label while in Layout mode. Figure 10.23 shows the setting for the Inventory layout. As you saw in Figure 10.2 earlier in this chapter, there is a very subtle variation in the background color of each row. (Subtle variations are often the best.)

note

Colors appear differently on paper and on the screen, and they often differ from printer to printer and from display to display. Text on dark colors can be legible on the screen, but on paper can be unreadable. If you are using color for either the row background or the text, make certain that there is a strong contrast. If the row background color will be alternating, the contrast must be strong with both colors.

Figure 10.23
The option to alternate row colors can be applied only to body parts; it is grayed out as an option for any other type of part.



Sorting Data in a Table

One of the easiest methods to use for sorting reports is to teach users how to employ the built-in Sort dialog in FileMaker; however, there are other ways to accomplish the goal.

Sorting with a Click on a Field

Another easy way to sort a set of records in FileMaker Pro is to (Control-click) [right-click] any field and choose one of the three sort options. You don't need to know the name of the field or fret about finding it in a long list of available fields.

Sorting with a Column Title

An interface convention that has been widely adopted by software applications involves clicking the various column headers of a list report to sort the set of records by that column.

The two components of a sortable column header routine are a script (which does the actual sorting) and a graphic indicator to let the user know by which column the list is sorted. You can use whatever graphic indicator you want for this purpose.

The other part of the logic is the indicator for the order of the sorted column. This is a calculation field, the value of which is a container. This calculation relies on a global container with two repeating values: the up arrow image and the down arrow image. (This field can be filled by creating a layout with the field on it, pasting the images in, and then deleting the layout. The values will remain.)

The appropriate image is inserted into the container field through the calculation; note that it is blank if the column is not sorted.

Sorting with a Pop-up Control

In the Inventory Starter Solution, yet another method of sorting is used. As you see in Figure 10.24, a pop-up menu lets you choose the sort order of the report.

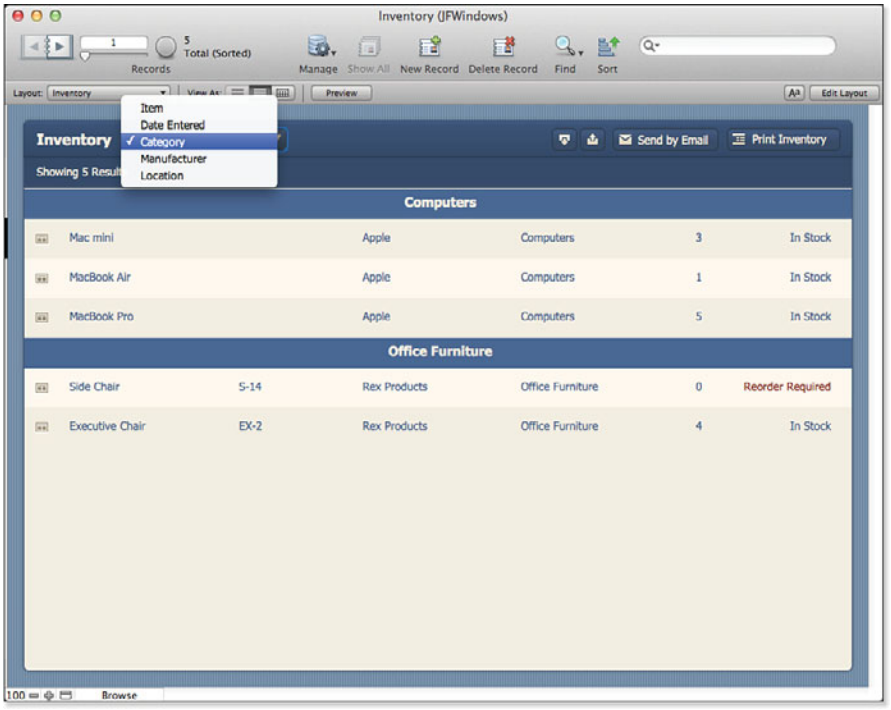


Figure 10.24 Choose the sort order of the report.

As you can see at the top of Figure 10.25, the pop-up menu is based on a value list, and it stores its value in the database in a field called Inventory::Sort Selection.

Write a script to perform the sort as shown in Figure 10.26. Note that the sort criteria for some of the sorts is specified in the specific sort script steps, so they are not visible in the script.

Figure 10.25
Store the sort order in the database.

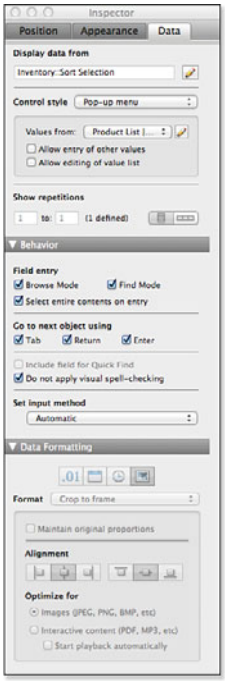
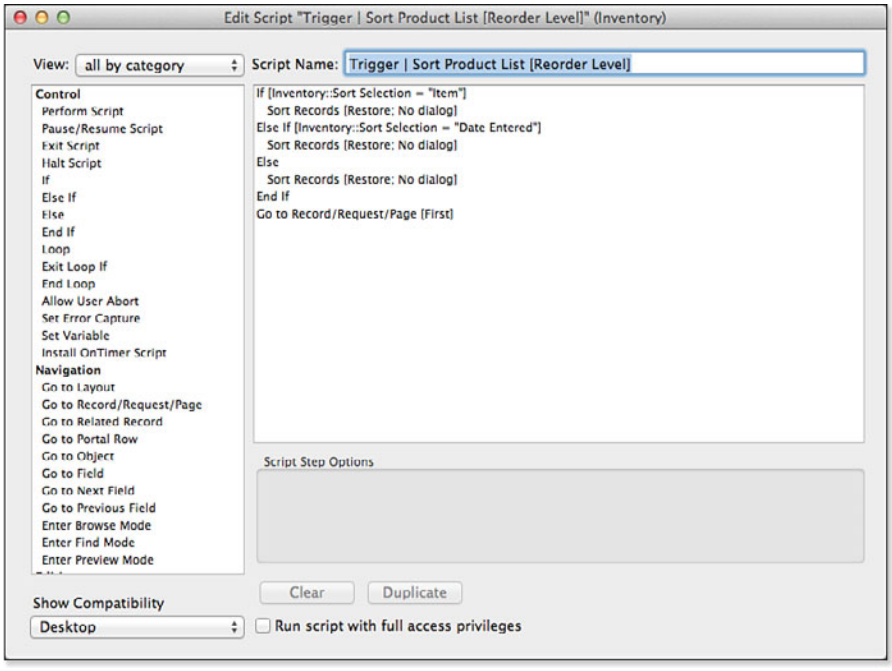


Figure 10.26
Write the sorting script.



Finally, select the field for the pop-up menu control. Choose Format, Set Script Triggers and enable the trigger for OnObjectModify; then attach the script to it as shown in Figure 10.27. Now when a user chooses a sort order, the sort will automatically be performed.

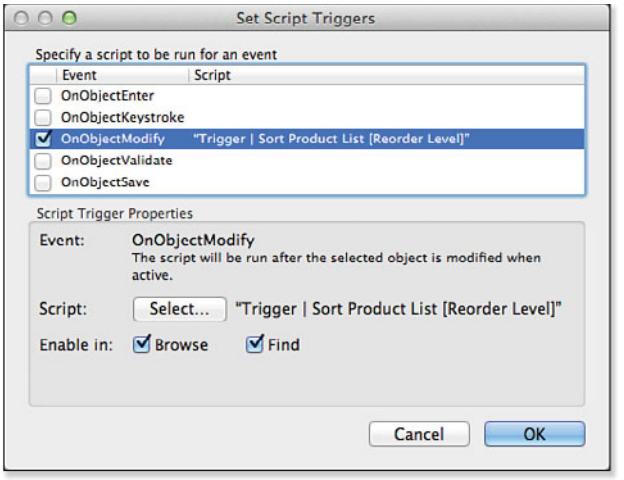


Figure 10.27
Implement the sort with a trigger.

Sliding Objects

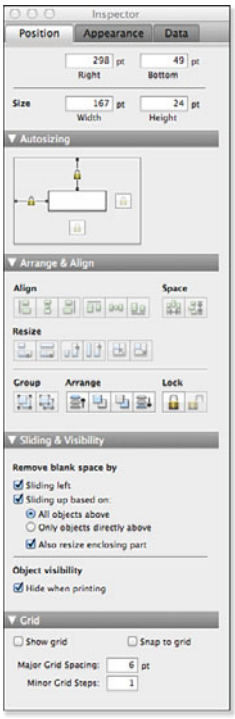
If you are developing layouts that you intend to be printed, and you have variable amounts of text in certain fields, you might want to configure some objects on your layout to slide. Sliding eliminates excess whitespace from an object, allowing it to appear closer to its neighboring objects. You can configure an object to slide either up or to the left, using the controls at the bottom of the Position Inspector shown in Figure 10.28.

The effects of sliding can't be seen unless you are in Preview mode (or you actually print). If you set a field to slide, any whitespace in the field is removed in Preview mode. One caveat to know is that the contents of a field must be top-aligned to slide up and left-aligned to slide left.

Sliding does not reduce the amount of space between objects. Imagine you have a large text field with a horizontal line located 10 pixels below the bottom of the field. If you set both objects to slide up, empty space in the field will be removed, and the line will slide up until it is 10 pixels away from the bottom of the field.

The option titled Also Resize Enclosing Part is useful when you have a list of variable-length records. Set the layout to accommodate the longest possible amount of data and then turn on sliding for all the fields in the body and reduce the size of the enclosing part. The rows of the list will have a variable length when you preview and print them. You must be sure to set all the objects in the list to slide; a single nonsliding object can prevent the part from reducing properly. Objects such as vertical lines do not shrink in size to accommodate variable record widths, so if you need this effect, use left or right field borders, which do shrink appropriately.

Figure 10.28
You can configure an object to slide either up or to the left by using the Sliding & Visibility on the Inspector's Position tab.



Sliding can be applied to portals as well, but objects in a portal can't slide. If a portal is set to slide up, any blank rows of the portal are suppressed, but there's no way to make the height of the individual rows of the portal variable. Portal sliding is useful and necessary in reports that must pull in data from related files. Typically, if there's a portal on a printable report, you should set the portal to display a large number of records and not to have a vertical scroll. If you enable sliding as well, any unneeded portal rows simply disappear.

The Sliding & Visibility section of the Position tab also has an option to hide a layout object when printing. As with sliding, this setting is apparent only in Preview mode. Typically, you use this option to allow buttons, background images, and data-entry instructions—items you typically wouldn't want to have on a printout—to be visible only in Browse and Find modes.

Delivering Reports

Reports are typically among the most important things a database solution produces. Workflows often include using a database for data entry and then running a routine of some sort to have that data synthesized and presented as output in the form of a report. After a report has been generated (usually onscreen in Preview mode), users almost always want to take an additional step and deliver that report to some other medium. Often, delivering a report is as simple as clicking a Print menu option; however, FileMaker Pro provides additional capabilities for distributing reports to various users.

Save/Send as PDF

Available from the File menu is the option titled Save/Send Records as PDF. For the FileMaker family of products, FileMaker, Inc., offers the complete PDF application programming interface; the creation features provide comprehensive control over PDFs generated from FileMaker Pro (see Figure 10.29).

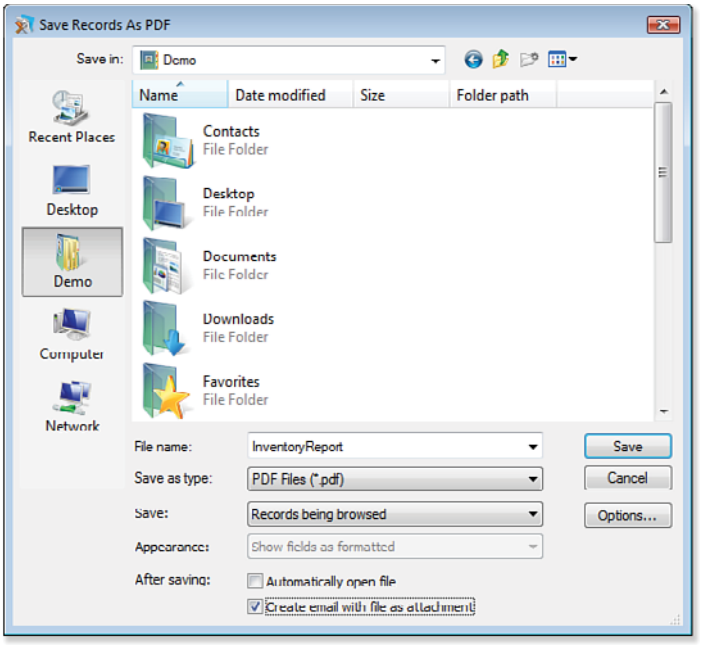


Figure 10.29
From any layout in FileMaker, users can save PDF reports directly from the File menu.

Just as with printing, users can opt to save to PDF a single record, a set of records, or a blank view of their current layout. The end result is a PDF file that can be viewed by anyone with the capability of opening PDF files—virtually everyone with a modern Windows or Mac computer.

The Create Email with File as Attachment option in the dialog where you select the file to send creates a PDF and automatically opens a new email message with the PDF document as an attachment. This one-step process makes it simple to send documents directly from FileMaker.

The Options button at the lower right of Figure 10.29 gives you access to document options as shown in Figure 10.30.

The Security tab lets you control the document's security as shown in Figure 10.31.

Last but certainly not least, notice the Create Email with File as Attachment option at the very bottom of Figure 10.30. When you select this choice, FileMaker creates a PDF and automatically opens a new email message with the PDF document as an attachment. This one-step process makes it simple to send documents directly from FileMaker.

Figure 10.30
Set document metadata for PDF files.

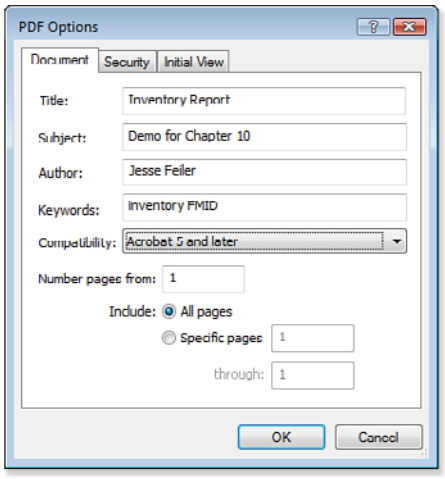
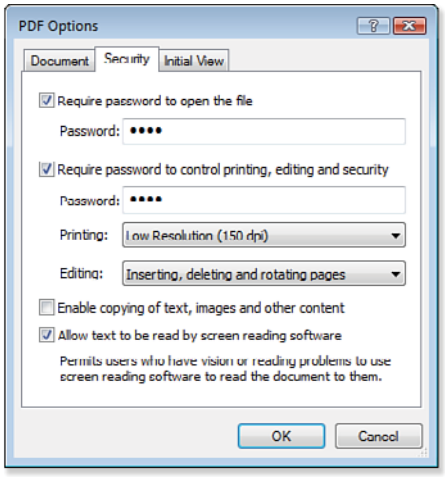


Figure 10.31
Set security for the PDF document.



Save/Send as Excel

Just as FileMaker allows you to export data, users can now save and email Excel documents directly from the File menu. Users don't have to manipulate export dialogs; they simply get whatever data columns are available on their current layout, and the resultant file is a native Excel document. No formatting is available, but the document properties can be set from the Excel Options dialog, as shown in Figure 10.32.

Notice that as with Save/Send Record as PDF, users can opt to create a new email message with the resultant file attached in a single, easy step. Likewise, as with the `Save as PDF` script step, developers can automate the creation of Excel documents by using the `Save As Excel` script step.

In FileMaker Pro, you can save Excel workbooks both in .xls and .xlsx formats.

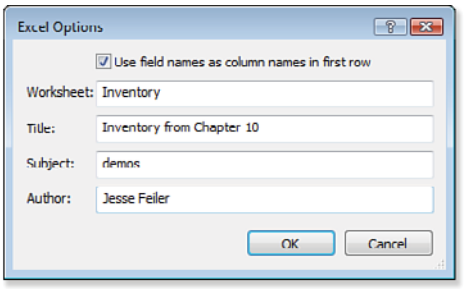


Figure 10.32
Saving documents to Excel directly can save multiple steps and delivers information in a form that is often more familiar to other constituents in an organization.

Send Mail

FileMaker has had the capability to send email via the `Send Mail` script step as well as directly from the File menu without your having to do any scripting or development work.

Note that the Send Mail dialog allows users to pull calculated values from a database and can send multiple emails—one per record in the found set—in a batch process.

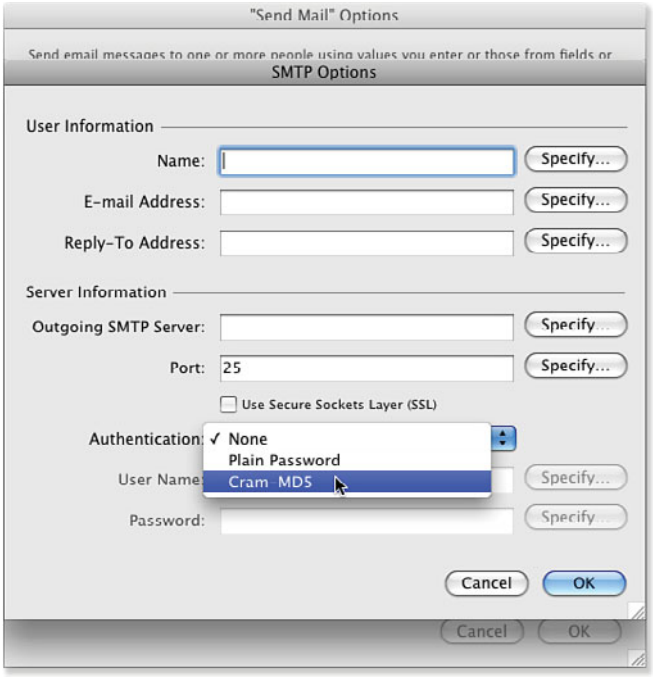
The `Send Mail` script step enables developers to automate batch email processes and can dynamically generate recipient addresses, subject lines, email body text, and more from the records in a given database.

Figure 10.33 shows the options dialog for the Send Mail script step. The pop-up menu at the top lets you choose to send mail from the mail client on the user's computer or through SMTP on a server that you identify.

Figure 10.33
Send email with an email client.

You can use the pop-up menu at the top of the dialog to choose an SMTP client, as you see in Figure 10.34.

Figure 10.34
Send email with an SMTP host.



Notice that each of the fields in the dialogs can be specified with a calculation, thus making the entire process totally customizable.

Scripting Send Mail

This process can be scripted as you can see if you explore the Send by Email button in the Inventory layout shown previously in Figure 10.2. The script that is attached to the button is shown in Figure 10.35.

If you use the SMTP option, it can be scripted, or the user can specify the values as the script runs. This is shown in Figure 10.36.

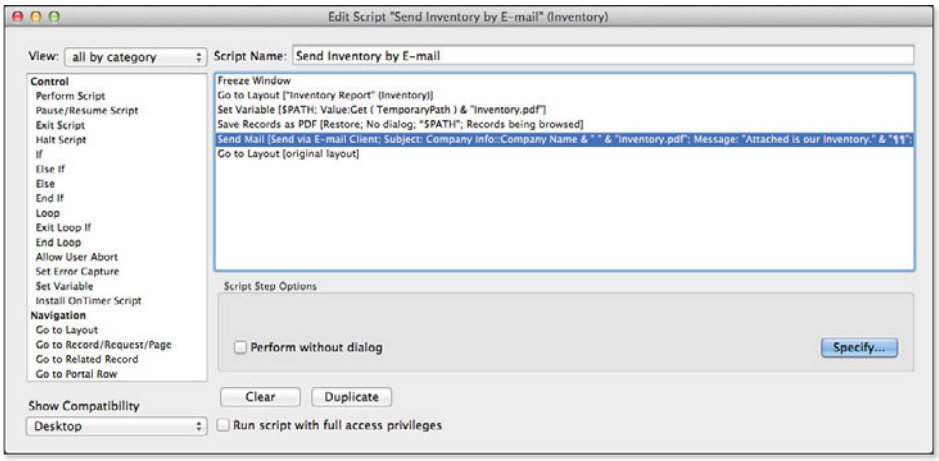


Figure 10.35
You can script a button to produce a report and send it.

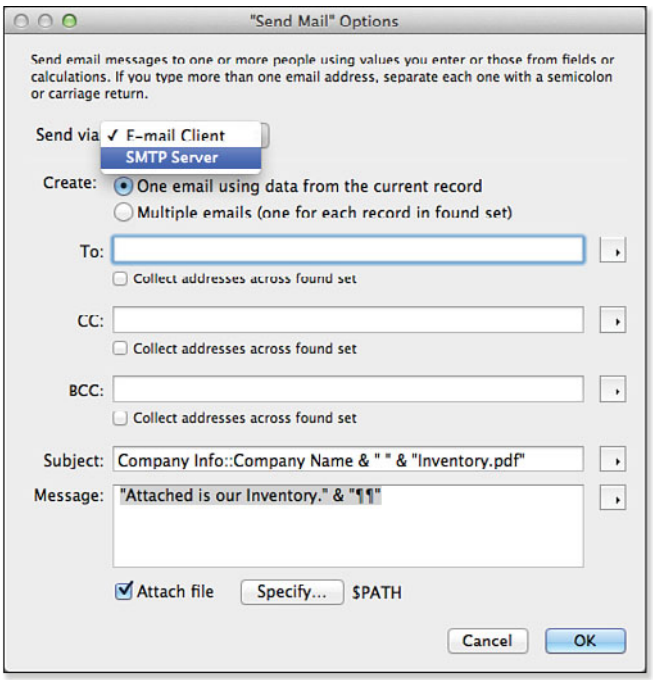


Figure 10.36
Users can enter SMTP values as the script runs.

The end result of the script is shown in Figure 10.37.

*To learn more about scripting routines, **see** Chapter 9, "Getting Started with Scripting," and Chapter 16, "Advanced Scripting Techniques."*

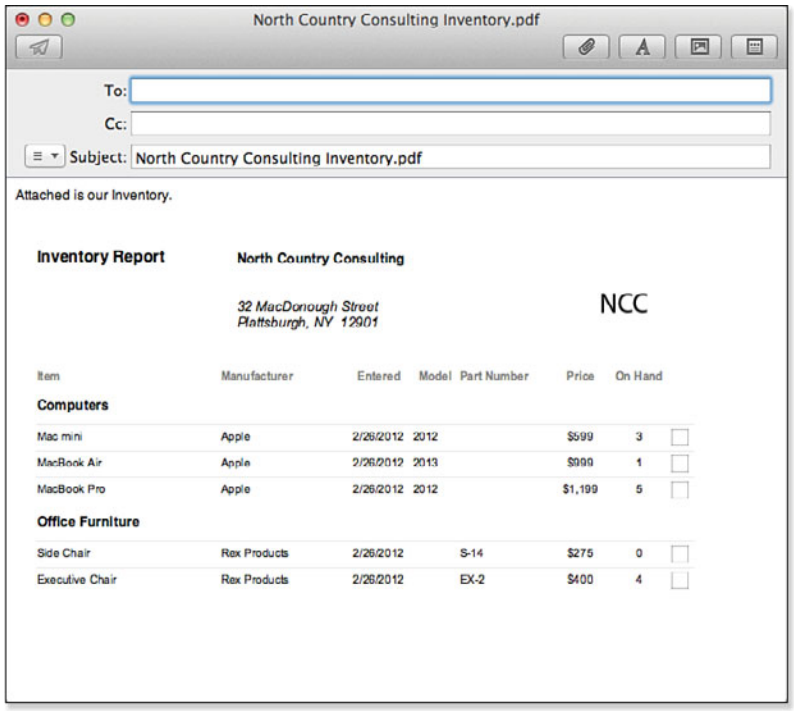


Figure 10.37
View the email message with the PDF.

Introducing Charting

FileMaker Pro includes built-in charting that is very powerful and easy to use. The principle of charting in FileMaker Pro is that you can easily create a chart that is placed in its own layout. From there, you can adjust that chart itself. Alternatively, if the data in the chart changes, the chart is adjusted automatically. The chart on the layout is easily modified if you need to do so. In this section, you will see an introductory overview.

More information about charting can be found in Chapter 14, "Advanced Interface Techniques."

You start by selecting a field in Browse mode to serve as the basis for the chart. Charts often have two data sources (an x-axis and a y-axis), but you normally start with one of them. Get started by selecting the basic field and opening the shortcut menu shown in Figure 10.38.

A basic chart is created for you in its own window, as shown in Figure 10.39.

tip

As always, the shortcut menu is opened with (Control-click) on Mac and with a [right-click] on Windows. Note that charting is available in the shortcut menu from Form, List, and Table views in Browse mode.

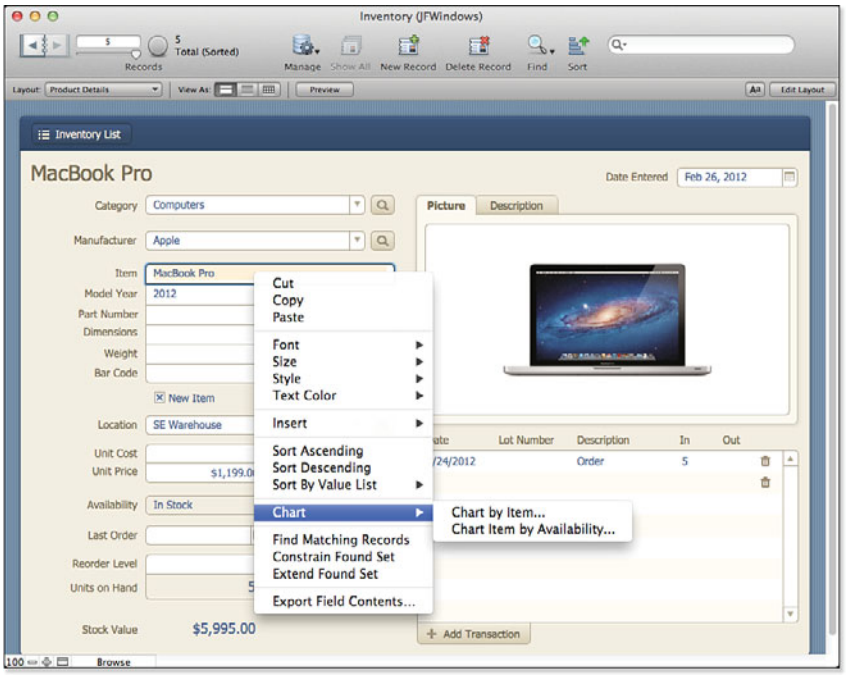


Figure 10.38
Choose a chart from the shortcut menu.

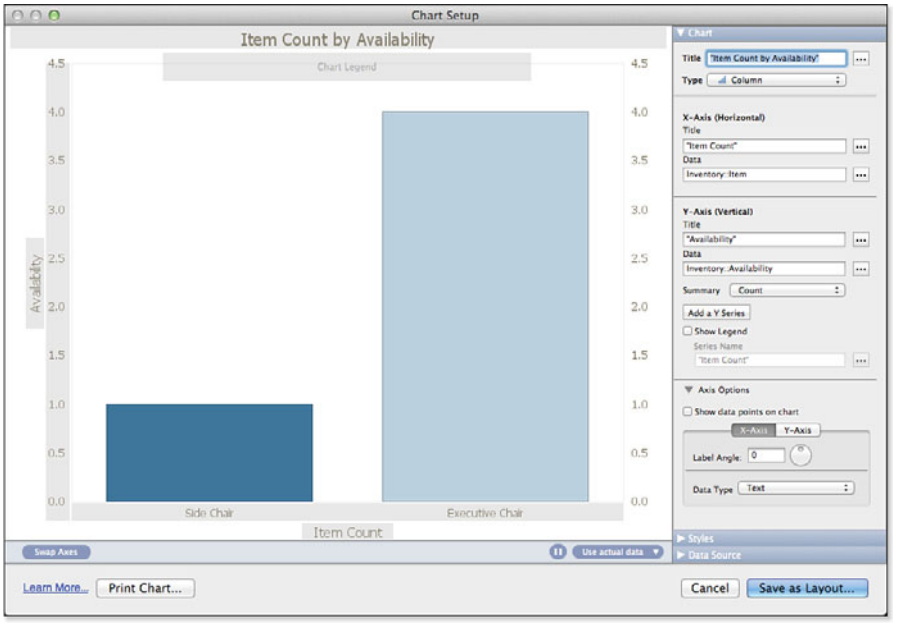


Figure 10.39
FileMaker Pro creates a basic chart.

From here, you can use the Inspector-like tools at the right to adjust the chart. You also can use the buttons at the bottom to make changes to the chart. Perhaps the most useful of these tools is the Swap Axes button at the lower left. This switches the x- and y-axes; often experimenting is the fastest way to come up with the best presentation. Also, at the lower right of the chart area, you can choose to switch between actual data and sample data. Sometimes the sample data is easier to work with, as shown in Figure 10.40. This is because it significantly distinguishes between the chart components; sometimes real data is close in value so that the chart is not as clear as it might be.

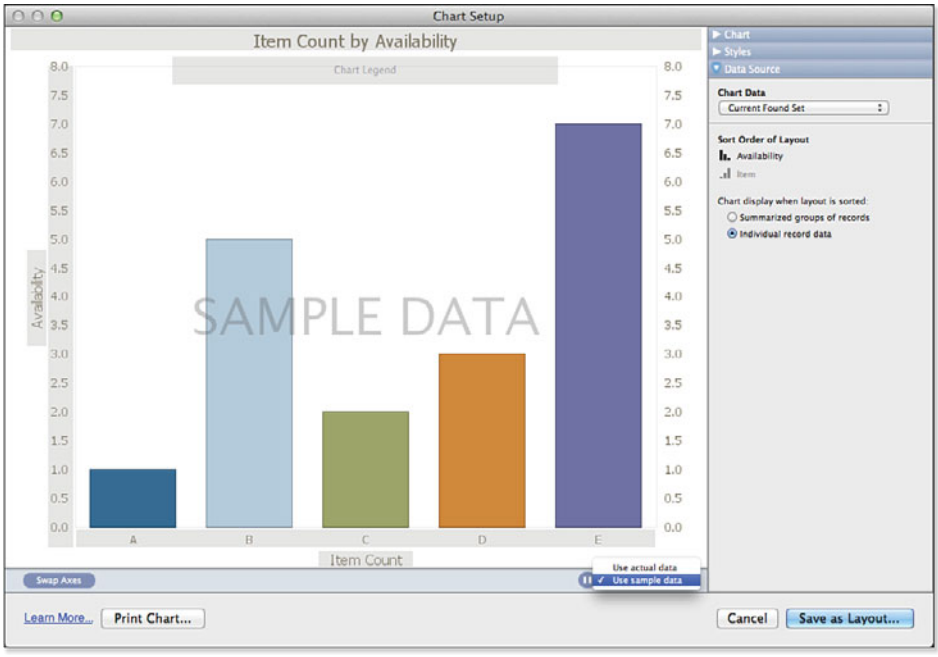


Figure 10.40 Use sample data.

You can also explore the sections of the Inspector at the right to adjust the settings for the chart, styles, and data sources. In the Styles section, for example, you can choose from the chart types shown in Figure 10.41.

When you are satisfied (or want to stop for a break or to rethink things), you can save the chart as a layout with the button in the lower right. Name it as shown in Figure 10.42.

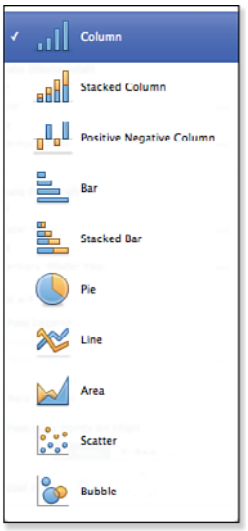


Figure 10.41
Select the chart type.

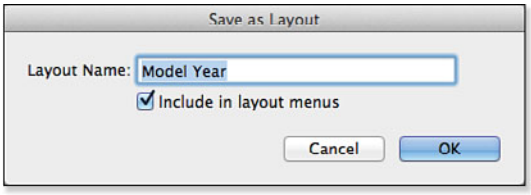


Figure 10.42
Save the chart as a layout.

As you explore the chart, you will probably catch a number of errors. You can always come back to the chart layout, go into Layout view, and adjust them. For example, in Figure 10.43, you can see that, by default, the years are shown as decimal numbers. You probably want to remove the decimal point. A simple way to do that is to use a calculation instead of the raw field.

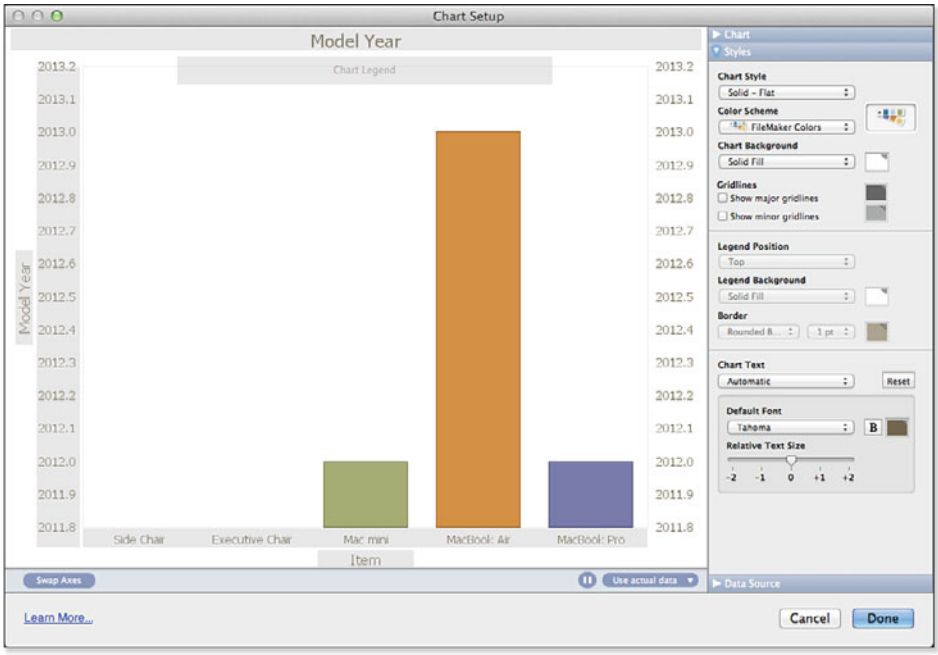


Figure 10.43 Review the chart.

Troubleshooting

Printed Reports Show Only a Single Record

Sometimes, my printed reports contain only the first record of data. Why is that?

Chances are that your print settings are configured to print the current record rather than the current found count. When printing from a List view, be sure to select the Records Being Browsed option. This configuration can be specified within a script, so be sure to set your print scripts to use this configuration as well.

Reports Don't Go to Preview Mode from the Assistant

I just created a new report with the Layout/Report assistant but it didn't end up in Preview mode the way it used to.

FileMaker Pro supports dynamic reports. In previous versions, the last screen of the assistant let you choose to open the report in Preview mode or Layout mode. If you look carefully, you see that the choice is now Browse mode or Layout mode. Here's what FileMaker Pro is doing if you choose the non–Layout mode option. It inserts code in the script it is generating to test if the user is running FileMaker Pro or later. If so, the report opens in Browse mode. If you move to List view, you can manipulate the data and see the subsummary parts update appropriately. You can always manually

go to Preview mode if you want to print it or see the results of paginations. If the user is running an earlier version of FileMaker, the older process (opening in Preview mode) continues. Here's where it gets slightly complicated. In order to keep old scripts running as they have always done, this behavior is only for new scripts created with the assistant. You might want to copy and paste the code from the end of the new script into older scripts so that the behavior is consistent across your scripts. Changing the behavior of your existing scripts is a choice for you to make—not for FileMaker to automatically do behind your back.

FileMaker Extra: Incorporating Reports into the Workflow

The focus of this chapter has been on the creation of list and subsummary report layouts. There's a bit more to creating useful reports, however, than merely setting up nice-looking layouts: You have to incorporate reports into the user workflow, controlling how a user both accesses and exits a report. The methods you choose can vary from solution to solution, and your choice is a function of both what the system does and the particular audience. If the users are proficient with FileMaker, they might be comfortable manually finding and sorting a set of records and navigating to the appropriate layout. More often, however, users benefit from your taking some time to set up some infrastructure to help them access the reports properly.

There are many ways you can go about building reports into the workflow of a solution. Following are some of the most common we've seen over the years:

- **Place buttons to run reports on relevant data-entry layouts**—For instance, on an Invoice Entry screen, you might have buttons for creating an Invoice Aging report; on a Contact Entry layout, there might be a Callback Report and a Contact Activity Report. Users typically are expected to find whatever data they want included in the report; the script simply goes to the correct layout, sorts, and previews, and then potentially returns the user to the original layout.
- **In your report scripts, use custom dialogs to give users certain choices about how the report will be generated**—For instance, a dialog might prompt users as to whether they want to produce a report for the current month's data or the previous month's.
- **You can create a centralized Report Menu layout that can be accessed from anyplace in your solution**—By centralizing your reports, you can avoid having to clutter data-entry layouts with report buttons. In addition, you give your users one place to go any time they want a report, rather than requiring that they memorize which reports they can generate where. A centralized report menu works well when the report scripts run predetermined finds.
- **As a variation on the Report Menu concept, you can give users control over finding and sorting the data**—You can, for example, place global fields on a layout so that the user can enter a date range on which to search. The find criteria are usually specific to a certain report or group of reports, so you need to branch to the appropriate "finder" layout when a user makes a selection from the report menu.

- **A third variation on the Report Menu idea is to literally create a Reports custom menu**—A custom menu of reports could offer contextual listings of available reports from a given area of your database, or it might simply offer all the reports available within your solution.
- **You can enable users to modify the title of a report or to add a secondary header of their own choice**—This typically is done with custom dialogs, but you can also incorporate this element into a report menu or layout dedicated to preparing records sets for reports.

After generating the report, you'll probably want to return users to wherever they were before running the report. Try to avoid a situation in which a user is stranded on a report layout without any tools to get back to familiar territory.

You should also strive to have some consistency in how reports look and function in your system; this will make using them easier and more intuitive for your users. For instance, you might set up as a convention that reports are always (or never) previewed onscreen, and users are prompted as to whether they want to print a report. Similarly, place layout elements such as the title, page number, and report date and time in consistent locations on your reports so that users don't have to hunt for them.

This page intentionally left blank

11

DEVELOPING FOR MULTIUSER DEPLOYMENT

Developing for Multiple Users

Some of the best, most lovingly developed FileMaker Pro systems are only ever used by a single person. Then there are the rest of the databases out there. FileMaker Pro enjoys a graceful growth curve from single-user applications to systems that support enterprise-level workgroups and operations of hundreds of users.

We can be thankful that this graceful transition from single user to multiuser means that issues to take into consideration when building multiuser systems are reasonably modest. Much of what you already know about building FileMaker Pro systems—regardless of your planned deployment—also applies directly to building a multiuser application.

In this chapter we cover two primary topics: how the FileMaker engine handles multiple users and the development techniques you need to consider when building multiuser applications. As a third discussion, we also go into some depth about audit trails, given that they often are used to help ensure data integrity in systems used by larger organizations and are used specifically to track multiple-user scenarios.

We recommend that anyone intending to deploy a system to multiple users read this chapter. Some of the issues we discuss become necessary considerations only in systems getting heavy use from multiple users, but they're good to have in mind nonetheless.

note

Multiuser used to be a simple concept: several users access the solution at the same time. Today, the concept needs to be considered in a broader manner. Although only one user might be involved, a solution that runs on FileMaker Pro in someone's office and on FileMaker Go when that person is, well, on the go, can be considered a multiuser solution. It is not the identity of the user that makes it multiuser but rather that more than one person may be accessing the system at the same time. Whether it is one person on several devices or several people, the concepts and concerns described in this chapter are the same.

It is also worth noting that, in this chapter, FileMaker is used to refer to both FileMaker Go and FileMaker Pro (and FileMaker Pro Advanced) because, in a multiuser environment, they function in the same basic manner.

This chapter is a good companion to Chapter 26, "Deploying and Extending FileMaker."

*To grasp the IT infrastructural logistics of hosting a FileMaker Pro solution, **see** Chapter 27, "FileMaker Server and Server Advanced."*

Sessions in FileMaker Pro

FileMaker Pro is a client/server application (at least when files are hosted by an individual user or by FileMaker Server). Each time someone using FileMaker Pro or FileMaker Go (a client) connects to FileMaker Server (or, using peer-to-peer sharing, connects to a copy of FileMaker Pro sharing a database) and opens an instance of the database hosted there, that person creates a *session*.

In practical terms, this means that one of your users can be on layout #10 while you yourself are working with layout #2. You can run a script, and nothing will necessarily happen on another user's computer; likewise, someone else can export data on a computer while you're performing a find request in the same database table on yours. You each have a separate connection to the database, with its own unique environment. While working with the same data, all your users can be performing separate, distinct tasks in your system. Each user can have a separate view of the database, with different active windows, active tables, or active found sets, among other things.

Generally, these individual user sessions don't interfere with each other at all; however, in some cases they can conflict—for example, when two users try to edit the same record at the same time. Throughout this chapter we cover various techniques for

tip

Remember that sessions apply to connections to hosted files—that is, connections opened with File, Open Remote rather than File, Open. Direct connections created with the File, Open command may use sessions internally, but the concept is not visible to users. On FileMaker Go, sessions come into play when you connect to a FileMaker database on a server. When you open a database located on your mobile device, that is analogous to using the File, Open command on FileMaker Pro.

note

The one thing that *is* consistent across all user sessions is the actual data in the database. Changes you make to records you are editing are immediately visible to other users in the system, and vice versa. Our discussion of sessions pertains only to global fields and variables, window states, and layouts. Actual data is stored and displayed consistently for everyone.

identifying and coping with such issues, although most of the work is already done for you inside FileMaker.

Before we approach ways to manage sessions and potential conflicts, it is important to understand what a session is and how FileMaker manages multiple users. In FileMaker, sessions are implicit and enjoy a stateful, persistent, always-on connection to the server. The system preserves and isolates each user experience in the FileMaker client. Keep in mind that after the session is over (an individual user closes the database), all information about that session—what layout was in use, where windows were positioned, what the found set was—is discarded. The next time that user opens the database in question, it opens in its default state, with no preservation of how the user last left the system.

tip

You can use a trigger to run a script when the last window of a file is closed (that is, when the file is closed). That script can capture and store the user's layout and choices so that they can be restored when the file is opened again.

You might have heard the term *session* as applied to the Web. FileMaker is quite different. On the Web, connections are stateless by default; they have no memory. The web server does not maintain a connection to a user; the effect of a persistent session is approximated by the explicit creation of an identifier for a given user when she logs in to a system. That identifier is then passed (and often stored and retrieved via a cookie) through all the page requests a person may make in a given time period. Web developers need to explicitly create the mechanics of a session to preserve a user's experience from page to page. Whenever you buy a book from Amazon, the developers there have no doubt labored to make sure that each page you visit tracks sensibly your use of the site—especially when it comes to the multipage shopping cart experience. FileMaker, by contrast, provides persistent database sessions with no additional effort by you, the developer.

Session-Specific Elements

FileMaker's sessions maintain a consistent user experience until the application itself is closed. This experience includes your login account (unless you explicitly log out and log back in), the position and number of windows you have open, which layouts you're on, your current found set, your current sort order, and the portal scroll positions. On the development side of things, custom colors you've stored in the layout tools are, unfortunately, lost at the end of a session as well. And, as you might expect, global variables (which are not stored in the database) are session specific.

Global Behavior

Globals (fields specified as having global storage as well as global script variables) are session specific and require additional discussion. In a multiuser client session, they utilize and hold values unique to one specific user's session. This enables you, as a developer, to depend on globals storing different information for each user.

*For more details on global field storage, **see** "Storage and Indexing," **p. 104**.*

At the start of a session, each global field is initialized to the last value it had in single-user mode. If you run only in single-user mode, this makes the global field value appear to persist across sessions,

but it's misleading to infer that there are multiuser and single-user types of sessions. Storing information in global fields for single users is a handy way to leave things the way they were, but it also allows developers to create a default state for global fields.

Global fields are used for a range of functions in multiuser databases: They often hold images for navigation and user interface purposes, and they sometimes hold session information such as the current date or the active, logged-in user. It makes sense, then, that they would be specific to a given user's experience.

*If your global fields suddenly seem to be holding wrong data, **see** "Unpredictable Global Default Values" in the "Troubleshooting" section at the end of this chapter.*

Global variables, on the other hand, do not have stored values from session to session in single-user or multiuser mode. As a developer, you have to explicitly initialize the variables you intend to have the system utilize, ideally at the beginning of each session.

*For a complete discussion of script variables, **see** "Script Variables," **p. 453**.*

User Accounts and Session Data

One common use of global behavior in a multiuser environment is to set a global field with your currently logged-in account. This enables you always to have a central stored value that's easy to use in calculation formulas and scripts. One could argue that simply using the `Get (AccountName)` function wherever necessary would accomplish the same end, but there's an additional use for storing the current account name in a global: You can drive a relationship with it into a user table by using the account name as a unique match field.

note

Note that a global variable, as opposed to a global field, cannot drive a relationship. This difference can play a significant role in determining whether to store particular session data in a global field or a global variable.

By using the account name this way, you can tie account information to data. You might want to do this if, for example, you need to store someone's real name, her preference always to start on a specific layout when the system opens, or in what language she wants to use your database.

tip

If you are storing user settings and preferences, consider which of them should be restored when the user reopens the database file. Storing open window locations is relevant only to FileMaker Pro (there is only a single window on FileMaker Go).

All these examples depend on your having done something with the information you store in a user table. It's useful to store someone's preference for a starting layout only if you then write the requisite script that uses this as a reference.

Another possibility lies with tracking database use. Although you might debate whether a database or database administrator should be looking over someone's shoulder, you could write routines that post records to a user log table whenever users log in, log out, or even when they perform certain scripted actions (delete records, create records, run an invoice report, and so on).

One more user-friendly option is to accommodate users simply by enabling them to specify where they prefer a window to be positioned and sized. All these various options can be enabled by storing information specific to a single person's session in global fields.

Concurrency

You might have heard the term *concurrency* as it relates to databases. It refers to the logic and behavior of database systems when two (or more) users attempt to interact with the same information. A simple metaphor might be two people trying to use a phone book or dictionary at once; they're likely to trip over each other a bit. Every multiuser database platform has to address this issue. Certainly, the easiest solution would be simply to restrict using the database to one user or function at a time, but clearly that's unrealistic.

The ACID Test

To address issues of concurrency and transaction integrity, database engineers have developed what has come to be known as the *ACID test*. Database software needs to pass this test to manage concurrency issues completely. ACID stands for *atomicity*, *consistency*, *isolation*, and *durability*; these four terms describe the features and requirements for processing transactions in a database system. If a system does not meet these requirements, the integrity of the database—and its data—cannot be perfectly guaranteed.

In the context of databases, the term *transaction* relates to a single logical operation comprising one or more steps that results in data being posted to the system. Examples might include committing a new record to the database, performing a script that calculates summary information, or, in real-world terms, completing the multiple steps of debiting one financial account and crediting another. The ACID test exists to ensure such transactions are reliable.

FileMaker databases, on their own, do not fully meet ACID compliance, nor is it realistic to develop a solution in FileMaker that perfectly does. FileMaker scripts can be interrupted (a machine crash or a force-quit of the application), and as such it is possible to leave a transaction half completed. Rolling back a half-completed operation is entirely feasible in FileMaker, but if it is necessary, you must implement it yourself.

We're including this section not to point out a shortcoming of FileMaker, but rather to illustrate some important guidelines on how you should consider building solutions for critical business systems or large workgroups. It is possible to go a long way toward ACID compliance in a FileMaker Pro database—if it's properly engineered. It's also quite possible to build a FileMaker Pro database that leaves wide opportunity for data integrity problems to crop up (as with any other database tool).

Here are the components of the ACID test:

- **Atomicity**—Atomicity requires that transactions be completed either in their entirety or not at all. In other words, a logical routine (say, crediting one account and debiting another) cannot be left half done. In FileMaker terms, data is either committed or not committed to your database, a script needs to reach its logical conclusion, and a calculation function stores and indexes its results properly. Although a script can be interrupted, it

note

As consultants, we're pragmatists. Often the craftsman in all of us yearns to build the world's most perfect system, but in reality there are trade-offs in complexity, time, and flexibility to consider. We use the guidelines that follow as just that—guidelines. By identifying the criticality of certain data and using sensible safeguards to ensure its integrity to the degree possible, we are able to cover all but the most extreme cases of database failures.

is important to approach atomicity by writing scripts that conclude whatever routines they're designed for. In the case of complex processes that might be carried out by several scripts, having a master scripts that manages the subscripts can help you make certain the process is fully completed or rolled back properly.

- **Consistency**—Consistency ensures that your database is left in a legal state at the beginning and end of any given transaction. This means that the transaction won't break any of the rules, or integrity constraints, of the system. This often can encompass business logic: An example might be that all financial credit transactions be positive numbers.
- **Isolation**—Transactions in mid-process are never exposed to other processes or users. In the credit/debit example, a user should never see a credit appear on one account before the debit has been posted. Likewise, an account balance report should not be allowed to run when a credit or debit is in the midst of being added.
- **Durability**—After a transaction has been performed and completed, the information resulting from that process needs to be persistent. It should be saved with the database, and if someone pulls that computer's plug, the information is still present in the file.

ACID compliance is a goal of development to ensure data integrity. We encourage you, especially when writing scripts, to focus on delivering on these guidelines to an appropriate degree, especially in a multiuser environment.

tip

Even though FileMaker does not provide a mechanism for rolling back partially completed operations, your design can often use a simple technique to achieve almost the same goal. It is the technique often used to update websites. New or updated pages are created, and then, as the last step, a link from a landing page or other known location is provided to the new or updated pages.

Likewise, in a FileMaker solution, take care to make the last operation whatever it is that reveals all the other components of the transaction. You will have a structure in which, if the transaction fails in the middle, there might be some orphan or incomplete records, but they will not be visible because the main link or the main record that will point to the detail records has not been updated. This is not always possible, but, if it is, it can make your FileMaker solution more robust.

And you can also use one of the oldest techniques in the book to prevent some interrupted transactions. Make certain that all non-battery-powered computers are connected to uninterruptible power supply (UPS) systems.

Script Log

One technique we use for verifying processes and debugging is a *script log*. By building one, you better approach atomicity and are able to identify cases where it fails.

In large, complex solutions where transaction integrity is vital, it might be warranted to create a process that causes all scripts to write log records to a separate table (often in a separate file as well) when they start and again when they are successfully completed. It's possible to track other

data as well: who initiated the script, on what layout the user was, which instance of a window was in use, timestamp data for start and end (for performance and troubleshooting purposes), and potentially any data the script manipulates.

By adding a script log to your system and periodically checking it for incomplete conclusions, you can identify cases where scripts fail and then manually address such issues when necessary. By definition, if a script log start entry doesn't have a corresponding close entry, it failed ACID's atomicity test and possibly the consistency test as well.

note

This script log is not to be confused with an audit trail, covered later in the chapter. Audit trails enable you to record all data transactions in a database. A script log is a means of confirming that your functional routines are completed properly.

Commit Versus Create and Serial IDs

In FileMaker, data is committed (saved) after a user exits the record, either by clicking outside a field or by performing a range of other actions such as running a script, changing modes, changing layouts, or pressing a "record-entry" key. The default is the Enter key, but field behaviors can be changed to allow the Return and Tab keys as well.

*For more details on field behaviors, **see** "Controlling Field Behavior," **p. 139**.*

tip

One final note on script logs: We encourage you to create a global variable that, when turned off, disables all script logging in your system. This is one of the few examples in which a global variable (rather than a local one) is a good idea.

It is possible to use the Records, Revert Records option to undo the creation of a record. Until a record has been committed, it exists in a temporary state, not yet visible to other users of the system. Relying on a transaction remaining unsaved until expressly committed helps ensure better ACID compliance. This point is important to remember in a multiuser environment where you might be operating on assumptions established with prior versions of FileMaker. For example, if you're attempting to serially number certain records and two users create two records at the same time, it is possible that one will commit the record in an order different from that in which the records were initially created. It is also possible that a user will undo his or her changes with a Revert Record command and leave you with a gap in your serialization.

In the case of auto-entry serial values, FileMaker enables you to specify when the serial number is incremented: on creation or on commit. This enables you to control auto-enter serialization; however, it does not protect you from other assumptions. For example, if you're relying on `GetSummary()` calculation fields to keep track of an incremented total, remember that the calculations that control this are evaluated and displayed only after a record is committed.

Record Locking

Just as a record is not saved to your database until it is committed—maintaining an isolated state while you create new records—FileMaker does not allow editing of a record by more than one person at a time. In this way, FileMaker meets the isolation test of ACID for posting data. Record locking exists to ensure that no two edits collide with each other (such as when multiple users attempt to edit the same record simultaneously).

After a user begins editing a record, FileMaker locks that record from other users and script processes and (when not captured and suppressed by a script) presents users with an error message if they attempt to enter or change any data in that record.

It's possible to place your cursor in a field and still leave the record unlocked (safe for other users to enter data into the same record), but at the point when you actively begin typing, that record essentially becomes yours until you either commit or revert it.

Locking applies to related records in portals as well. If you are modifying a record in a portal row, that record's parent is also locked. This behavior occurs only when the related child record is edited via a portal or related field from the context of a parent record. If you are simply editing the child record on its own table-specific layout (within its own context), just that single child record is locked.

Also keep in mind that record locking applies only to editing. You can still find locked records, view reports with them included, change sort orders with locked records in your found set, and even export data. Only editing is protected.

If another user is editing a record and you try to edit it, you will receive the message shown on the left in Figure 11.1. If you choose, you can click Send Message to type a message that will be sent to the other user; when it is received, it will appear as on the right in Figure 11.1.

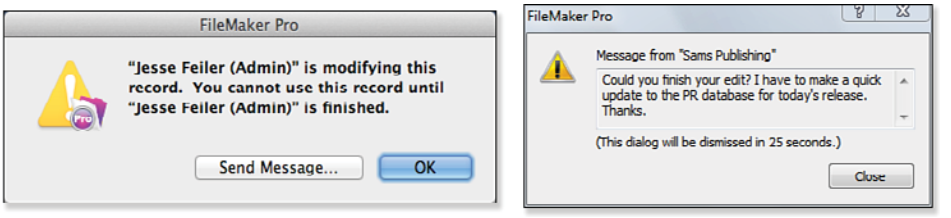


Figure 11.1 You see this message if you try to edit a record someone else is modifying. If necessary, use the Send Message command to ask for control.

See Chapter 17, "Working with FileMaker Triggers," to learn how you can use an idle handler to deal with this issue.

To help with multiuser account testing, see "Use `Re-Login` *for Testing Access and Sessions" in the "Troubleshooting" section at the end of this chapter.*

Trapping for Record Locking in Scripts

A subtle way your database might prove error prone is in always making the assumption in scripts that the routine in question

tip

The one downside to record locking is that you cannot force a user out of a record remotely through FileMaker Pro. If someone begins editing a record and then goes to lunch, you need to kick him off by using FileMaker Server's Admin Console, shutting down the file, restarting the server, or addressing the issue at the user's local computer.

has access to all the records in the current found set. Some of the records your script needs to work with might in fact be locked.

A script can explicitly open a record for editing with the `Open Record/Request` script step. After that script command has been issued, the record is reserved for that routine, and other users who try to edit the record get a record lock error until the script (or the user running the script) releases the record. Because any attempt to modify a record results in the same condition, explicitly using an `Open Record/Request` script step might not be technically necessary, but we find it helpful to turn to for clarity within scripts. The more important step is deliberately checking to see whether a given record is open for editing or if some other user (or routine) has it locked.

To capture the error that results in cases where either one's current privileges don't allow editing of the record in question or the record is locked by another user, we recommend testing first to see whether a record can be opened. If that doesn't work, deal with the result prior to attempting an edit. Use the `Open Record/Request` script step followed by a `Get(LastError)` check. Here's how it might look:

```
Set Error Capture [On]
Open Record/Request
Set Variable [$$error; Get (LastError)]
If[$$error <> 0]
    Show Custom Dialog ["Error"; fnErrorMessage ( "recordLock" )]
   // or write an error handler process here...
End If
//Execute your "real" script here...
//and don't forget to commit your record at the end.
```

Use a `Commit Record/Request` script step at the end of your script to release the record back into nonedit mode and unlock it for other users.

Instead of checking simply for a nonzero error, you could also write a series of `If -> Else If` script steps checking for errors such as 301 (Record is in use by another user), 303 (Database schema is in use by another user), and so on.

note

Consider building error utility tables, or perhaps using custom functions, for error handling. This enables you to easily tailor error messages in a central, easy-to-edit location based on whatever value is held in `$$error`. The `Custom Dialog` step in the preceding code snippet references a custom function that presumably returns error-handling text to the user.

Multiwindow Locking

Multiwindow locking is closely related to multiuser record locking. It is possible to open a new window (via the Window, New Window menu command), begin editing a record there, and in so doing, lock yourself out of editing the same record in your original window. If you are actively editing a record that has yet to be committed and you try to edit the same record in another window, you'll see this error message: `This record cannot be modified in this window because it is already being modified in a different window`. FileMaker tries to ensure that you're not losing data or edits you're in the midst of creating.

The point here is that a user can lock himself out of a record. He might not realize that he has left a record in an edit state before moving on to a new window. The simple answer is simply not to

try to edit a record in two places at once. A user would have to go a bit out of his way to encounter this problem. If you've scripted routines for creating new windows with a script, you might want to include a `Commit Record/Request` step before opening the new window.

Given the fact that window locking so closely resembles multiuser record locking, testing a solution with multiple windows is an effective and efficient way to ensure that your scripts manage record-locking checks properly, without having to resort to using two computers.

Launch Files

One of the challenges users on a network have is actually finding the specific FileMaker files they need to use. This is a no-brainer if you have only one FileMaker Pro solution with a single file, but over time your Hosts dialog can become quite crowded in multiuser situations. In large organizations or companies with many different FileMaker files, a server's file list can be a bit daunting.

note

We generally put a solution logo and a `system loading... please wait...` message on the single layout.

To offer a solution to this simple problem, we often build *launch files*. These utility files are distributed as single-user files and sit on each individual person's computer. They have generally one layout and one script that calls an `open` routine in a network file.

Although you might be tempted to put other niceties in these launch files—the capability to load clusters of files or perhaps some sense of acknowledging the individual user logging in—we encourage you to leave things as simple as possible. You'll have dozens of these files distributed on your network with no easy means of replacing them with upgrades. The simpler you keep them, the easier they will be to maintain.

A final nice touch on launch files is that they close themselves after launching the system in question. They're no longer needed and shouldn't have to clutter the Windows menu.

Troubleshooting

Unpredictable Global Default Values

I have global fields, used for holding system settings, that have been working perfectly for weeks, but today suddenly they have different data in them. What happened?

It's likely they got reset by some script modification you've recently made, or when you had files in an offline, single-user state. In our practice, we find it difficult to remember to set globals for default states in single-user mode through the course of developing and maintaining a system. This is a common source of bugs, and we've learned over the years not to make any assumptions about global values; it's better simply to set them explicitly within a startup script. It's also important to either explicitly set or test for values at the beginning of a script that depends on them.

Use `Re-Login` for Testing Access and Sessions

One of my users is reporting a problem that I don't see when I'm logged in. I'm getting sick of having to re-log in time and again to test this. Is there an easier way to test this?

If you're having trouble testing how other users, with different access levels, might be interacting with your system, write a re-login script that enables you to hop into another account at the click of your mouse. You can even store passwords when using the `Re-Login` step. Connect it to a convenient button or place it in the Scripts menu, and you have one-click account switching.

Another approach might be to create a "debugging" custom menu (with the various login scripts available) and disable the menu before deploying the system.

Making Sure That Your Auto-Entry Always Edits

My auto-entry function worked the first time I edited a field, but then it got stuck and won't update again. What setting is the likely culprit?

If the auto-entry field for your audit log isn't updating—it updates once but then never again—make sure that you uncheck the Do Not Replace Existing Value for Field (If Any) option. It is always checked by default and is easy to miss.

Likewise, the Audit Log routine we described depends on there being data in the field to begin with. Either seed it with something (we use `Creation TimeStamp`) or turn off the Do Not Evaluate If All Referenced Fields Are Empty option. It, too, is enabled by default.

Trapping for Errors

I need to tighten my scripts and don't want to have to code for every exception under the sun. What's the best approach to trapping for errors?

Trapping for errors is always a smart development practice. Get into the habit, and you'll save yourself years of your life debugging. A simple approach is to simply use the `Get(LastError)` function and use a `Case` or `If / If Else` routine to display meaningful messages and logic branches to your users. You can trap for either explicit errors or just a nonzero number.

A better way to abstract your code and provide yourself with a central place to reuse error handling is to simply write an error routine once and be done with it.

There are two ways to manage error messaging. You can either set up your own errorCodes table or build a custom function. Setting up a table is simple and allows you to add your own custom error conditions and messages. You can do this as well with a custom function. The idea is simple: Establish a global `gError` field in your main system and relate that to an `errorID` in your error table. You can also use a `$$error` global variable and have a custom function reference it.

FileMaker Extra: Development with a Team

Sometimes systems are big enough that they warrant multiple developers in addition to multiple users. Developing as a team can be a bit complex with FileMaker Pro, but one of the best (and often unsung) features of FileMaker is that database schema changes can be made while the database is live, on a server, as other users are in the system. This capability is an extraordinary boon for FileMaker developers and will make a real difference in all of our lives.

The idea is simple: Set up a server (far better than multiuser peer-to-peer hosting) and have as many developers as a given system needs to work together.

It's important to keep a few points in mind: Only one person can adjust the schema in a given file at a time. This is true for editing scripts as well. If another developer is working in ScriptMaker, you can view scripts there, but you will be unable to make changes or add new scripts until your teammate finishes. This means you can have one person focused on scripting, one defining a new calculation field, and a handful of others working on different layouts all at once. One way to avoid conflicts in this regard is to split your solution into multiple files and have those files reference external table occurrences as needed.

Over the years we've assembled some best practices for working on a team. Here's a list of techniques we draw on:

- **Use FileMaker Server**—Server (as opposed to simply working peer-to-peer) allows you to run frequent backups, and if any one machine crashes, the files are still protected from the crash.
- **Use FileMaker Pro Advanced**—The Script Debugger is handy to use in the multideveloper environment, and the Data Viewer is an invaluable tool as well. When another developer is editing scripts and you can't open a script in ScriptMaker, turn on the debugging tool and you'll at least be able to see the script in question.
- **Use custom functions**—Custom functions can be written while other programming activities are underway, and they provide a deep layer of possible abstraction. It's possible to have multiple developers building custom functions while others work in the core system, and it's also a great way to reuse code across a team.
- **Set up a bug-tracking database**—If you're working on a multiuser system, then testing, requests, random ideas, and other communication are vital. You've got some of the world's best database software at your fingertips; put it to use and build a bug-tracking system for your development team and your users.
- **Build re-login scripts, toggle status area scripts, and developer layouts**—Giving developers access to the back stage area of a system is vital. Build scripts to get them there.
- **Assign a chief architect**—With creating a meal, having too many cooks in the kitchen spoils the broth. Similarly, one person should ultimately be responsible for the overall technical directions the system requires.

- **Comment**—Comment. Comment. Comment. Document what you intend to do and what you have done in your scripts and field definitions. It is also useful to add comments to scripts about what has been removed or found not to work so that someone does not come along a year later and repeat a mistake.
- **Join TechNet and FileMaker Developer Subscription**—The TechNet program at FileMaker provides access to webinars and discussion groups. It is free. The $99/year FileMaker Developer Subscription provides as one of its benefits a development license for FileMaker Server and FileMaker Server Advanced. These licenses are limited to three simultaneous users, but that is sufficient for testing. You also may have access to pre-release software and briefings on future directions. Find both at www.filemaker.com/technet/.

*For more information on TechNet, **see** www.filemaker.com/technet/index.html.*

This page intentionally left blank

12

IMPLEMENTING SECURITY

Approaching Security

Security is a primary concern for all database developers and a significant factor in an organization's requirements for both the internal workings of a database system and the technology used to build it. IT departments in particular pay close attention to security issues and often have specific needs that go beyond those of the users of your database solution.

FileMaker's security architecture was completely overhauled in the FileMaker 7 product line, and it offers a robust set of features for managing security. It meets the common standards for security and account administration most IT organizations require of modern server-based technologies.

Regardless of how you plan to deploy a solution (you might not even have an IT department), we strongly urge all developers to learn about security and choose appropriate levels of safeguards for their FileMaker solutions. This might be as simple as locking down the capability to modify the database schema or as complex as deploying your solution on a network with ties to an external authentication server. Whatever your specific needs for security, three primary concerns bear consideration:

- **Physical access**—The first issue for security is making sure that you protect the database file itself. No matter how robust a security architecture is for any kind of software or server application, you will face risks if a malevolent person gets direct access to your database file or server. You have to protect your backups, including offsite copies, just as much as your live database.

- **Network access**—The second area for security is the network traffic between a FileMaker hosting computer and the client computers connected to it. If you are working on an open network, you might want to consider encrypting the data stream between FileMaker Server and its clients.
- **Internal data-level security**—The third area for security has to do with the internal logic of your specific database solution when someone is legitimately logged in. Who has rights to delete records, who can make programming changes to the database, and who can view various layouts in the system? These details are internal to the workings of a FileMaker solution and deal with ensuring that your data remains both secure and reliable.

Every database solution should address these three areas. They might be addressed by the facts that your database will never leave your personal hard drive or be available to the network at large, but what happens if your computer is stolen or if a colleague sits down at it while you are away from your desk? We encourage all developers to consider security issues and make deliberate choices that are appropriate to the sensitivity of their information and the consequences they might face if it were compromised.

Together with version control, security is very difficult to retrofit. You can plan a security mechanism that you do not implement at the beginning, but the planning and any necessary database design changes should be present from the start.

Identifying Risks

Security concerns are not all targeted at the clichéd image of a sophisticated hacker sitting in a dark room somewhere surrounded by Mountain Dew cans and pizza boxes. Most FileMaker systems will never be exposed to that level of threat. If you have a reasonably secure network and keep access to your server (or hosting computer) controlled, you have addressed many of the concerns that an extreme case such as hacking represents.

The biggest security threat a database system faces is actually from the legitimate users of the system itself and often has most to do with data integrity. Let's use an example to illustrate: Consider a system for managing invoices that a company depends on for reporting monthly revenue. If every user of the system (including perhaps a temporary employee there to answer phones for a few days) has the capability to delete records, the chances that someone would inadvertently delete invoice records are quite high.

Or let's take a less clear-cut example: What if someone duplicated a record, intending to use the new record to create a similar invoice, but miskeyed the command and duplicated it twice? In those situations, the database could not be reliably depended on to deliver accurate revenue totals. Although these sorts of issues are not the result of intended harm to a database, they are a risk to the system, and its security architecture and data validation mechanisms have to address them.

The second general threat developers face is data sensitivity: In the examples given previously, would it be appropriate for everyone in the system to be able to run the monthly invoice summary report and see the financial performance of the organization? Or in the case of a database that tracks, say, human resources information, which users should have the ability to view the layouts on which people's salary history appears? Security plans need to include an assessment of what data

users can access (see and manipulate) in a given solution in addition to what they can do to that data. We find it useful to work with two general categories of risks to data within a database:

- **Data integrity**—Define the actions various users can perform on the data in your solution. Often revolving around the creation and deletion of records, risks can also include the capability to edit certain fields or run specific scripted routines.
- **Data sensitivity**—Define the degree to which information should be visible and accessible after a user legitimately logs in to a system. Risks include inappropriate access to private and proprietary information.

When you're approaching security for a given solution, it is important to identify the risks the organization faces in terms of both of these areas. We advocate the creation of a risks document in project planning that identifies these issues and the planned means of addressing them.

Your starting point in working with security must be any relevant laws, rules, and best practices you must adhere to in a specific business. In recent years, laws protecting privacy and identity theft have proliferated. In revising old databases, you might encounter data you would never put in a new database (credit card numbers, identification numbers such as Social Security numbers, and the like). Today, sensitive data requires a more than cursory review before its inclusion in a database. In the United States, legislation such as the Sarbanes-Oxley Act (officially the Public Company Accounting Reform and Investor Protection Act of 2002), HIPAA (the Health Insurance Portability and Accountability Act of 1996), and many other rules and statutes can enter into your security needs. In most cases, the client is responsible for letting you know what the needs are; a consultant can perform a valuable service by asking about these matters, but interpretation of the law is generally not the consultant's job.

Planning Security

When you're approaching a new system, it's important to identify the security issues you face and include a plan for your security architecture early in your development process. For example, you will have to plan ahead if some users of your system should not be allowed to view or work with some set of fields, records, or layouts. Security, like reporting, is often left until last when building a system, and, as with reporting, this tends to be a mistake. You will need to interweave access issues throughout your database solution (considering security when placing objects on layouts, writing scripts, and so on), and it is best to have this mapped out before building a solution.

Using a Security Matrix

To make sense of the myriad security issues many systems face, we recommend the use of a security matrix. Table 12.1 shows a simple example. It identifies parts of the system such as access, account control, data tables, scripts, and layouts (commonly known as *assets*) and specifies the type of access for each type of user.

Table 12.1 Security Matrix Example

Asset	Developer	IT Admin	Manager	Sales	Finance	Admin
Server Administration						
Access to server	Limited[1]	Full	None	None	None	None
Access to backup directory	Limited[1]	Full	None	None	None	None
Access to server admin tool	Limited[1]	Full	None	None	None	None
User Accounts						
New account	Full	Full	None	None	None	None
Delete account	Full	Full	None	None	None	None
Change password	Full	Full	None	None	None	None
Data Tables						
Customer view	Full	None	Full	Limited[2]	Full	Full
Customer new	Full	None	Full	Limited[2]	Full	None
Customer delete	Full	None	Full	None	None	None
Customer edit	Full	None	Full	Limited[2]	Full	Full
Invoice view	Full	None	Full	Limited[2]	Full	Full
Invoice new	Full	None	Full	Limited[2]	Full	None
Invoice delete	Full	None	Full	None	Full	None
Invoice edit	Full	None	Full	Limited[2]	Full	None
Product view	Full	None	Full	Full	Full	Full
Product new	Full	None	Full	Full	None	Full
Product delete	Full	None	Full	None	None	None
Product edit	Full	None	Full	Full	None	Full
Script Routines						
Monthly Revenue Report	Full	None	Full	None	None	None
Regional Revenue Report	Full	None	Full	Limited[3]	None	None
Layouts						
Customer List	Full	None	Full	Full	Full	Full
Customer Detail	Full	None	Full	Full	Full	Full
Invoice List	Full	None	Full	Full	Full	None
Invoice Detail	Full	None	Full	Full	Full	None

Asset	Developer	IT Admin	Manager	Sales	Finance	Admin
Product List	Full	None	Full	Full	Full	Full
Product Detail	Full	None	Full	Full	Full	Full

[1] Database developer will have full access to server during testing, but after deployment, passwords will be changed.

[2] Salespeople will be able to create, view, and edit customer and invoice records for customers and invoices in their region only.

[3] Salespeople will be able to run the regional revenue report, but it will report only on the region to which a salesperson belongs.

Note in Table 12.1 that managers have full access to create and delete data records, that salespeople have limited access to do so for customer records, and that people in the Admin role cannot make any changes to invoices (however, they can view invoice information).

An additional distinction to note is that although people in the Admin role can view invoice information, they do not have access to the Invoice List or Invoice Detail layouts. This suggests that other layouts might display invoice information, perhaps as related fields or within a portal. It is important to consider both the capability to view data globally throughout a system and the capability to make use of specific layouts. In most cases, it is not enough simply to limit access to specific layouts; you also need to limit access to the data itself.

Security grids such as the example in Table 12.1 need to be as detailed as they need to be; in other words, they depend on the circumstances you face. If you don't have six different roles in your organization, clearly you won't need the distinctions made in the example. If you want to grant some development privileges to people other than developers (say, the capability to modify certain layouts), you would need to add a subsection for that. This table should be taken as an instructional example and is not a comprehensive representation of a real-world system.

Finally, be sure to grasp the use of the phrase "to view" (both in this book and within FileMaker itself). In this context, we mean the ability to consume the data in various ways; a user can see the data onscreen, can choose to print (if printing is enabled for the user's account), can export that data (if exporting is enabled for the user's account), and can email data.

Planning Implementation

Implementing security is done largely in the Manage Security dialog, but before we walk through the mechanics of setting up security, you have to plan where and how to implement it from an overall perspective.

A significant part of your planning must include user interface considerations. If a user shouldn't have access to run a script, for example, she should be presented with a graceful message to that effect if she inadvertently attempts to do so (as opposed to the script simply not doing anything). Likewise, if someone doesn't have access to a layout, your navigation system should reliably prevent him from ever being left on that layout, or at least you should provide a way to get back to the part of the system to which he does have access.

Another consideration is the aesthetics of seeing <no access> displayed in various places throughout the system. FileMaker displays <no access> when a user isn't allowed to view field data, record data, or a layout. If you do not want to remind your users of their own limited privileges, you might choose to hide away restricted areas by controlling navigation or window access.

FileMaker 8 introduced the Custom Menus feature, allowing you to deal with many security considerations by simply removing access to certain menu items. For example, if you want to restrict users from being able to delete all records, you can choose to remove that menu item. It is critical to note, however, that this is simply a user interface mechanism. If users have some other means of deleting records (say, through a custom script you've written or some other aspect of FileMaker's interface), the only way to ensure that they cannot perform the restricted action is to control their ability to perform the fundamental action in their security settings.

*For more detail on custom menus, **see "Working with Custom Menus," p. 396.***

Here's another example of how security plays a role in your planning: If you want to prevent people from having to see fields to which they have no access, you can choose to place them on their own layout. You can control access to specific layouts; however, you cannot prevent users from accessing a Tab Control pane if they have access to the layout on which it sits. Given this, you might choose to create separate layouts where a Tab Control object might have served had you not considered security issues.

Your solution's scripts are another area where you will want to plan for different levels of access. If a user has a means of running a restricted script (say, by clicking a button that is present on all layouts), you will need to present him with a message that he is not permitted to use that function. A more subtle issue is what to do with scripts internal to the database operations; for example, you might write a script that allows users to choose different printer settings. If you restrict access to this script for some users, but then reference the script from all your reporting and printing routines, you will need to address that conflicting dependency. Likewise, if a script takes the system to a layout tied to a data table to establish context but the current user doesn't have access to that layout, your script might deliver unexpected results.

However you choose to approach security in your system, thinking through the user experience will be an important part of the overall plan. You should note in your layout designs and scripting where security considerations have to be taken into account.

Maintaining Security

The best planned security can deteriorate over time in many ways. User IDs might be reused as employees come and go, passwords might be posted on the sides of computer displays, and a variety of other compromises can occur.

Much of security maintenance can be managed if you have a robust password policy in place. For many organizations, use of someone else's user ID can be grounds for dismissal. You really need that type of clout to keep people from destroying audit trails and otherwise appearing to be someone they are not (with all the privileges set up for the other person). For this to work, your security

mechanism must allow for quick ad hoc adjustments to security. If someone is away, no one else should use that user ID; rather, the security administrator should be able to create a temporary user ID on demand.

A set of issues involves passwords: How often should they be changed, and how complex should they be? The proliferation of passwords is one of the biggest nuisances for computer users. Some passwords must have special characters in them; others must not. Some must be a certain minimum length, whereas others cannot be more than a specified length. The rules vary enough that a single individual might need a half-dozen passwords to comply with the rules of various systems. If you add to that a rule that passwords must be changed frequently, you quickly wind up with passwords taped to the display because otherwise no one can remember them. There is no simple answer to this problem other than the use of nonpassword security (which is generally biometric).

Yet another password issue has to do with who knows the passwords. In some organizations, a password is assigned to a new user of a system, and the user must then change it (perhaps periodically). After the initial password is changed, the administrator has no control over the password except to reset or delete it. This allows people to reuse passwords that they can remember.

In other cases, passwords are administered and changed by the administrator. (This is common with passwords for email at some ISPs.) Thus, the administrator has access to all the organization's passwords; that document is clearly one of the most important security vulnerabilities. For this reason alone, it is often a good idea not to allow anyone except each user to know a password. This solution means less centralized control, but more security in the end.

User-Level Internal Security

The mechanics of implementing security begin with the database file (or files) within your solution itself. Generally, security is first a development task (first planning and then implementation) and is then followed by issues of deployment. This chapter follows that same approach by first discussing how to grant individual users access to your database.

User Accounts

If you select File, Manage, Security, you are taken to the Manage Security dialog. This dialog has a good deal of depth, and it is through this dialog that you will implement much of your security architecture.

On the first tab of the dialog, Accounts, you create individual user accounts and assign a privilege set to each. It's important to grasp that various security settings in FileMaker are not controlled at the user account level, but rather are assigned with privilege sets. Accounts are associated with a privilege set, and this association determines the functionality a given user has access to, as shown in Figure 12.1. This allows you to define a privilege set for each role in your system and assign individual users to the corresponding set that matches their role for the database.

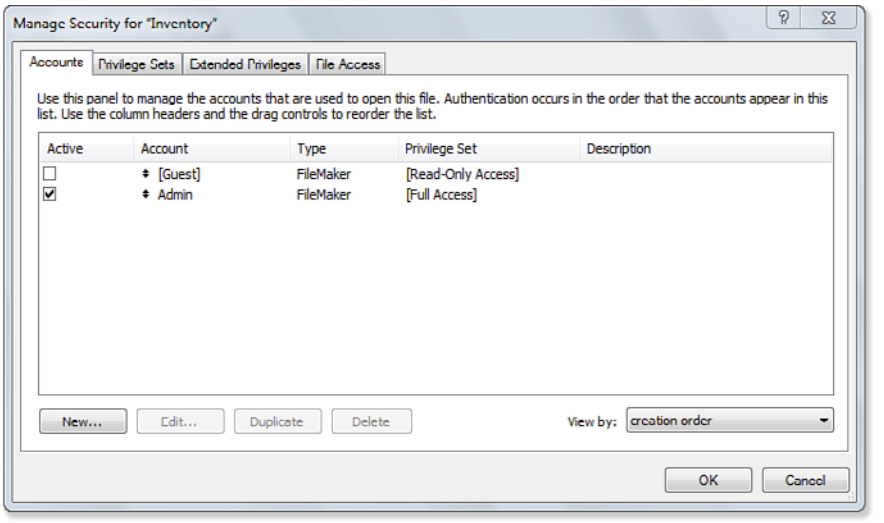


Figure 12.1
The Accounts tab of the Manage Security dialog enables you to see which accounts are active and what their respective privilege sets are.

Default Accounts and Automatic Login

By default, any new FileMaker file is created with an account named Admin with a blank password, and it is set to log in to that account. The Admin account is assigned full access privileges, so in effect the file is created with no restrictions whatsoever, but will have an account and privilege set in place. If you choose to lock down your database, either give the Admin account a password, mark it as inactive in the Accounts tab of Manage Security, or delete it. You should also disable the File Options setting that first tries the Admin account and password on login.

In addition to the Admin account, FileMaker provides a [Guest] account with each new database. The [Guest] account cannot be deleted and is set to be inactive by default. You can choose to enable this [Guest] account in cases in which you want to restrict the development functions of a database but want to open the rest of the system to any user.

To set a file to a default state in which users are not prompted to log in, create an account with the appropriate access level you prefer and then turn on the Log In Using option in the File Options dialog shown in Figure 12.2. Users can override the Log In Using option by holding down the Option key (OS X) or Shift (Windows).

Account Management

The settings in Figure 12.3 are more typical for a small workgroup application: The Admin account has been marked as inactive and there are four individual users with three privilege sets (Staff is shared). Note the full list of accounts (some of which are disabled) and the assignment of privilege sets.

Figure 12.2
A system can automatically log in users with a default account via the File Options dialog.

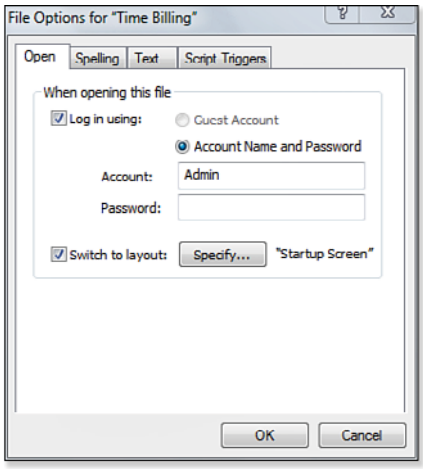
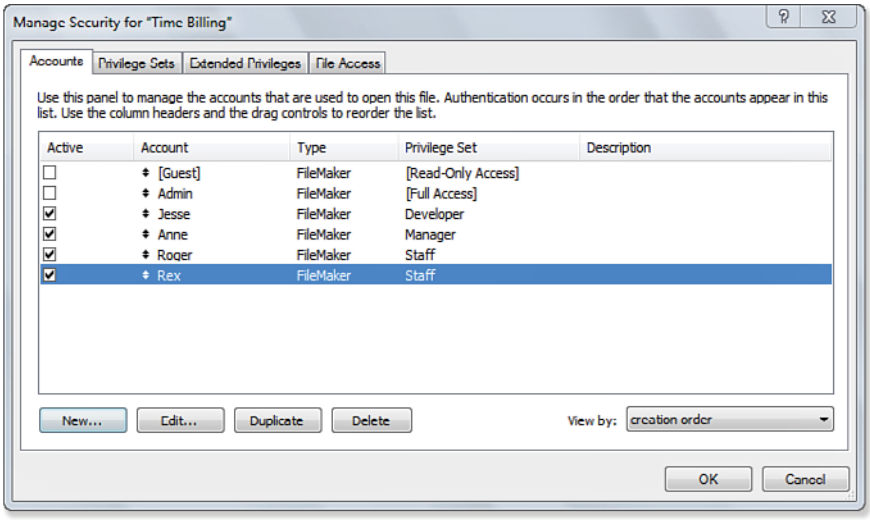


Figure 12.3
On the Accounts tab of the Manage Security dialog, you can review which accounts are active and to what privilege sets they belong.



The Type column shows the means by which authentication is set to occur. It shows either FileMaker, in which case a user's password is stored within FileMaker (in a fully encrypted, reliably secure form), or External Server, in which case authentication is managed by a separate authentication server. We cover external authentication later in the chapter.

When editing an individual account via the Edit Account dialog shown in Figure 12.4, you can control settings specific to that user. The setting to prompt users to change their password on their next login allows developers and database administrators to reset passwords without having to know the private passwords of their users. To administer a FileMaker database, we recommend creating temporary passwords for people and requiring them to change passwords on their next login.

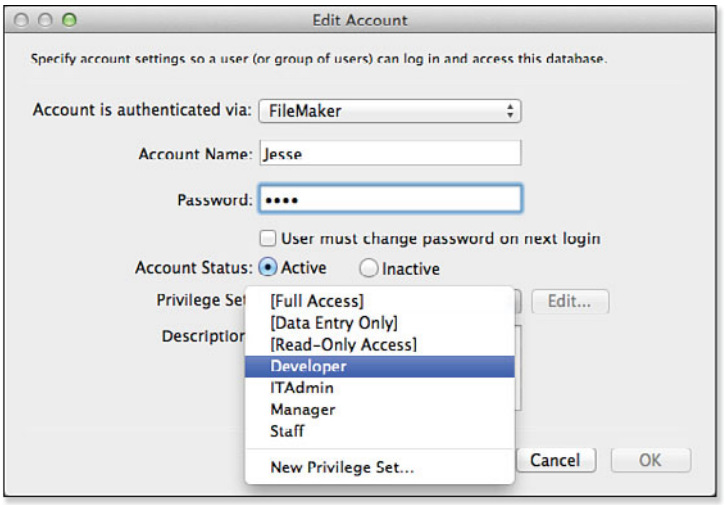


Figure 12.4
The Edit Account dialog enables you to control the authentication and active status for each user.

> *Note that this practice of creating temporary passwords is not recommended for Instant Web Publishing or for external authentication. Both topics are covered in "Setting Other Feature Privileges," **p. 353**.*

Note also that you can disable an account from the Edit Account dialog. This capability allows a database administrator to mark an account inactive without having to delete it. Having the capability to mark an account inactive is useful if some users are gone for extended periods or if you want to preserve the fact that an account exists with that specific name. You can also simply toggle the check box on the leftmost side of the Accounts tab (unchecking it to disable an account).

Finally, you can assign a user's privilege set. An account can have only one privilege set assigned, and that privilege set determines the specific rights and privileges the user will have.

Privilege Sets

Privilege sets compose the bulk of security control in FileMaker. With a privilege set, you can set various access levels, restrict functions and areas within a database, and control who can do development work within a given file. Privilege sets are associated (one to many) with accounts, and they can be thought of as analogous to groups. It is common to see privilege sets established for developers, managers, and so on.

> *If you need help with the testing process for privilege sets, **see** "Closing and Reopening File for Testing" in the "Troubleshooting" section at the end of this chapter.*

The Privilege Sets tab of the Manage Security dialog, shown in Figure 12.5, enables you to see at a glance which accounts are assigned to which privilege set.

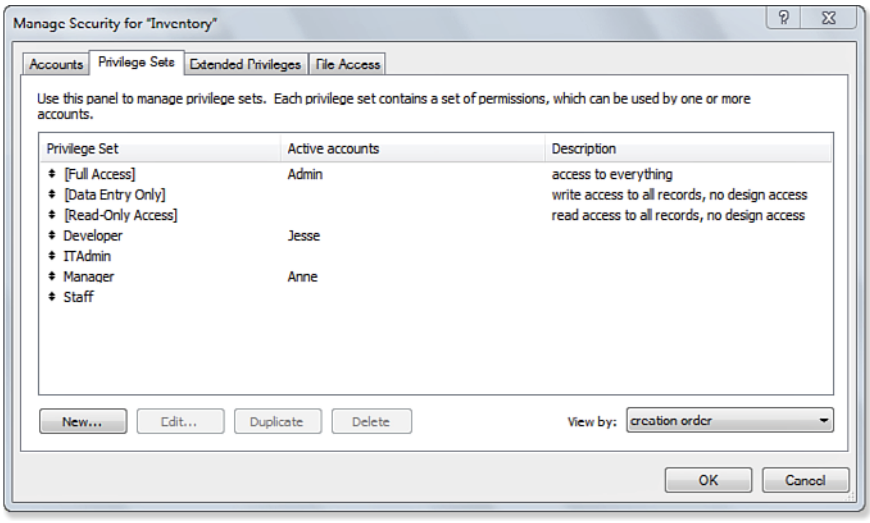


Figure 12.5
The three sets in brackets are defaults created for each new FileMaker file; those below are custom sets created for a specific database solution.

Notice the three sets at the top of the dialog: [Full Access], [Data Entry Only], and [Read-Only Access] are the default sets that FileMaker creates for a new FileMaker file. You cannot delete these sets.

The [Full Access] privilege set is a unique set: It is the single set that has complete access to the file including all development functionality. It cannot be duplicated, and your file must have at least one FileMaker-authenticated account associated with the [Full Access] privilege set. Without [Full Access], you wouldn't be able to get in and modify your database. Therefore, although you can temporarily create the configuration that was shown in Figure 12.3, you have to provide one account with Full Access before closing the window. (Although the default account is Admin, you can create another one to use Full Access.)

note

By using the Remove Admin Access feature of the Developer Utilities features in FileMaker Pro Advanced, you can remove the administrative/full access accounts associated with a file and prevent any future development. This is most frequently done with runtime solutions.

*To learn more about the Developer Utilities features, **see "Removing Admin Access," p. 631.***

If you select a privilege set from those listed and double-click (or click the Edit button), you are taken to the Edit Privilege Set dialog shown in Figure 12.6. It enables you to control both the features within FileMaker that assigned users can access and the degree to which members of a privilege set can perform additional development work on your database file.

The Edit Privilege Set dialog is divided into three areas: Data Access and Design, Other Privileges, and Extended Privileges. We look more closely at each area in the sections that follow.

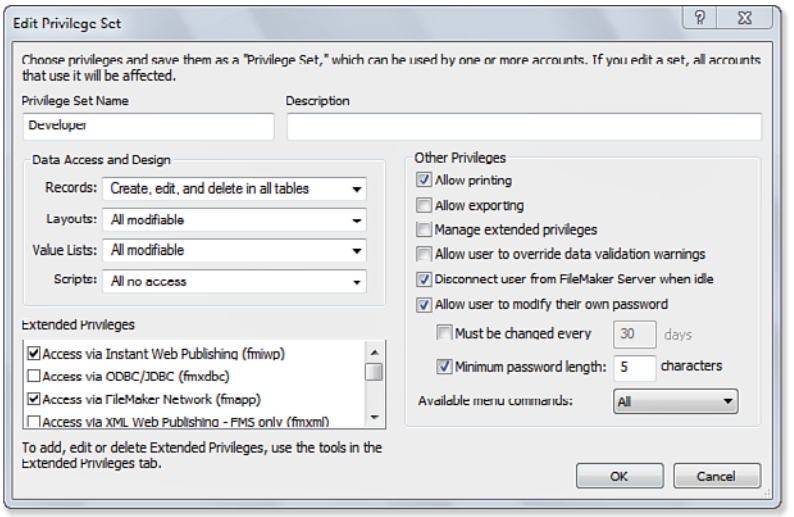


Figure 12.6
The Edit Privilege Set dialog allows you to define the security access for all accounts associated with a given privilege set.

Controlling Data Access

The actual data of your file is protected by the Records drop-down list in the Edit Privilege Set dialog. It is important to remember that although you can hide fields from users in various ways (for example, by not placing any field layout objects on layouts), the only way to fully protect your data is through the Records drop-down list.

The menu enables you to apply global permissions where a privilege set can have full access to all tables, no access at all, only the capability to create and edit records, or view-only access. View-only access means that users with this privilege set can see data but cannot make changes or create new records.

By choosing the fifth option, Custom Privileges, you open the Custom Record Privileges dialog, shown in Figure 12.7. Within this dialog, you can control on a table-by-table basis, or even a field-by-field basis, what data a given set of users can view, edit, create, and delete. Each table in your file is listed. You can select multiple tables by Shift-clicking for contiguous selections or Control-clicking for noncontiguous selections. Any changes made to the settings below are applied to each selected table.

Notice the settings for [Any New Table] at the bottom of your table listings. This privilege controls tables added to the file after your security settings have been defined. In other words, if you were to add a TeaPackage table to the database shown, this privilege set would initially have no access to the records in that table. The settings at the bottom of the dialog are listed here:

- **View**—Controls whether a set of users can consume the information stored in a selected table. By *consume*, we mean see in Browse mode, search for in Find mode, export, print, email, and so on. Users with View access can perform such actions as clicking into a given field and copying data to their Clipboard.

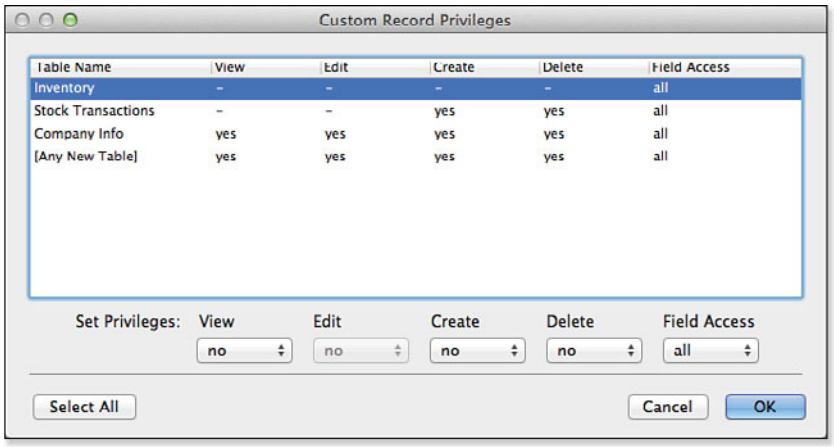


Figure 12.7
These settings show that access to this database has been restricted to a significant degree. The hyphen indicates no access.

- **Edit**—Allows users to make changes to data within a given table. Note that if you set View to No, Edit automatically shows as No as well.
- **Create**—Controls whether users can create new records in a selected table.
- **Delete**—Determines whether users can delete records from a given table.
- **Field Access**—Allows developers to apply view or edit privileges to individual fields rather than to an entire record. In cases in which you have applied settings to the View, Edit, and Field Access settings, the most restrictive setting takes precedence. In other words, if you set a table's View privileges to Yes but Field Access to None, users will not be able to see or edit any of the fields within that table. Likewise, if you set Field Access to All and View privileges to No, users will not be able to view any records in the given table.

The first four privileges listed offer Yes, No, and Limited options (with the exception of Create, which offers only Yes and No). We cover limited privileges shortly. Field Access controls have more granularity than the other record privileges. The All and None options should be self-explanatory, but the Limited option presents a list of all the fields in a given table, as shown in Figure 12.8.

note

By setting a field to View Only, you are ensuring that users logged in with this privilege set will be able to see data but not to be able to make changes to those fields.

Conditional Privileges

For record privileges except Create, you also have the option to choose limited privileges. By doing so for View, Edit, and Delete, you open a calculation dialog and can create conditional circumstances by which you can control access on a record-by-record basis.

*For a review of the calculation dialog and working with formulas, **see** Chapter 8, "Getting Started with Calculations."*

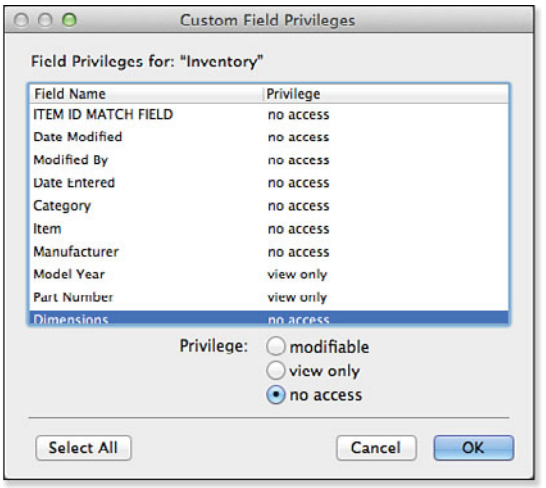


Figure 12.8
Field-level access enables you to control individual fields for a given privilege set.

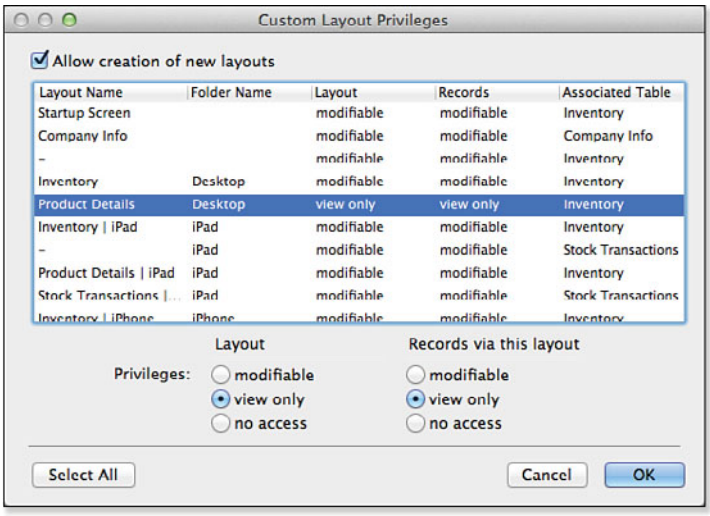
One common conditional privilege calculation is used to govern the deletion of records. If you are storing the creation date or last modification date of a record in a field, you can create a calculation that checks if the date is today. If it is, deletion is allowed; if not, deletion is not allowed. If you are storing the creator or last modifier name, you can further refine the test so that deletion is allowed today only for records that the user created (or modified).

Controlling Layout Use and Development

In the Edit Privilege Set dialog, the next setting after Records is the Layouts drop-down list for controlling layout privileges. With it, you can set the following:

- **All No Access**—This setting ensures that people associated with the privilege set you're defining will have no access to any layouts within the current file.
- **All View Only**—The term *view*, again, really means *consume* or *use*. Users assigned this privilege will not be able to make changes to a layout in Layout mode, but they will be able to use the layout and (assuming that the developer hasn't omitted the layout from that menu) see it in the menu of layouts offered via the Status toolbar.
- **All Modifiable**—This option enables you to change to Layout mode and to then edit all the layouts within a file.
- **Custom Privileges**—Choosing this option takes you to a dialog similar to the one shown previously in Figure 12.7 except that this one controls layouts rather than records (see Figure 12.9).

Figure 12.9
This dialog enables you to control who can modify which layouts.



The Custom Layout Privileges dialog enables you to set only specific layouts as modifiable or to turn off access to selected layouts. Furthermore, you can control how users interact with records via the layout in question. You can choose to lock down record access on a layout-by-layout basis. Be aware of the Allow Creation of New Layouts option with the check box in the upper left of the dialog. With it, you can enable someone to add layouts to a file without giving them access to the layouts you, as a developer, created. Imagine the possibility of allowing users to add columns to report layouts, for instance, without having to give them unfettered access to the entire system. In general, modification is safer than creation when it comes to layouts.

tip

One important note about layout access: Just because you lock down access to a certain layout does not mean your users cannot get access to the data in your file. They might be able to pull information via export, might be able to create another FileMaker file and create their own layouts, and so on. The best way to control your data is to lock down both record access and layout access as appropriate.

Controlling Access to Value Lists

The drop-down list for controlling value list privileges in the Edit Privilege Set dialog is similar in function to that of layouts. You can enable all value lists to be modifiable, view (or use) only, and all no access. Likewise, you can choose Custom Privileges and will be presented with the Custom Value List Privileges dialog shown in Figure 12.10.

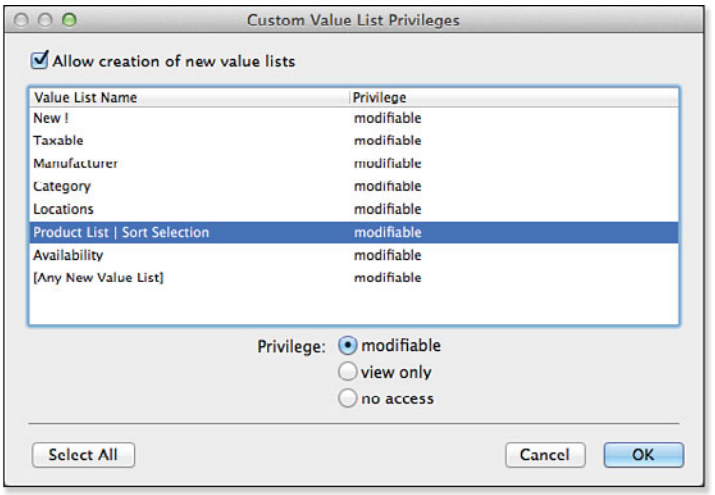


Figure 12.10
The Custom Value List Privileges dialog enables you to, among other things, enable others to edit value lists.

The dialog shown in Figure 12.10 can be used to prevent value lists from being edited or, indeed, from being used at all. If a field has a value list associated but a given user doesn't have access to use it, that user will be presented with <No Access> messages for radio buttons or check boxes. In the case of a pop-up menu, the user will be able to see an already selected value but will not be able to select a new one. And, last, for a drop-down list, the list will simply not appear (nor will the down arrow, if present, do anything), and the field will behave as though no value list were associated with it.

tip

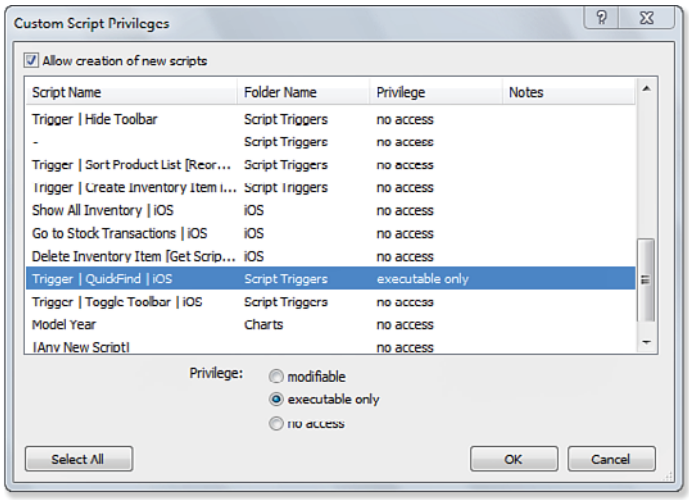
Note that, as with layouts, you can control the capability to create new value lists. If you've given some users limited capabilities to create layouts, it's somewhat likely that they will need to create value lists as well. The two settings often go hand in hand.

Controlling the Capability to Run Scripts

Developers can often control access to scripts by controlling where in a database's interface scripts are executed: by button, via the Scripts menu, or as associated with a custom menu. However, in cases in which you simply do not want a class of users to run scripts, the fourth drop-down list in the Data Access and Design area of the Edit Privilege Set dialog controls the capability to execute scripts. As with the other menus, you can quickly set permissions so that all scripts are executable, all modifiable, or all disabled (no access) for a given privilege set. In addition to the global menu choices, you can choose Custom Privileges, which presents the dialog shown in Figure 12.11.

The dialog shown in Figure 12.11 lets you set scripts to be modifiable, executable only, or to allow no access for the current privilege set.

Figure 12.11
The Custom Script Privileges dialog enables script-by-script control over access.



The Notes column will indicate if a script has been set to run with full access. It also reminds you that only people logged in with the [Full Access] privilege set can modify scripts set to run in full access. This is also true for the capability to enable running scripts with full access: If someone is not logged in with [Full Access] privileges, she will not be presented with the Run Script with Full Access check box in ScriptMaker.

*For more discussion of running scripts with full access, **see** "Full Access Privileges," **p. 259**.*

It is important to understand that a script set to run with full access will do exactly that: A user's security privileges will be overridden, and the script will execute as though it were run by a user with [Full Access] privileges. This dialog, then, is useful in making sure that you can prevent users from executing a script, even if it is set to run with full access privileges.

Setting Other Feature Privileges

The area on the right of the Edit Privilege Set dialog controls access to a few of FileMaker's interface commands and offers some specific settings related to security.

The Allow Printing and Allow Exporting options should be somewhat obvious, but be sure to note that they also control the functions to Save as PDF (tied to the capability to print) and Save as Excel (tied to the capability to export). If you want to prevent your users from taking data elsewhere, you will need to turn off printing and exporting. Note, too, that the only way to prevent users from making use of the Email command is to use a custom menu to remove that menu item. There is no security setting that controls whether someone can use the Email command in the File menu.

Allowing users to override data validation warnings should be obvious as well. When a validation error occurs, users with this privilege will not be presented with the capability to accept an invalid entry into a validated field, regardless of whether Allow User to Override Data Validation Warnings is turned on for the field. This provides you with a means of taking away the capability to override validation warnings from some users.

*For more detail on validation, **see** "Field Validation," **p. 102**.*

The option Disconnect User from FileMaker Server When Idle (the time interval is defined in FileMaker Server's settings) should almost always be enabled. When it's disabled, the server will never disconnect idle users who have this privilege set. One occasion to disable this setting is if you need a client computer set up to perform automated tasks.

note

We recommend using the Allow User to Modify Their Own Password feature and requiring users to change their passwords, but remember that Instant Web Publishing and external authentication do not support this capability. Recommended best practices suggest that passwords should be changed regularly and should be of a certain minimum length. (There is no capacity in FileMaker to set rules about the content of a password, only its length.) Note, though, that these settings can get you in trouble: If you don't allow someone to change his own password, but on the Edit Account dialog require that he do so on the next login, the user can get trapped and be unable to log in to the database.

The last setting, Available Menu Commands, allows you to disable FileMaker's menu items, leaving just those to open and close a file, run scripts, and so on, or additionally the Clipboard and spelling items in the Edit menu. This option is often used to completely lock down a FileMaker solution. When you disable all menu items here, it's an all-or-nothing proposition that will then require that you re-create all the functionality you want users to be able to have. We often recommend instead using a custom menu set that doesn't contain the items you're trying to hide from users.

*To learn about custom menus, **see** "Working with Custom Menus," **p. 396**.*

caution

Note that the feature to change passwords is not supported by Instant Web Publishing. Do not enable it for users who will exclusively access your database via IWP.

tip

Note that if you choose Minimum or Edit Only, these settings will disable custom menus just as they will standard FileMaker menu items.

Extended Privileges

Extended privileges comprise the third area of security within FileMaker files. Think of extended privileges as nothing more than on/off switches. A privilege set has a specific extended privilege either enabled or disabled. There are no other settings or logic to extended privileges.

Default Extended Privileges

FileMaker Pro ships with some extended privileges already in place, as shown in Figure 12.12. These privileges are used to enable access into FileMaker by various means; the behaviors of these privileges are controlled by FileMaker itself.

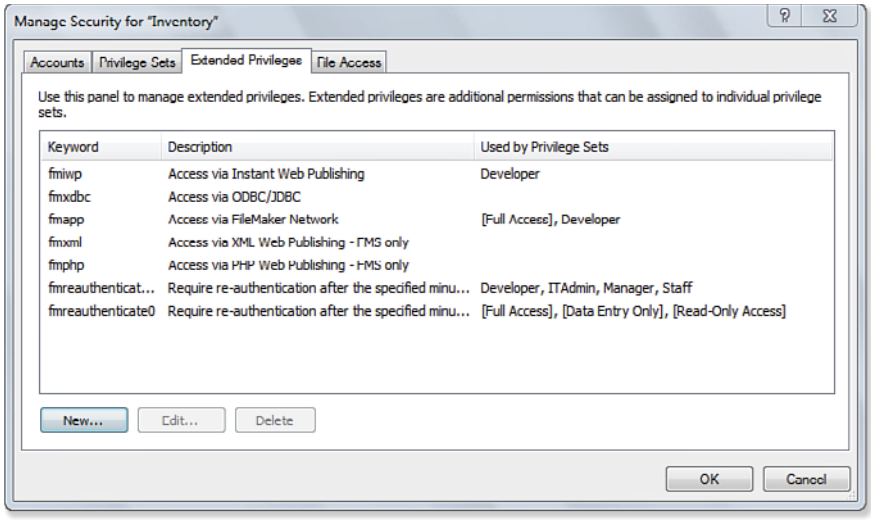


Figure 12.12
FileMaker ships with a set of existing extended privileges.

The default extended privileges are as follows:

- **Access via Instant Web Publishing**—This privilege enables users to access the file via a web browser using Instant Web Publishing.

  *To learn about Instant Web Publishing, **see** Chapter 24, "Instant Web Publishing."*

- **Access via ODBC/JDBC**—Access via ODBC/JDBC needs to be enabled if you want an ODBC or JDBC client to use SQL to converse with FileMaker.

  *To learn more about xDBC connectivity, **see** Chapter 22, "Importing Data into FileMaker Pro," and Chapter 23, "Exporting Data from FileMaker."*

- **Access via FileMaker Network**—This privilege allows users to access the file remotely, across a network, using FileMaker Pro (or Advanced) client connections. This is true for both peer-to-peer sharing and hosting files on FileMaker Server.

  *To learn about hosting FileMaker via FileMaker Server, **see** Chapter 27, "FileMaker Server and Server Advanced."*

- **Access via XML Web Publishing**—As noted in the dialog, this extended privilege works with files hosted by FileMaker Server Advanced. It allows users (or other systems) access to your data via XML.
- **Access via PHP Web Publishing**—This features enables you to publish files hosted by FileMaker Server Advanced using the FileMaker PHP API.
- **Require re-authentication after the specified minutes**—There are two privileges in this category: `fmreauthenticate10` and `fmreauthenticate0`. For FileMaker Go, they require reauthentication after it has been asleep or in background for the designated number of minutes. A 10-minute grace period might be right for trusted users, whereas casual users might need a 0-minute grace period. Figure 12.13 shows details of this extended privilege.

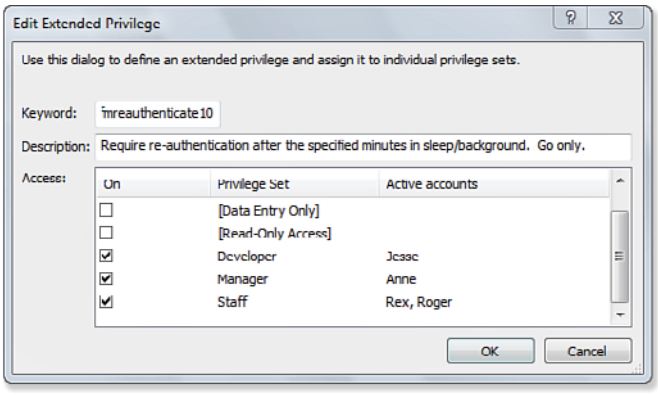


Figure 12.13
You can control re-authentication in FileMaker Go with extended privilege sets.

*To learn about Custom Web Publishing, PHP, and XML, **see** Chapter 25, "Custom Web Publishing with PHP and XML."*

*If you are having trouble getting your database files to appear on FileMaker Server and are sure that your authentication is correct, **see** "Database Doesn't Appear on FileMaker Server" in the "Troubleshooting" section at the end of this chapter.*

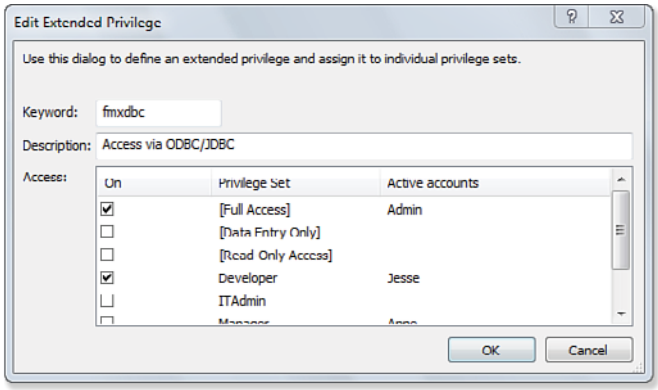
Custom Extended Privileges

Beyond the six extended privileges included with FileMaker, you can add your own. After creating the extended privileges, you can use the `Get (ExtendedPrivileges)` function to see what extended privileges have been granted to the current user. This gives you the capability of modifying your solution's logic to take these extended privileges into account; script branching, calculation results, field validation, and even custom menu loading could all take the current extended privileges into account. The options are nearly endless.

One of the advantages of extended privileges is that you can grant users the ability to manage them. On the right of the Edit Extended Privilege dialog, as shown in Figure 12.14, you can give

users access to a security dialog devoted only to extended privileges. You might choose to do this if you want to expose some security control in your system without granting someone [Full Access].

Figure 12.14
Users can assign which privilege sets are associated with an extended privilege when the Manage Extended Privileges security setting is enabled.



Another advantage to extended privileges is that they are not session specific: If you enable an extended privilege for a privilege set, it is immediately available to all users logged in with that privilege set. If, for example, you want to disable access to a file via the Web while you complete some development, you can simply turn off the extended privilege for all the associated privilege sets.

As an example of putting extended privileges to use, consider a database with a series of reports that many, but not all, users will need to have access to. You could certainly control whether users could execute the individual scripts and navigate to the layouts in question, but creating an extended privilege for these reports will allow you to enable them across your various privilege sets in one central place. You can also disable them when it would perhaps be inappropriate to run them before some full set of data is input. A portion of the script you might use to begin the reporting procedure could look like this:

```
If [PatternCount (Get (ExtendedPrivileges); "yearendReports")]
  Perform Script ["Goto Report Menu"]
Else
  Show Custom Dialog ["Reports Access"; "Access not allowed."]
End If
```

Extended privileges are a useful way to extend FileMaker's security model into the logic of your solutions.

File Access

The last tab on the Security dialog lets you control which files can access the file in which you are working. This provides a new dimension of security beyond that of accounts and privileges. As you see in Figure 12.15, by default this tab is blank. No external files have been authorized, which means that *any* file can access this one subject to the security settings for the accounts defined in this file.

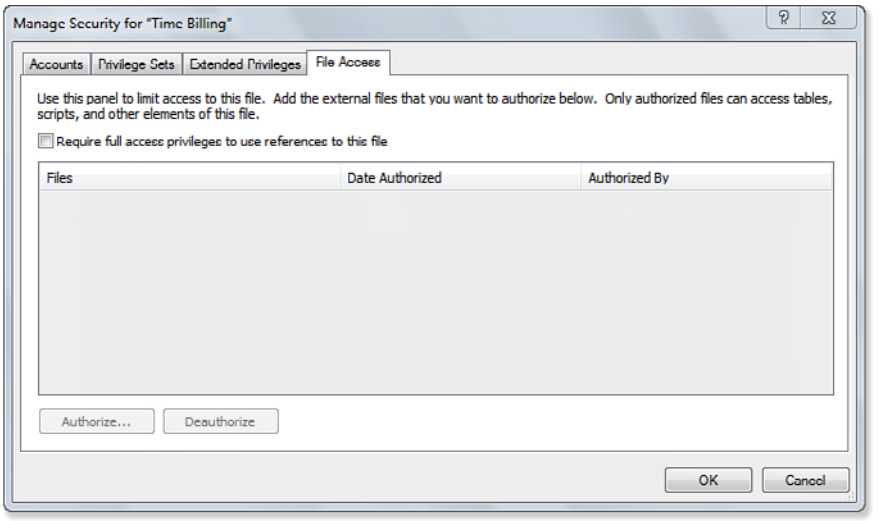


Figure 12.15
Blank file access means all files have access.

Use the check box to flip this setting; when it is checked, only listed files have access. Once you have turned the check box on, the Authorize button at the lower left of the dialog is enabled. You can use it to navigate to a file that you want to authorize, as shown in Figure 12.16.

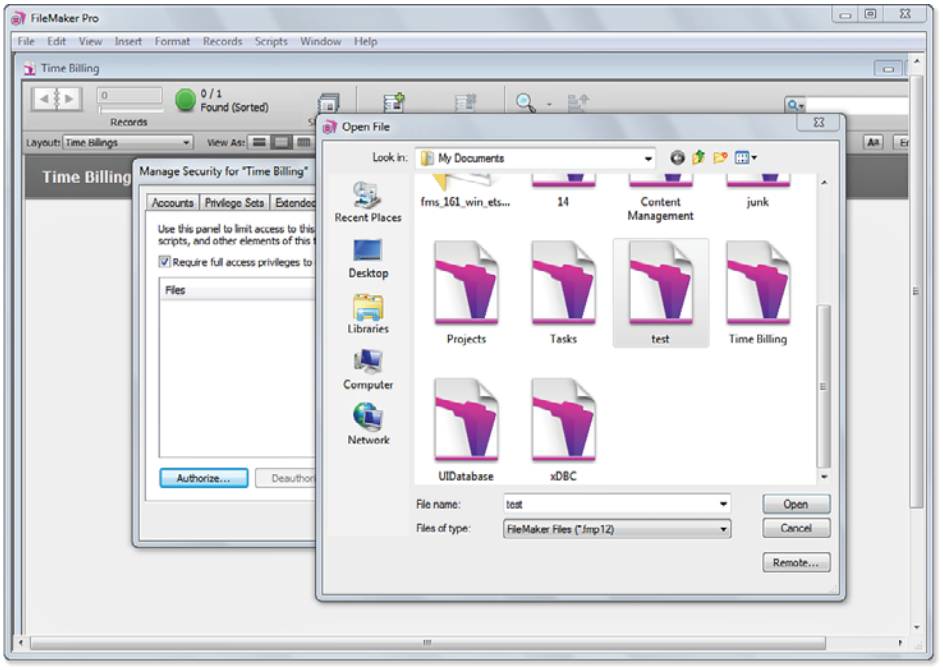


Figure 12.16
Authorize a file to access the current one.

When you click Open, that file will be added to the list of authorized files, as shown in Figure 12.17. Also note that a record is created of who authorized that file and when.

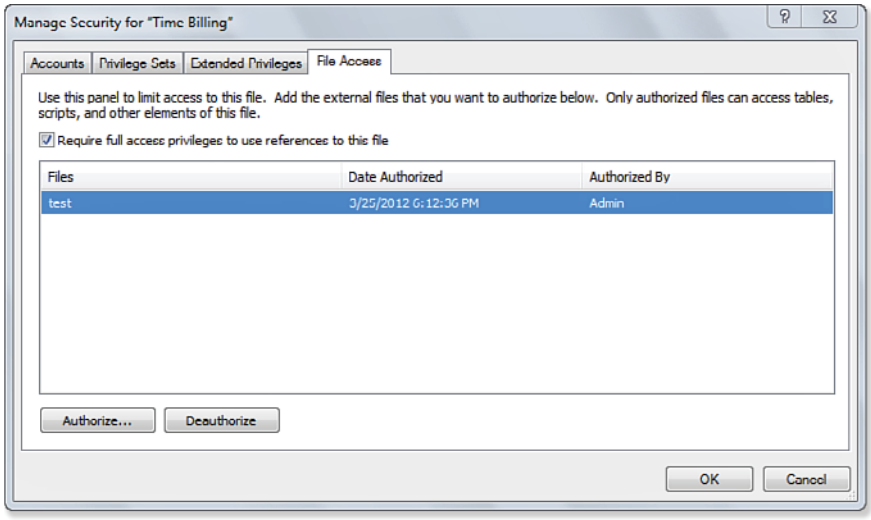


Figure 12.17
A record of authorizations is kept.

This level of security works together with account security: Both the account and the external file must have access.

File-Level Access Security

After you have a FileMaker database properly secured from a development standpoint, you need to consider how people will gain access to the file itself and log in. If you are hosting the file peer-to-peer, FileMaker's internal file security is your only option. FileMaker Server, on the other hand, has additional security settings and capabilities that can better safeguard your solutions.

*If you've forgotten your Admin-level password to a database (and mostly just need sympathy), **see** "Forgotten Admin Password" in the "Troubleshooting" section at the end of this chapter.*

Server Administration Security

To protect the files and access to your databases properly, you will need to consider the physical makeup of your server and its environment. Is it in a locked room? Is it properly situated behind a firewall? The actual hardware configuration for FileMaker Server from a security standpoint requires

that you follow the best practices of IT organizations in general. Although this is not an exhaustive list, here are some guidelines:

- Place your physical server in a controlled, locked room.
- Make certain that the server is situated behind a firewall and that as few ports as possible allow traffic, especially incoming, to it.
- Do not turn on file sharing for the server. Putting files on the server should be something you do *on* the server or from another computer using Admin Console.
- Do not allow file sharing on your backup directory, or at least secure it from your organization's general network.
- Make certain to secure the server with OS-level accounts and passwords, and set it to lock automatically after a short period of idle time.

As stated previously, this list is hardly comprehensive. The intent here is that you consider the environment in which you place FileMaker files as carefully as you've considered the development of your files internally. If you spend effort to lock away data from certain accounts but then leave backup files within easy reach of everyone on your network, your exposure to risk increases.

The first step in securing your server is to establish a password for administering the server itself. You will need to use the FileMaker Server Admin Console to configure these settings.

*To understand how to configure FileMaker Server using the FileMaker Server Admin Console, **see** "Using Admin Console," **p. 657**.*

You also have the option of enabling remote administration and setting an additional password for such. This will allow you to open the FileMaker Server Admin Console from any computer and, using a network address, connect to your server to perform tasks such as opening and closing databases as well as running backup routines.

tip

Even when your server is physically secure, we recommend setting a password for administering FileMaker Server. Good security is a case of rainy-day thinking, and the more precaution you take, the better you'll avoid unanticipated problems.

Security over the Network

In addition to the administration of the server, you will have to consider securing the data stream that passes from FileMaker clients and FileMaker Server. FileMaker uses TCP/IP as its network protocol, and when you either host a file via FileMaker Server or share it via peer-to-peer connections, information passes from host to client in a near-constant exchange. To secure this stream of data from possible threats such as network packet sniffing software, you should minimally use firewall and VPN technologies to prevent outsiders from gaining access to your internal network. To provide the most secure environment possible, you can also choose to encrypt the data passing from FileMaker Server to clients. You begin by going into FileMaker Server Admin Console, as shown in Figure 12.18.

In the Database Server section at the upper right, choose Security Settings to open the dialog shown in Figure 12.19.

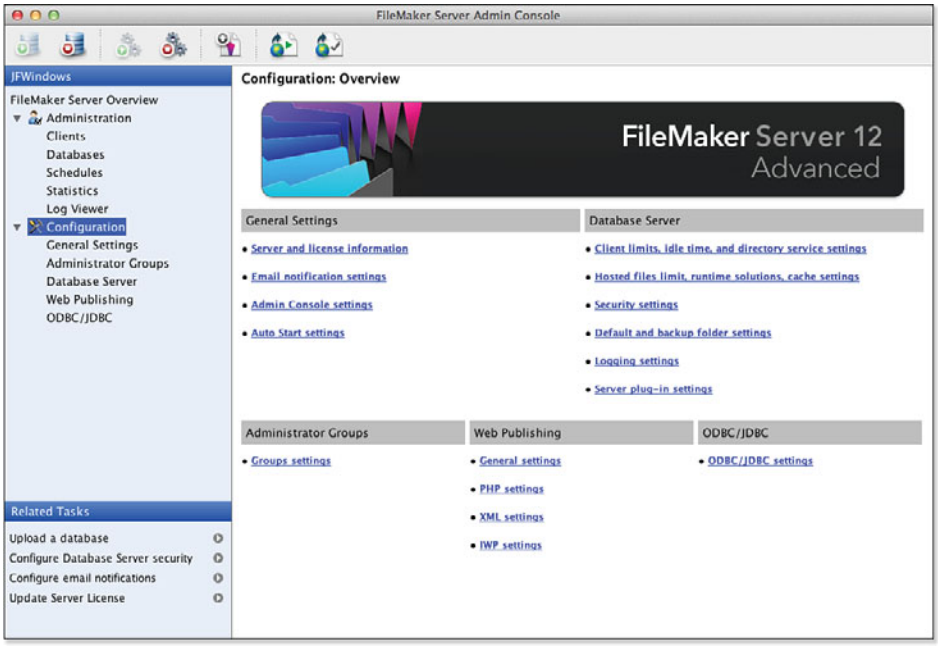


Figure 12.18 Use Server Admin Console to adjust security.

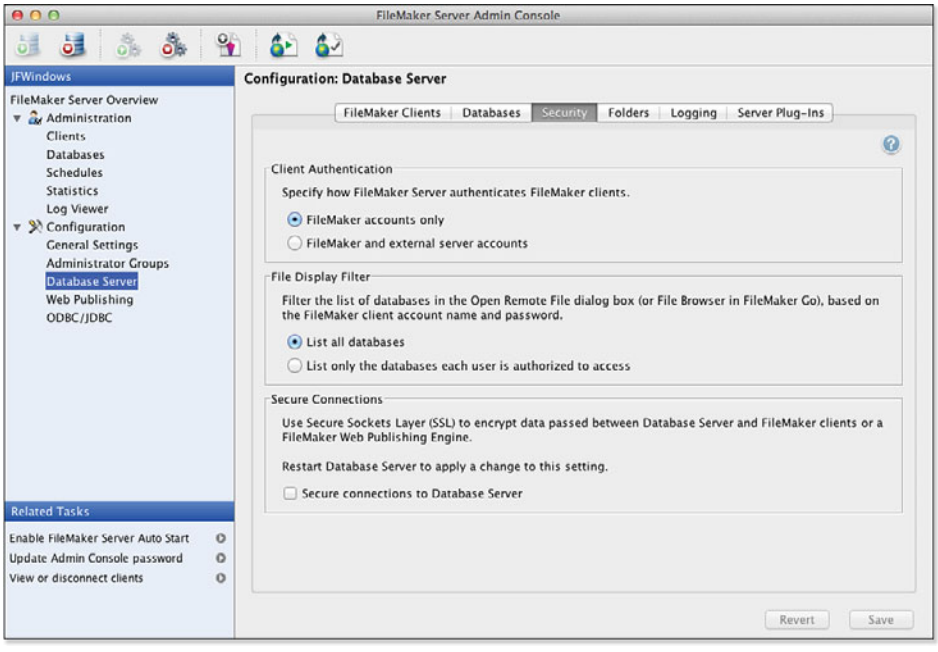


Figure 12.19 Adjust security settings in Server Admin Console.

FileMaker client computers decrypt encrypted information before displaying it. The setting is at the bottom of the dialog shown in Figure 12.18: Secure Connections to Database Server.

User Authentication

After your server is secure and you protect your network traffic by either isolating your network itself or encrypting your data (or both), you need to establish a means for each individual user to authenticate to your databases. An account can be authenticated either internally or externally. In the Edit Account dialog, if you set the authentication method to FileMaker, the account names will be stored within the file. Passwords are not actually stored in the file; they are encrypted every time they are used or changed with a one-way hash algorithm (based on respected industry-standard security methods). Each time your password is used, its encrypted hash changes. If someone were to gain access to your FileMaker file and crack the file somehow, he would be able to decipher only the last used hash algorithm. If you take the added precaution of securing your FileMaker files on FileMaker Server, you will remove even the opportunity to manipulate the physical database file.

External Authentication

External authentication is the other means by which users' credentials can be tested before they gain access to a database file. When you designate an account as externally authenticated, in the Edit Account dialog shown previously in Figure 12.4, the dialog changes to be slightly different in that no password or individual user account is specified. Instead, a group name is associated with a privilege set (just as accounts are), as shown in Figure 12.20.

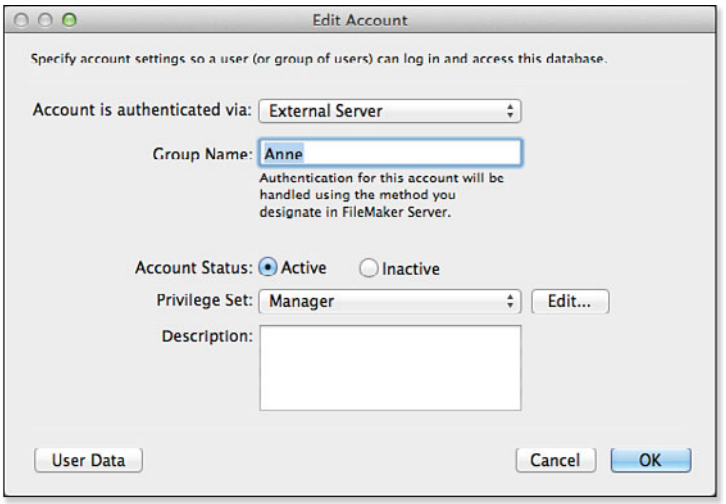


Figure 12.20
External authentication passes credentials to an external server and expects a list of valid groups in return.

Instead of creating an account name and password, with external authentication, you designate just the account name. Most often this name will correspond to a group created on the external authentication server. For example, consider a user of your system, Merzal Gold, with an Active Directory account of `mgold` and a password of `c0pp3r`. His Active Directory administrator assigned him to the

companywide groups Sales and Marketing. When he logs in with `mgold/c0pp3r`, FileMaker Server passes these credentials to the Active Directory server. That external server then verifies that he's logged in correctly according to his credentials and returns a list of groups—Sales and Marketing—to FileMaker. FileMaker scans its accounts list for externally authenticated accounts with either of those names and logs Merzal in to the first match it finds.

It is important to understand that, for an externally authenticated account, individual user accounts and their passwords are managed and stored not by FileMaker itself, but by your server's operating system (hence, external to FileMaker). If your FileMaker Server is part of an Active Directory (on Windows) or Open Directory (on OS X) domain, your users will be authenticated by the server that controls access to your domain. This authentication server could be the same computer on which FileMaker Server is hosted or a different computer. If your server instead makes use of local users and groups, FileMaker Server uses those accounts for authentication.

This external authentication is used solely to determine whether someone should have access to a FileMaker file and to what groups that person belongs. The only thing FileMaker relies on in external authentication is to have the operating system verify a person's password and return the group names to which she belongs. Those names are compared to the externally authenticated accounts within FileMaker and a user's privileges determined by the first valid match.

Keep in mind that authentication is determined on an account-by-account basis. You can combine externally authenticated accounts with internal FileMaker accounts as needed. In fact, FileMaker requires that you keep at least one internally authenticated account associated with the [Full Access] privilege set to ensure that you will still be able to access the file if the external authentication server is unavailable.

Note that you can opt to have an internal account name and externally authenticated account name be identical. You, as a developer, might have credentials on the external server but also want internal authentication. The first account listed by authentication order in the Manage Security dialog will be used if duplicates are present.

By default, FileMaker Server is set to allow only internal authentication. To enable both external and internal authentication, you will have to use Server Admin Console and check the FileMaker and External Server Accounts check box shown at the top in the Client Authentication section, which was shown in Figure 12.20.

note

Note that within FileMaker an account can be associated with only one privilege set; however, in an externally authenticated scenario, a single user might belong to multiple groups. It is the first group, from top to bottom (when sorted by authentication order), to which a user will be associated when externally authenticated.

This then means that you will need to coordinate the naming of groups between your authentication server and accounts within FileMaker. We recommend adopting a naming convention that reminds your server administrator that the groups established are there to serve your FileMaker databases.

caution

A few words of caution regarding external authentication: It is theoretically possible for someone to gain access to the physical file of your database, host it on his own FileMaker Server, and then manage to rightly determine what group names were used in its security scheme to grant himself access. This multistep process is fairly unlikely, and you can protect against it by securing your server and keeping your [Full Access] accounts tightly controlled. Note, too, that external authentication requires FileMaker Server. It is not supported in peer-to-peer hosting.

File List Filtering

The last element in security is a final bit of protection and convenience: Users can't break into databases they don't know are there. Using FileMaker Server, you can limit the list of databases users see to only those to which they have access. In the FileMaker Server Admin Console, enable the Display Only the Databases Each User Is Authorized to Access setting.

When a user uses the Open Remote File dialog and chooses a server, FileMaker first tries the credentials the person used to log in on her computer (based on operating system). If that fails, the user is asked for a username and password. On Windows, you can go directly to the username and password prompt by holding down the Shift key when selecting a server.

The process is similar in OS X, but the credentials used are stored in the keychain. To override the keychain, hold down the Option key when selecting a server.

After users are authenticated, they are shown a list of databases within which they have valid accounts or group memberships. In the case of external authentication, this is a seamless process. In the case of internally authenticated databases, users may have to enter their login information twice: once to get a list of databases and a second time when logging in to the specific database they then choose. However, if you have set an auto-enter login in File Options, that will be used and there is no two-step login.

Troubleshooting

Forgotten Admin Password

What can I do if I forget my Admin password and no longer have [Full Access] privileges?

Unfortunately, you're out of luck. In the past, FileMaker was able to open its own databases and provide access again, but this is no longer technically possible given the encryption used.

We recommend that you create two accounts with [Full Access] privileges and make sure that you (or a single person in your organization) aren't the only one who can gain full access into a database file.

Database Doesn't Appear on FileMaker Server

I am hosting a file on FileMaker Server. It's open and I know my [Full Access] admin account information, but the file doesn't show up in the list of databases available on that server.

Remember that to access a FileMaker file remotely through the network, you will need to turn on the extended privilege for FileMaker Network Access (fmapp) from a computer other than the server. Even though you have a [Full Access] account, by default no extended privileges are enabled for any accounts within FileMaker.

Closing and Reopening File for Testing

I'm trying to enable security settings, and I have to keep closing and reopening the file to do proper testing. Is there some better way to test other accounts?

We strongly recommend that you create a logout script and button for your users to use. Then, after that's available via whatever interface you choose, we recommend folding some conditional logic in for you as a developer. If, for example, you hold down the Shift key in the following script, it performs a re-login step without forcing the user to close and reopen the file:

```
Logout
# purpose: Logout with relogin for testing
# Shift key performs relogin
If [ Get ( ActiveModifierKeys ) = 1 ]
    Re-Login [ ]
Else
    Close File [ Current File ]
End If
#
```

FileMaker Extra: Working with Multiple Files

Throughout this chapter we have been careful to note that security settings are specific to a single file within FileMaker. If you have a solution that spans multiple files, you will have to duplicate your account and privilege settings across those multiple files. This task can become onerous when you have more than a handful of accounts or groups.

Privilege sets within FileMaker cannot be programmatically controlled. In other words, you cannot use a script to define a privilege set. You will have to create the appropriate privilege sets in each file of your solution. This is as expected; by definition, each privilege set should be specific to the file in which it sits.

Accounts, on the other hand, can be managed by script, and there are some techniques you can use to simplify the management of accounts within your suite of files. The following are the Accounts script steps available to you:

- `Add Account`—Using this script step, you can add an account to your file. Note that the script step dialog shown in Figure 12.21 requires that it be associated with a specific privilege set. You will need to create as many `Add Account` scripts as you have privilege sets. Note that this script step works only for internally authenticated accounts.
- `Delete Account`—This script step enables you to permanently remove an account (by name) from your file.
- `Reset Account Password`—Resetting the account password allows you to change it to a default of some sort without knowing the prior password.
- `Change Password`—`Change Password` requires the current password followed by the new password.

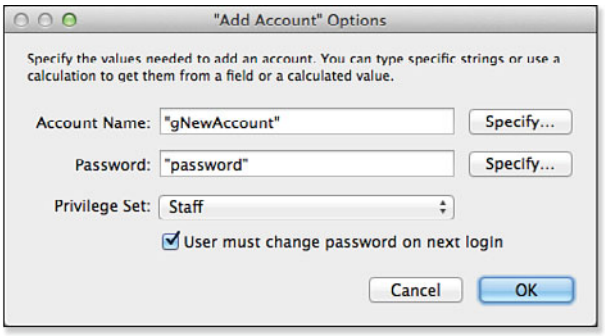


Figure 12.21
The `Add Account` script step allows you to add internally authenticated users to a file.

- `Enable Account`—This script step enables you to activate and deactivate an account. This is a nonpermanent way to deny someone access.
- `Re-login`—FileMaker prompts the user to log in again, or you can store credentials in this script step to do so without a dialog.

To use these script steps to support work in multiple files, you will need to create a user interface that allows users to create accounts and change passwords. Note that in the dialog shown in Figure 12.21, which you complete as you are building the script in ScriptMaker, there is a Specify button next to both the Account Name and the Password boxes.

First, create a layout (or dialog) that asks a user for a new account name and password she wants to add to your multifile solution. You can also, by default, set all new passwords to "password" and require the user to change them immediately. You should have the user enter that information into two fields set with global storage. You will also need a pop-up menu of privilege sets available within your solution, and certainly you should feel free to omit those that are sensitive, such as [Full Access].

Second, write a script that calls subscripts in each of your solution files. The subscript should use the `Add Account` script step to create the new user account. Where you have to provide an account name and password, simply reference the two global fields you had set up previously. Note that with global fields, you can add an external table occurrence from a different file and gain access to the fields as you need. You can follow similar techniques for changing passwords, deleting accounts, and enabling them.

13

USING THE WEB VIEWER

Introducing the Web Viewer

The Web Viewer is a layout element that uses the platform's native Web controls (Internet Explorer on Windows and Safari/Web Kit on OS X) to display web pages.

Although it displays web pages, the Web Viewer is not a browser. Some of a browser's features, such as built-in history, are not present. However, by the same token, any basic functionality built into the Web controls or added by the user is available. This includes plug-ins such as Flash.

You can use the Web Viewer to display a static address that you hard-code into the layout such as your own website. You can also use the Web Viewer to display data from a URL stored in a given database record—perhaps the home page of a client or a Contact Us page with updated names and phone numbers. You can also use built-in templates to merge a database field, such as a tracking number, with a request to a delivery service to display the delivery status.

The Web Viewer is frequently used in a tab control. It fills with web data when necessary—for example, in a tab control, when its tab is clicked. In general, Web Viewers do not unduly slow down FileMaker, but they do involve Internet access and imaging, both of which can take time. There is no cancel button or feature for the Web Viewer: You simply go to another record, another layout, or another tab. The current Web Viewer will stop loading, and if another one is visible, it will load.

Exploring the Web Viewer in Contacts

You can see Web Viewers in action in Starter Solutions such as Contacts. Figure 13.1 shows an address being typed into the address field of a new record. As soon as you tab or click out of the address field, the Web Viewer maps the address.

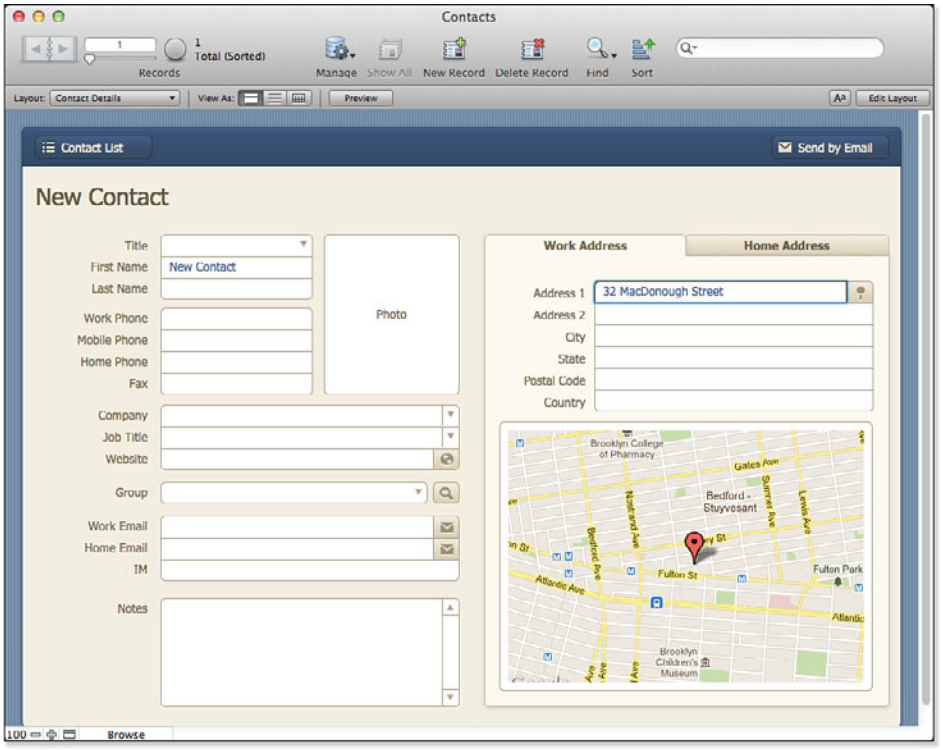


Figure 13.1 Start to map an address.

As you tab out of another field, the map may shift, as shown in Figure 13.2.

note

This behavior of the map is part of the mapping software. In Figure 13.1, it assumed New York City; the correct city (Plattsburgh) is mapped in Figure 13.2. This lets you see that this map is truly being drawn in real time based on the database fields. Your own tests with streets, cities, and states may vary. Main Street can be mapped for many states and for many localities within a given state.

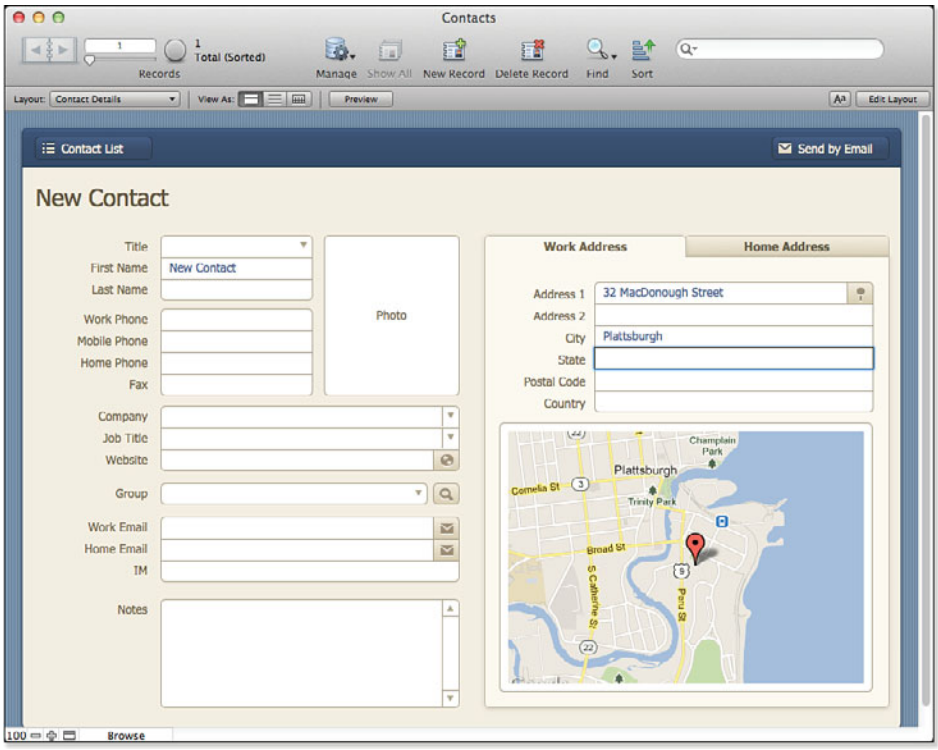


Figure 13.2
Add the city to the address.

Creating and Editing a Web Viewer

A Web Viewer can display a specific URL that you hard-code in the layout, or it can display dynamic data from the database. Furthermore, you can use a variety of built-in templates in the Web Viewer to construct URLs for a variety of common web destinations such as FedEx tracking, Wikipedia, and the like. All of these techniques are described in this section.

Creating a Web Viewer

To add a Web Viewer to a layout, begin by entering Layout mode. Then, select the Web Viewer tool, as shown in Figure 13.3, and draw the Web Viewer just as you would draw a tab, portal, or other object. Alternatively, you can use Insert, Web Viewer.

Either way, the Web Viewer Setup dialog shown in Figure 13.4 opens.

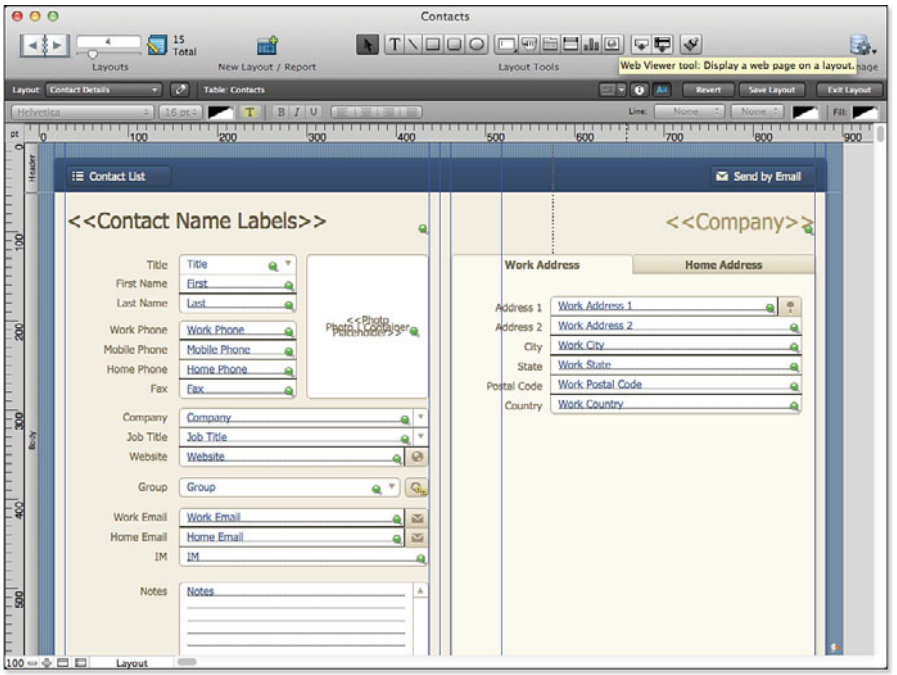


Figure 13.3
Add a Web Viewer to a layout.

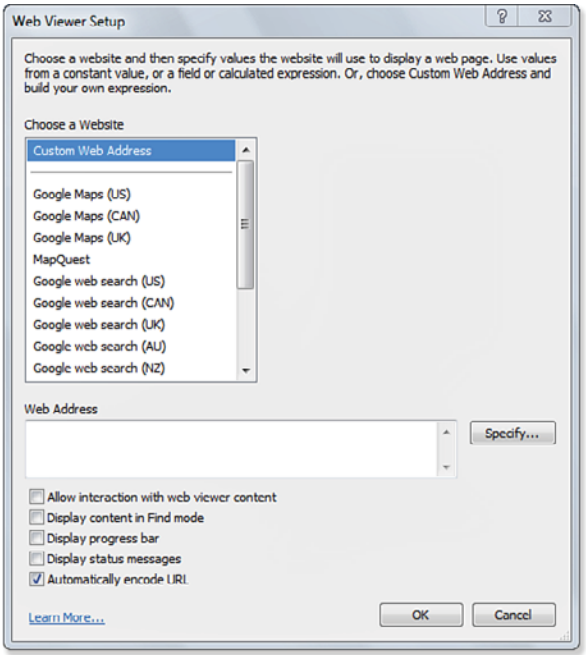


Figure 13.4
Set the web address.

Setting a Web Viewer to a Constant URL

To open a constant URL, simply type it into the Web Address field. When you go back to Browse mode, it opens automatically, provided you have an Internet connection.

*The check boxes at the bottom of the Web Viewer Setup dialog are described later in this chapter in the "Setting Web Viewer Options" section, **p. 373**.*

Constructing a URL Dynamically Based on a Search

Another way to build a web address is to interactively search for the data you want to retrieve. For example, you will see a FedEx template, but there isn't a corresponding UPS tracking template. Go to the UPS site and experiment with tracking a package (you'll need a valid UPS tracking number to do this). When you have retrieved the data, copy the URL that was generated and use it in a calculation to construct the URL you want to create. Use the calculation text functions, such as `Left`, `Middle`, and `Right`, to replace the tracking number you searched for with data from a field in your database.

Setting Up a Web Viewer with the Templates

The templates in the Web Viewer Setup dialog let you construct complex URLs from templates and specific database fields. You choose the template you are interested in, and the fields it contains are shown at the right of the dialog, as you see in Figure 13.5. For each field, you can select a database field to use or create a calculation for the field.

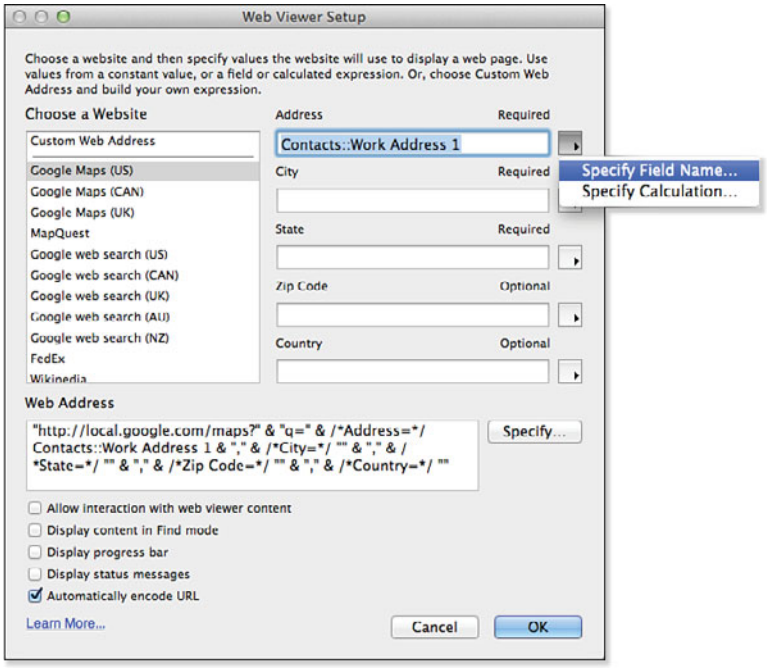


Figure 13.5
Explore fields in a template.

The default Google template appears quite complex, but if you click the Specify button next to the Web Address calculation field, you can open the Specify Calculation window and add some line feeds to rearrange the calculation, as shown in Figure 13.6.

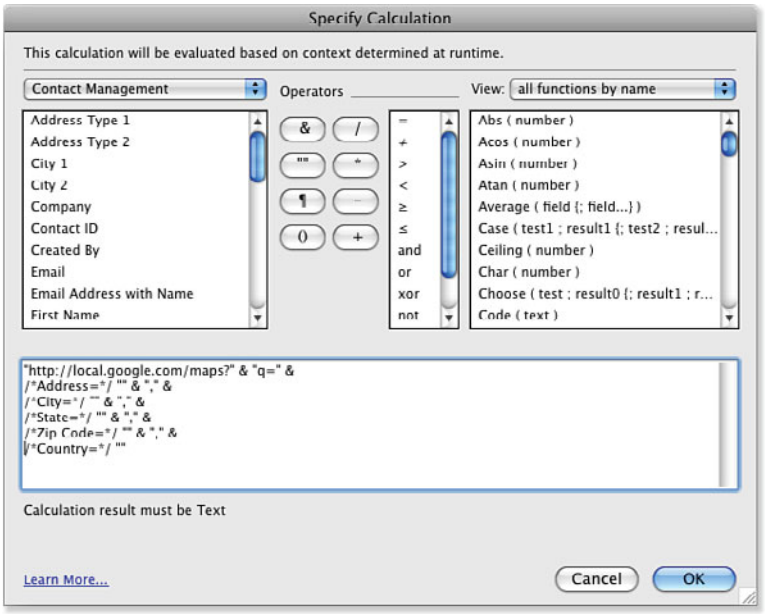


Figure 13.6
Make the Google mapping template more readable.

Note that all that has been done here is some reformatting. You can see that prompts enclosed as comments (`/*Address=*/`, for example) have been inserted into the calculation. Because they are comments, they have no effect on the calculation's evaluation. In fact, if you enter no data at all, this complex calculation does, in fact, produce a Google map.

You can remove the comments and adjust the calculation so that it uses the Contacts Starter Solution fields to map the address already specified in Contact Management, as shown in the following calculation code:

```
"http://local.google.com/maps?" & "q=" &
"Address=" & Contacts::Work Address 1 & "," &
"City=" & Contacts::Work City & "," &
"State=" & Contacts::State
```

Whether you modify the web address with the fields at the right or by typing in a calculation for the web address, the result is the same: The Web Viewer displays the data as you specified.

tip

Keep in mind that you can specify a URL that does not exist or that contains an error, so be careful when you construct the URL, and always test the resulting Web Viewer.

Setting Web Viewer Options

You can set five Web Viewer options, shown at the bottom of the Web Viewer Setup dialog shown previously in Figure 13.4:

- **Allow Interaction with Web Viewer Content**—If you turn off this option, the web page displays but no links are active. This can be the appropriate setting if the web page contains static information—for example, a known site's known page containing contact information. You can "trap" a user there, and the user will not be able to wander off to other places on the Web.
- **Display Content in Find Mode**—By default, this option is off. When the user is in Find mode, it is often the case that the Web Viewer should be blank. When a Find operation completes, its result can be used to display data in the Web Viewer. However, if the Web Viewer is based on a global or on data that will be entered during the Find setup, you might want to make the Web Viewer active at that time.
- **Display Progress Bar**—This feature provides a small progress bar across the bottom of the window. It is on by default.
- **Display Status Messages**—This option controls whether status messages appear. It is on by default.
- **Automatically Encode URL**—Sometimes, URLs contain characters that are not part of the standard ASCII character set. In such cases, the characters are encoded by specifying the bytes as a number that follows an ampersand (&) or a pound sign (#). FileMaker uses standard encoding practices so that when a space follows & or a number follows #, nothing is changed because these combinations specify nonprinting characters. When a space does not follow & or a number does not follow #, these are assumed to be literal characters, and they are encoded. Typically, you want to enable this option.

Controlling the Web Viewer with the `SET WEB VIEWER` Script Step

The Web Viewer itself provides no controls, but you can add your own controls. The addition of the Web Viewer to FileMaker layouts was one of the catalysts for providing object names. To control a Web Viewer, you must name it in the Position tab of the Inspector.

Then you can use the `Set Web Viewer` script step in ScriptMaker to control it, as shown in Figure 13.7.

Many of the options are simple—Go Forward, Go Back, and so forth. Reset sets the Web Viewer to its initial URL. You also can specify a URL to go to. If you choose the Go to URL option, the Web Viewer Setup dialog (shown previously in Figure 13.4) opens.

Figure 13.8 shows an all-purpose script that you can use for Web Viewer controls. It takes a variety of parameters. Depending on which one is sent, one of the standard actions is performed. `Go` is used to go to a URL entered in a database field.

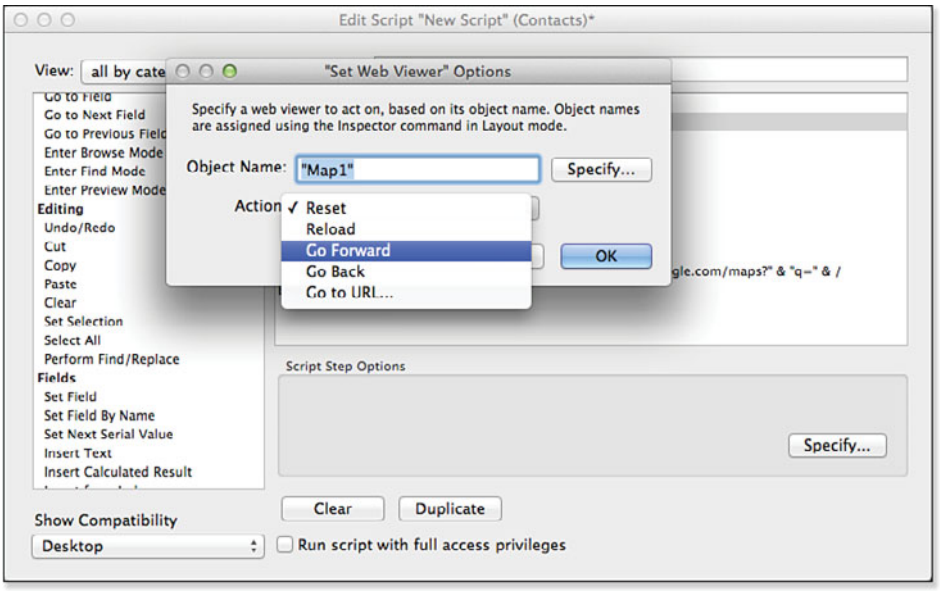


Figure 13.7
Use the `Set Web Viewer` script step.

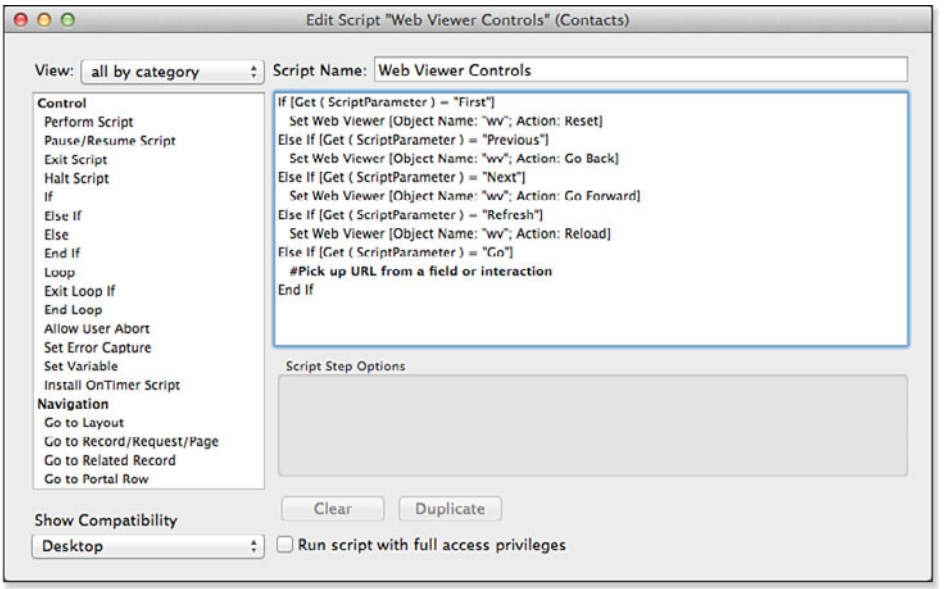


Figure 13.8
Use a script to control the Web Viewer.

*For more information on script parameters, **see** Chapter 16, "Advanced Scripting Techniques."*

*For more information on the Inspector and auto-resizing, **see** Chapter 4, "Working with Layouts."*

FileMaker Extra: Using the Web Viewer for Files

Any URL that can be processed by the appropriate control can be displayed in a Web Viewer. That means you can use a File URL to display files in the Web Viewer. This capability provides an additional way of displaying graphics files in FileMaker. You can add them to container fields themselves, or you can place references to them in container fields. You can also store the file location in a database field either as a File URL or as the filename part of the File URL. Then you can construct a File URL as you want.

For example, Figure 13.9 shows a Web Viewer with a calculated URL based on a path, folder, and filename (the file is one of the illustrations from this book). The calculated filename is shown in Figure 13.9, but you can simply place that calculation inside the Web Viewer itself. If you change the field containing the server address or path, you will quickly pick up another file.

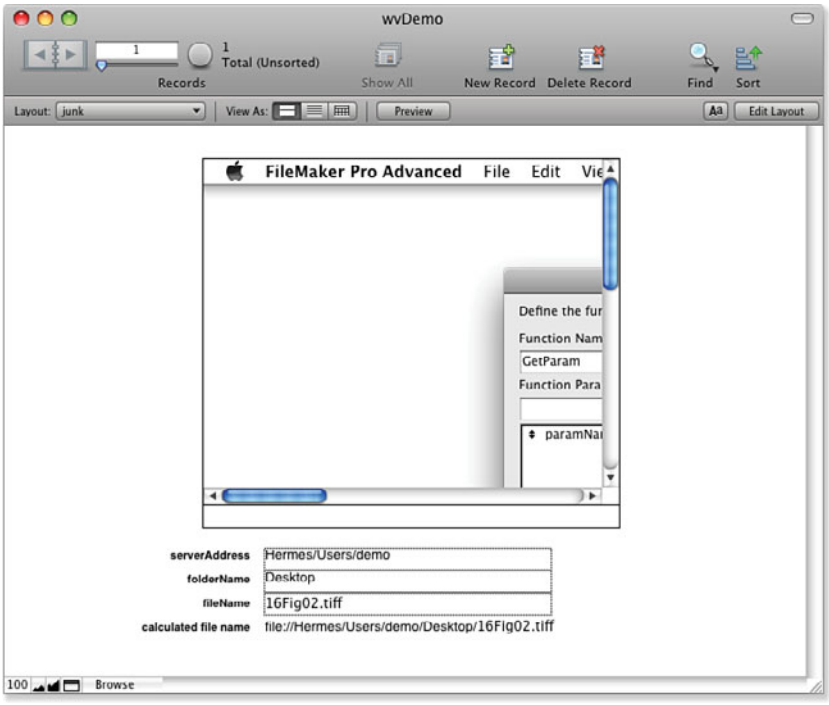


Figure 13.9
Use File URLs to display a file in a Web Viewer.

This technique handles an issue with file references in which they break when the files move. In a case like this, you can store the prefix in a global or database field and then concatenate it with the unchanging filename, as necessary.

This page intentionally left blank

14

ADVANCED INTERFACE TECHNIQUES

What's New in the Interface World

Arguably, the biggest event in the world of computer interfaces occurred on April 3, 2010 when Apple shipped the first iPad. Although the iPhone actually pioneered many of the interface changes that reached fruition in the iPad, the size and shape of the iPad quickly got people to think that this wasn't a new kind of phone but rather a new kind of computer. Compared to existing computers, the iPad was as powerful (or even more powerful) than many of them. What's more, the iPad's powerful battery meant that it was more portable than even laptop computers, which had seemed so svelte only a few years before.

People also noticed what the iPad did not have that desktop and laptop computers did: connections for external drives, expandable memory, ports to connect devices such as digital cameras, and the like. Rather than limit the iPad's usability, Apple's focus on its core functionality was arguably a wise choice.

One of the most important parts of the desktop computer interface—menus—was also missing on the iPad. The iPad caught on very quickly, and people soon used it for a wide variety of tasks. For most people, it just worked, but others examined the interface thoroughly and noticed differences from previous interfaces. (Many of these differences were based on Apple's Human Interface Guidelines.) As you'll see in the list that follows, many of the key features of the new interface are old news to FileMaker users:

- There are fewer buttons. Many things on the iPad happen automatically. For example, popovers present choices and information that

on OS X might have been presented in a dialog with an OK and a Cancel button. There is no Cancel button in a popover; you just tap anywhere outside the popover and it disappears. With FileMaker, triggers implement automatic behaviors eliminating the need for buttons in many cases. (In the Projects Starter Solution, a pop-up menu lists layouts you can choose. There's no Go button next to the pop-up menu: instead, an `OnObjectModify` trigger takes you to the layout automatically.)

- In most apps, there is no Save command. Saving is done for you automatically. This behavior has carried back to the Mac with OS X 10.7 (Lion). FileMaker saves data automatically and has done so for years (as do many databases).
- Information structures are flattened. To set values, instead of a sequence of windows as you might have on a desktop, you often have a view with tabs or other controls that let you see data from various sources and relationships all in a single layout.
- Finally, apps are very attractive. In everything from icons to the apps themselves, Apple has placed a high premium on appearance. Because the company approves all apps sold in the App Store, it can enforce that premium.

*For more information on interface design, **see** Apple's iOS Human Interface Guidelines and Mac OS X Human Interface Guidelines at developer.apple.com.*

The next section discusses the themes that were implemented in FileMaker Pro 12.

note

In practice, it appears that without having to take much action, Apple has been able to encourage developers to improve the look of their apps just by letting them know that esthetics will factor into the decision of accepting an app into the App Store. Many FileMaker solutions are beautiful and elegant to look at because they have been created by talented designers using the very flexible FileMaker layout tools. Other FileMaker solutions are not quite so beautiful to look at.

Working with Themes

Themes provide you with a variety of designs for your FileMaker solutions. They start you off with colors, fonts, field borders, and other graphic elements that are coordinated for the theme. Most important, themes are interchangeable.

If you create a new database, you will find that you have a table and a layout created for you automatically. Both carry the name of the database file. Thus, in Figure 14.1, you see a new database called UIDatabase and its UIDatabase layout. No records are present in the database.

The underlying table is also named UIDatabase. The Manage Database dialog shown in Figure 14.2 lets you see the table as well as the fact that no fields are yet defined for it.

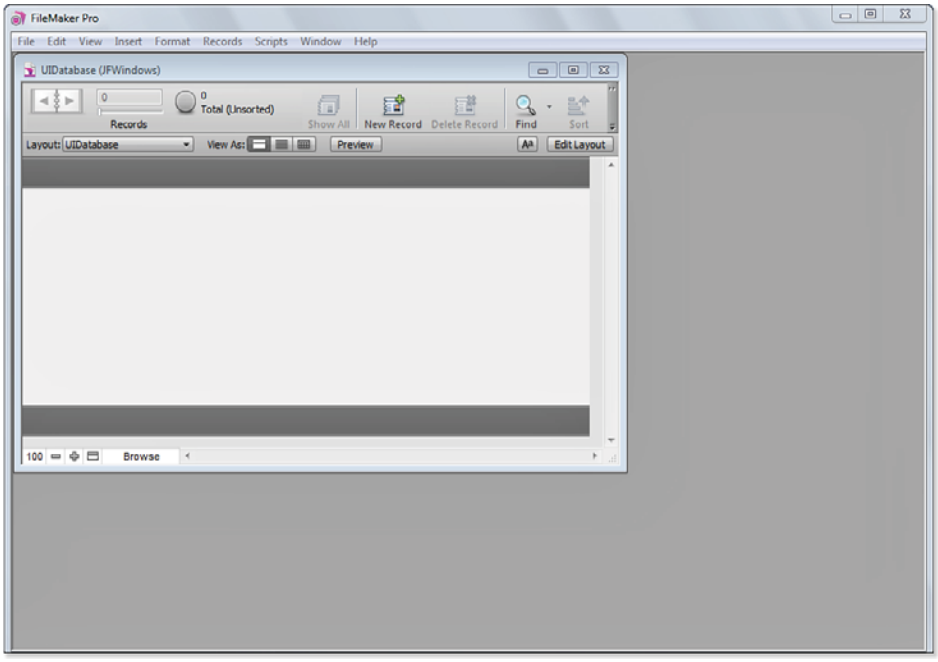


Figure 14.1
Create a new database.

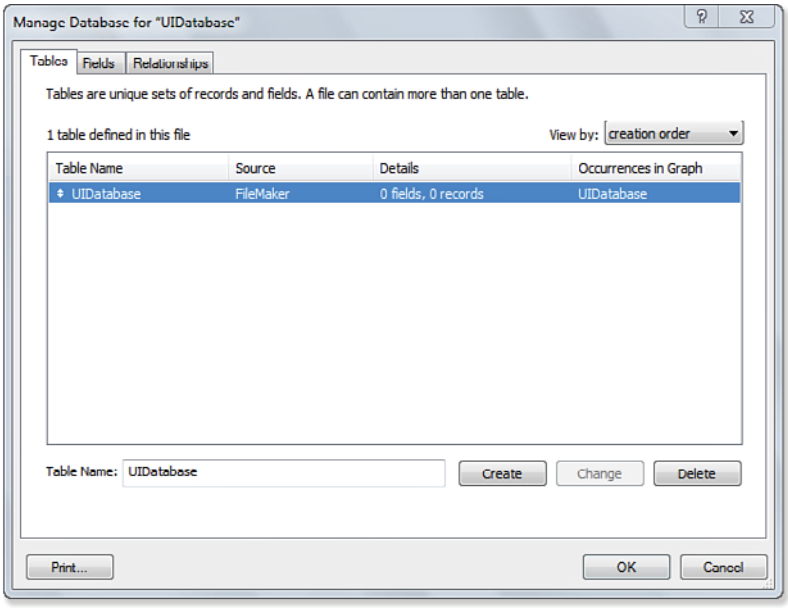


Figure 14.2
Review the database.

Changing a Theme

Not only do themes give you a head start on the design of your layouts, but they are changeable. The layout created in Figure 14.1 uses a default theme called Cool Gray. You can examine the current theme in Layout mode by selecting Layouts, Change Theme, as shown in Figure 14.3.

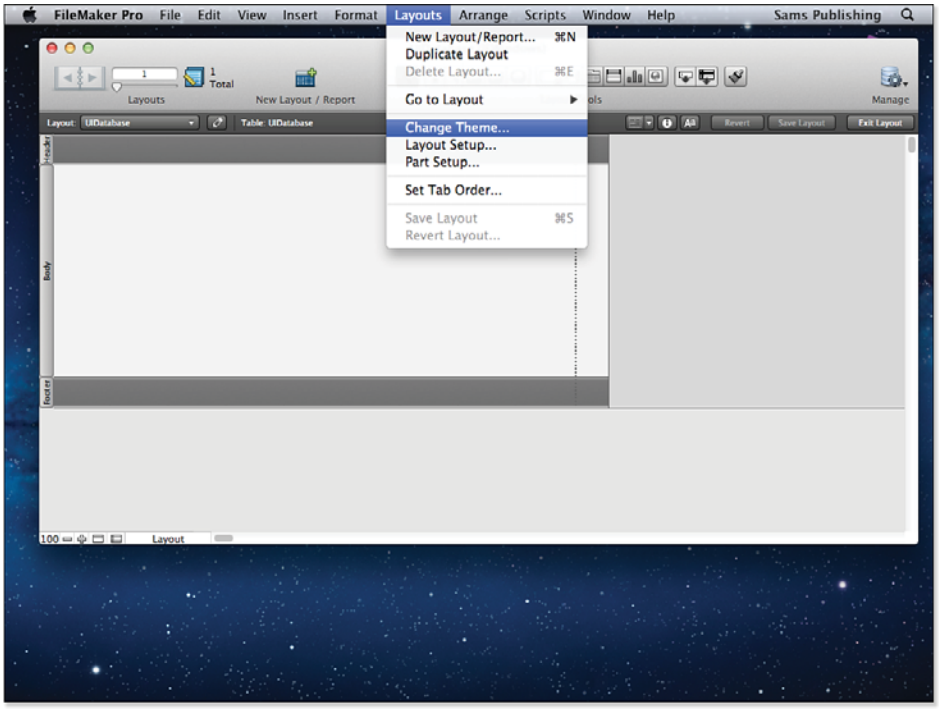


Figure 14.3 You can change themes.

As you can see in the Change Theme dialog shown in Figure 14.4, you have a lengthy list of themes from which to choose. They are grouped together in several collections. As you select various themes from the list, the sample layout at the right of the dialog changes so that you can get a general idea of the effect. However, until you actually start laying out the fields and other interface elements you will be using, you won't be able see exactly what your layout will look like in this preview.

At the top of the list of themes, in the Basic group, you find the Classic theme along with two variations of it in the Classic Refined group: Cool and Warm. Figure 14.5 shows the Classic theme selected. If you have used FileMaker in versions prior to FileMaker 12, you will recognize the default fonts and styles.

Figure 14.4
Choose themes and see previews.

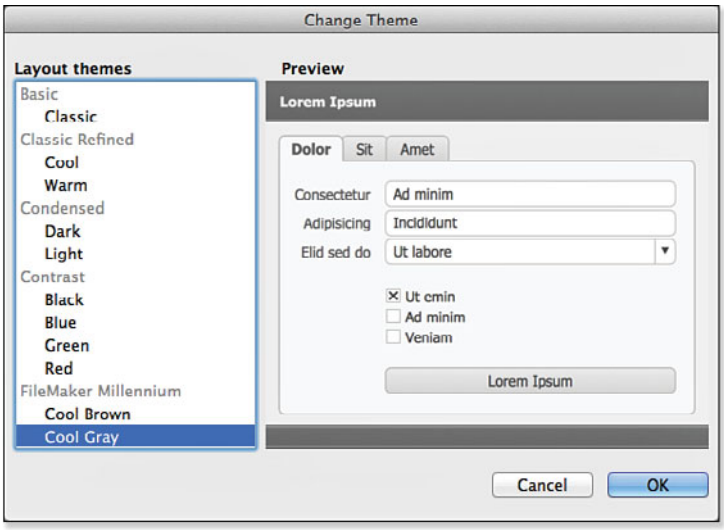
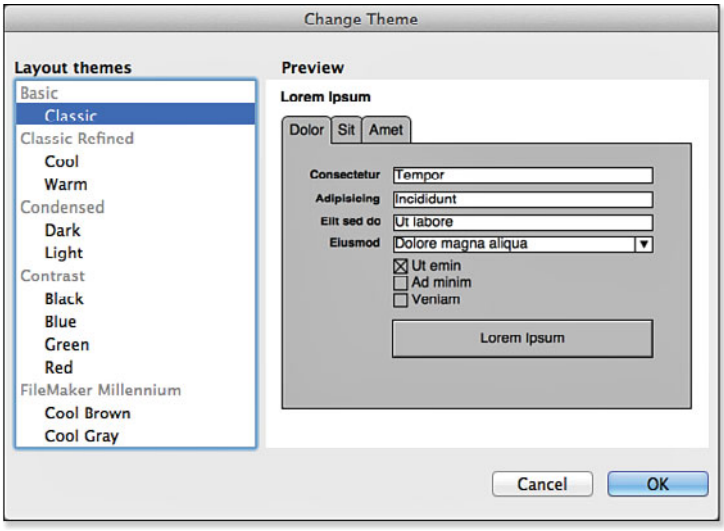


Figure 14.5
Change a theme.



As you explore the themes, you will note that some of them are designed specifically for touch devices. For example, just below Cool Gray you will find Cool Gray Touch, as shown in Figure 14.6. Note that the buttons and fields are larger in this theme.

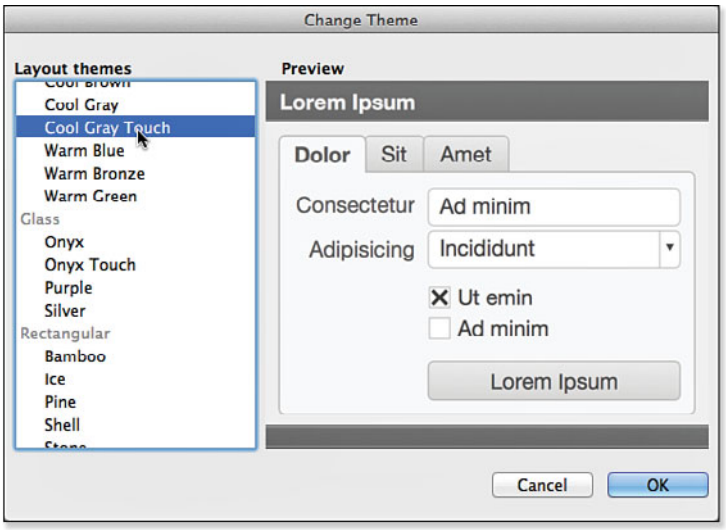


Figure 14.6
Explore touch themes.

Exploring Themes

To demonstrate themes in action, Figure 14.7 implements a few additions to the database:

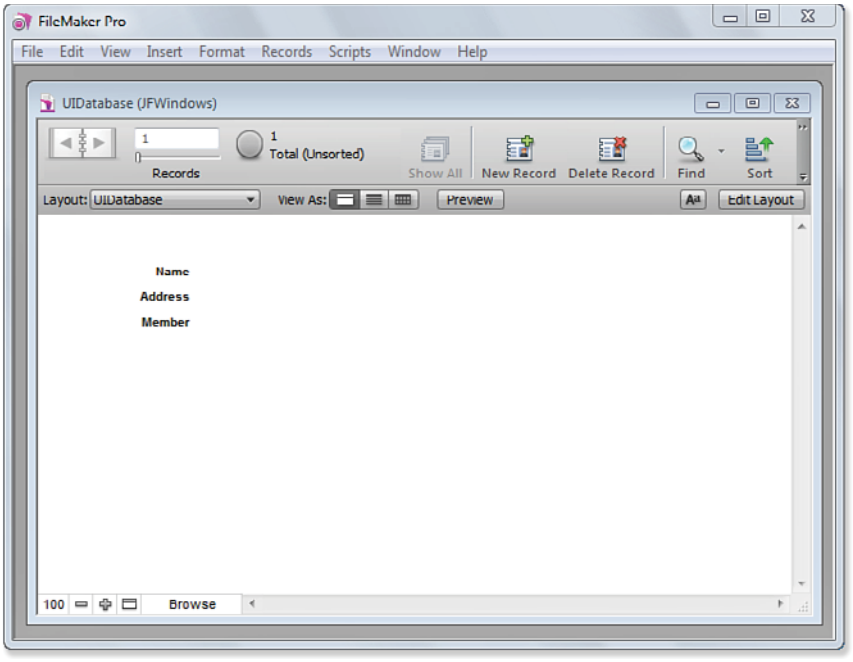


Figure 14.7
Add fields to the database.

- The text fields Name, Address, and Member are added to the table using File, Manage, Database.
- Default fields and labels are automatically added by FileMaker.
- A value list called YesNo, with the values Yes and No, is added using File, Manage, Value Lists. (It is not yet visible in Figure 14.7.)
- Using the Inspector, the Member field is changed to a radio button set that uses the YesNo value list, as you see in Figure 14.8.

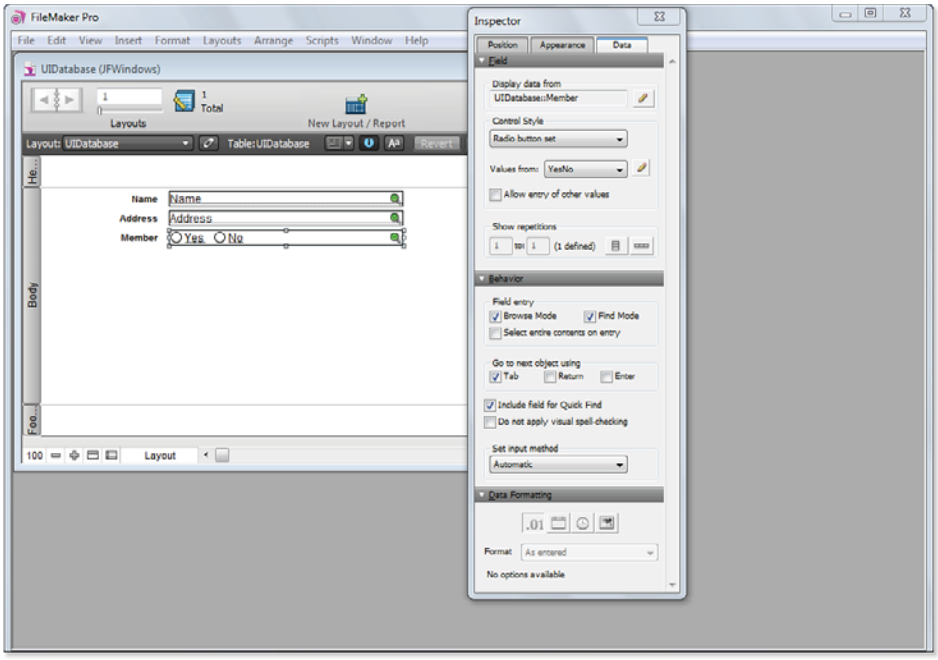


Figure 14.8 Change a field to a radio button set.

You can now change the theme. Use Layouts, Change Theme to change your theme back to Cool Gray, which was shown originally in Figure 14.1 before the fields were added.

As you can see in Figure 14.9, if you add a new field to the database and a new field to the layout, the new field reflects the sizing of the theme. At the left, the Name field has a height of .236 while at the right, the City field has a height of .306. The font (Tahoma) and size (12) are the same in both versions. If you have followed the steps in this section, the Name field (along with Address and Member) was created with the Classic theme. Classic uses Arial 12 for text fields, so the .236 height of the fields works out well.

tip

If you look closely, you will see that when the fonts change for the added fields, the spacing is a little off. This is because when you modify a theme, such as by adding a field to the layout, those modifications are not reflected when you subsequently change themes.

However, because Tahoma 12 is slightly larger than Arial 12, that text field height is no longer satisfactory when the theme is changed to Cool Gray.

Because you might have to make some adjustments after you change a theme, it is a good idea to experiment with themes using a few layout fields without bothering with the tweaks to field sizes. Select the theme you want, and then either tweak the existing fields or delete them and create new ones in the new theme.

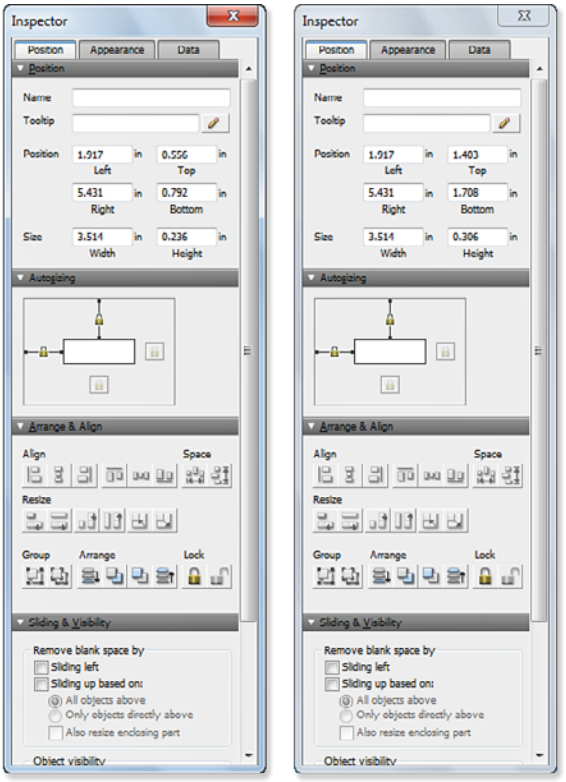


Figure 14.9
Compare field sizes with different themes.

Using Styles and States

Along with themes, FileMaker Pro 12 has implemented *styles* and *states* in the Inspector and design surface. These features enable you to build more sophisticated layouts that respond in more ways to user actions. You adjust styles and states in the top part of the Appearance tab of the Inspector, as shown in Figure 14.10.

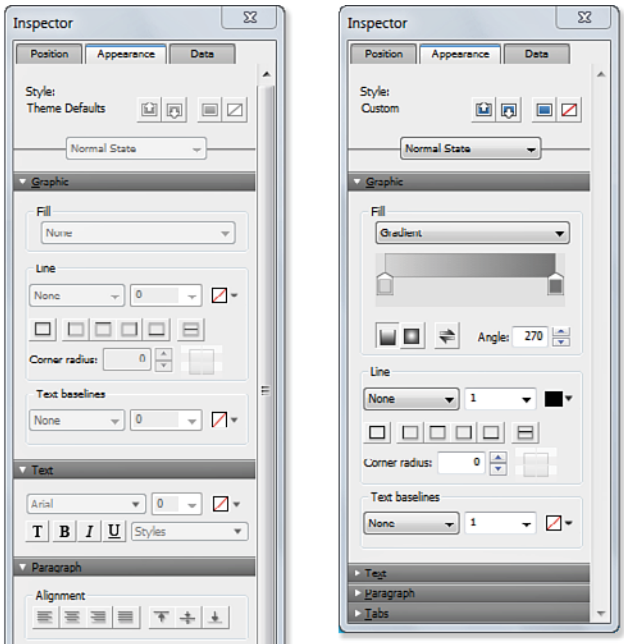


Figure 14.10
Adjust styles and states.

Using Styles

The Style section in the Inspector, shown on the left in Figure 14.10, is disabled there. No interface elements are selected and the theme defaults are used. That is not something that happened prior to FileMaker Pro 12. Before FileMaker Pro 12, changes to attributes, such as fonts and graphic fills that were made when no field was selected, were applied to what is now the theme. The new settings took effect for the next objects that were created. With the advent of themes and styles, this behavior no longer exists.

Instead, you can select an object in the layout and set its styles and states: You can no longer set attributes for objects that have not yet been created (that is the role of themes). The right side of Figure 14.10 shows the Appearance tab of the Inspector when an object in the layout is selected. Note that the Style section is now enabled.

The left side of Figure 14.10 shows that the style that will be used for new objects is Theme Defaults. All objects in a newly created layout start with the theme's default style. You can change attributes for the specific objects in your layout using the Inspector.

Using Gradients and Rounded Corners

Figure 14.10 shows several other features of the Appearance tab. You can fill an object with a color or an image, but you can also choose a gradient. You can select the two colors in the gradient by clicking the markers at the beginning and end of the gradient in the Inspector. You can also choose linear and radial gradients as well as the direction of the gradient.

Note also in the line section that you can now adjust the radius of rounded corners.

If you change the theme with Layouts, Change Theme, the style at the top of the Inspector is displayed as Theme Defaults. Objects you had customized before changing the theme (that is, objects with custom styles rather than Theme Defaults) now revert to Theme Defaults rather than the custom styles you created. Figure 14.11 shows a field that had been customized as it appears after a theme change.

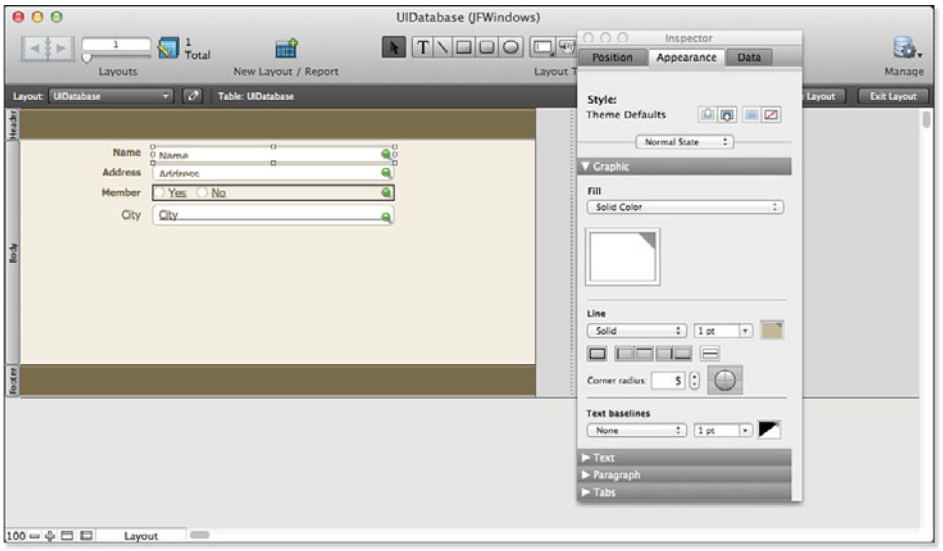


Figure 14.11 After a theme change, styles revert to theme defaults.

Often, this is the correct behavior: You want to adopt the theme in its entirety. However, sometimes you want to keep your customized styles for objects. To do that, use the Edit, Undo Styles command. Your previous customized styles will appear back, as you see in Figure 14.12.

You can use Edit, Undo Theme to revert to your previous theme if you want. What you can see from this behavior is that the theme and your customizations are changed in two separate tasks in the Change Theme command. They are done together, but you can undo them separately using the commands or the standard undo keyboard shortcut (Cmd-Z)[Ctrl-Z] twice.

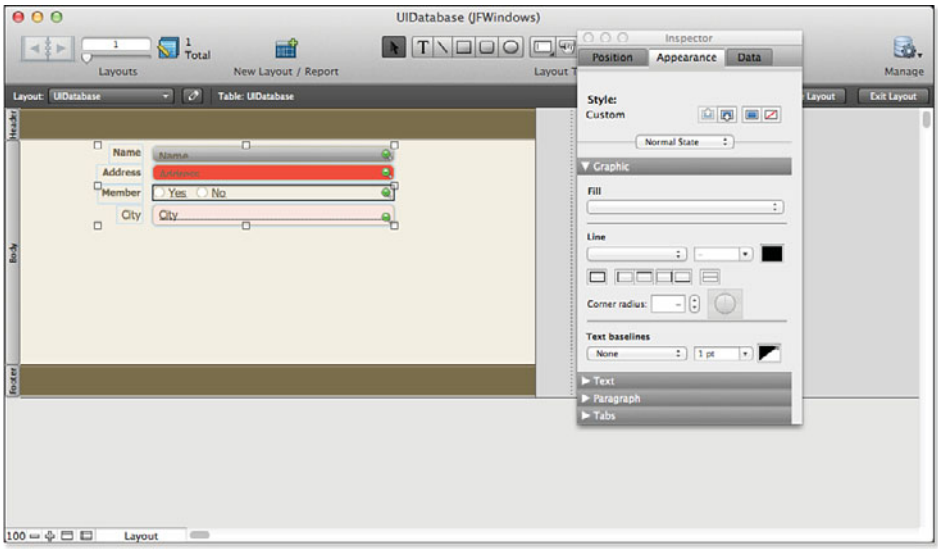


Figure 14.12 Use Undo Styles to revert to your previous customized styles.

Using States

States let you specify attributes for the selected interface element using the Inspector. These let you change the appearance of the selected element(s) based on what the user is doing in the interface. The pop-up menu shown in Figure 14.13 enables you to set four separate sets of state attributes:

- **Normal State**
- **In Focus**—This state is used when you have entered a field with a mouse click or tap, the Tab key, or the `Go to Object` script step.
- **Hover**—This state is unavailable on iOS devices because there no such thing as hovering.
- **Pressed**

The undo mechanism described in the previous section is frequently used with states. If, for example, you have created a pressed state attribute for a button, you might want that appearance to be brought over to the new theme. Just change the theme, which will revert to the Theme Defaults, but immediately after that use Edit, Undo Styles to bring over your pressed-state style. Do not use Edit, Theme after that so that you get the new theme together with the old state appearance.

tip

Experiment with styles for the various states, but also observe how apps behave. Often, very subtle changes are sufficient to let the user know that the button has been pressed. Remember, your FileMaker Pro solution is about the solution and its data rather than a never-ending parade of visual effects. Often, in usability labs, users will emphatically deny that anything about an object's appearance has changed with a change to state. The subtle (and often best) distinctions might not even register consciously, but users see and respond to them. Finding the right balance between subtle yet perceptible and grotesque state appearances can take a good deal of testing, but it can make your solution look better and be easier to use.

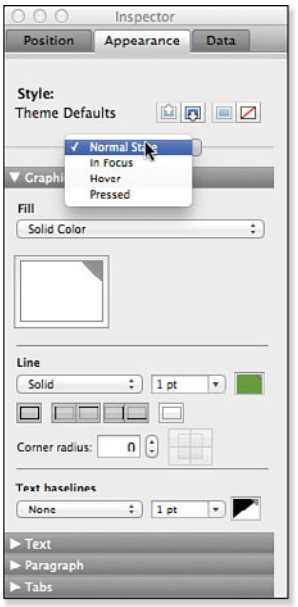


Figure 14.13
Set state attributes.

Copying Styles

When an object is selected in Layout mode, four buttons at the top of the Appearance pane may be enabled (they are just to the right of the style text—either Theme Defaults or Custom). They correspond to four commands in the Edit menu in Layout mode:

- Copy Object Style
- Paste Object Style
- Apply Theme Style
- Remove Styles

Copy Object Style

This command is available if you have set a custom style for the selected object (that means it is not available if the object's style is Theme Defaults). Copying the object's style with the menu command or the button at the top right of the Inspector copies all the attributes for all the states of the object.

Paste Object Style

This button and the corresponding menu command are enabled if you have copied an object's style and have also selected an object to edit. If so, the attributes for all the styles are pasted to the selected object.

Apply Theme Style

If an object is selected, this command or button will set the attributes to the theme's style.

Remove Styles

This removes the styles whether you customized them or you are using the theme's styles. Fill, Line, and Baseline are set to None, for example.

Using FileMaker Formatting Tools

FileMaker Pro provides several tools to help you format the appearance of layouts and objects. A few of the most useful tools are described in the following sections.

Conditional Formatting

Using styles and states, you can adjust the appearance of layout objects based on the state of the interface. Conditional formatting lets you adjust the appearance of an object based on its content.

In Layout mode, select the field or text object to which you want to apply conditional formatting, and choose the Conditional command from the Format menu to open the dialog shown in Figure 14.14.

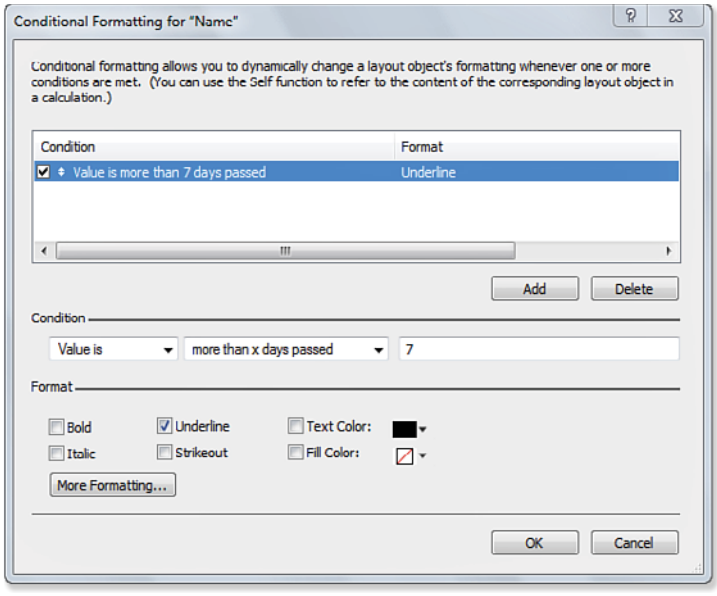


Figure 14.14
Use conditional formatting to provide user feedback.

You can set two choices. One is to set the condition based on the value of the field. To do this, select the test shown in Figure 14.15 and type in a value to use in the test.

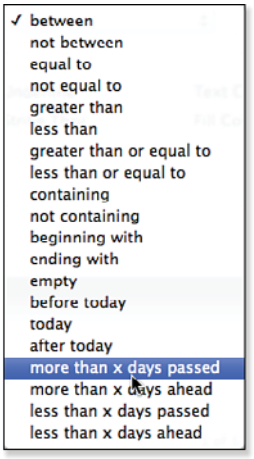


Figure 14.15
Select the test to perform.

You also can specify a formula to evaluate, as shown in Figure 14.16.

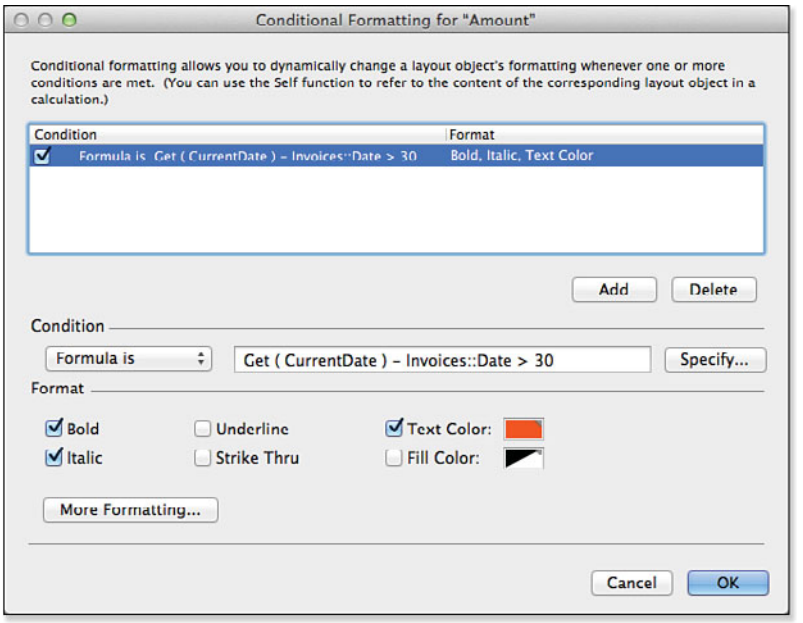


Figure 14.16
You can supply a formula for the test.

*To learn about the basics of calculation functions, **see** Chapter 8, "Getting Started with Calculations."*

*For detailed examples of text-formatting functions, **see** Chapter 15, "Advanced Calculation Techniques," and Chapter 19, "Debugging and Troubleshooting."*

Setting the Layout Width

In addition to themes and styles, FileMaker Pro 12 introduces the ability to set the width of a layout. Until now, you did not worry about the width of a layout: It automatically was sized to be wide enough for all the objects you placed on it. In other words, the right side of the layout was the right side of the rightmost object in the layout.

You now have the ability to explicitly set the layout width. As you see at the right of Figure 14.17, you can drag the right edge of the layout from side to side. Objects that are placed entirely beyond the layout's right edge are not visible in Browse mode, but they are still there and you can reference them in scripts if you want to.

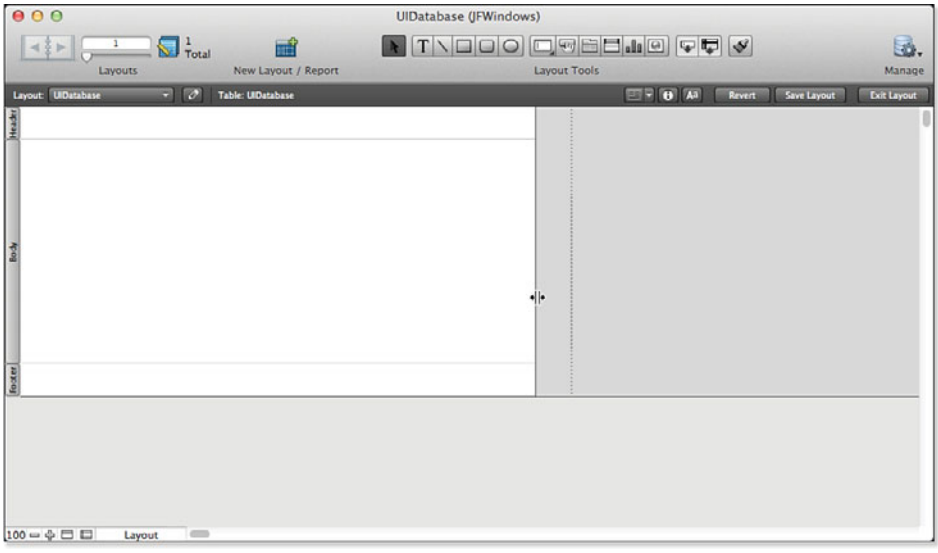


Figure 14.17 Adjust the layout width.

Using Grids

To help position interface elements consistently, you can turn on *grids* in Layout view. Use View, Grid, Show Grid and View, Grid, Snap to Grid to control the behavior of the grid. As shown in Figure 14.18, show rulers (View, Rulers) and set the units to inches, centimeters, or points.

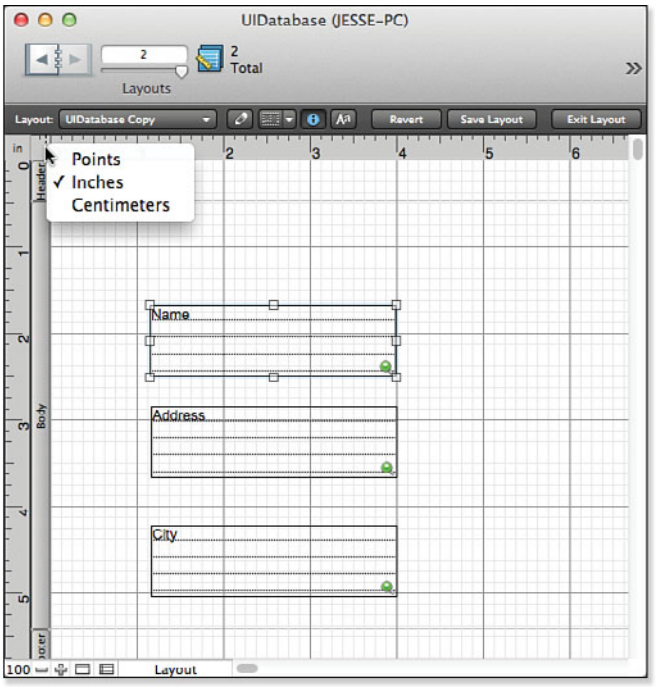


Figure 14.18
Choose grid units.

Note that beginning with FileMaker 12, the use of pixels as a measurement is deprecated in favor of points. Pixels are not a stable unit to use in the world of resolution independence. (See the "Points Versus Pixels" sidebar, later in this chapter.)

If you choose View, Grid, Snap to Grid, objects snap to the grid as you drag them. If you position them by nudging them with the arrow keys or by setting their location with the Inspector, the grid does not come into play.

Using Guides

Guides let you set a vertical or horizontal location to use as a reference point. You create a guide by dragging from the ruler at the left or top of the design surface in Layout view. Position it where you want regardless of whether it is a grid unit. You can control behavior for guides just as you do with grids. The commands are View, Guides, Show Guides and View, Guides, Snap to Guides. Just as with grids, snapping applies to dragging objects rather than nudging them or setting their location with the Inspector.

Using Dynamic Guides

Turn dynamic guides on and off with View, Dynamic Guides in Layout view. As you move objects, guides appear when they are aligned with adjacent objects based on an edge or the middle, as shown at the left in Figure 14.19. In that case, the lower text field (Address) is being moved. The dynamic guide shows when it is aligned with the middle of the Name field above it.

If you select multiple objects, they move as a group, as you see at the right of Figure 14.19. As you drag, the position of the grouped objects appears. Furthermore, you can drag any of the resizing knobs to resize all the selected objects.

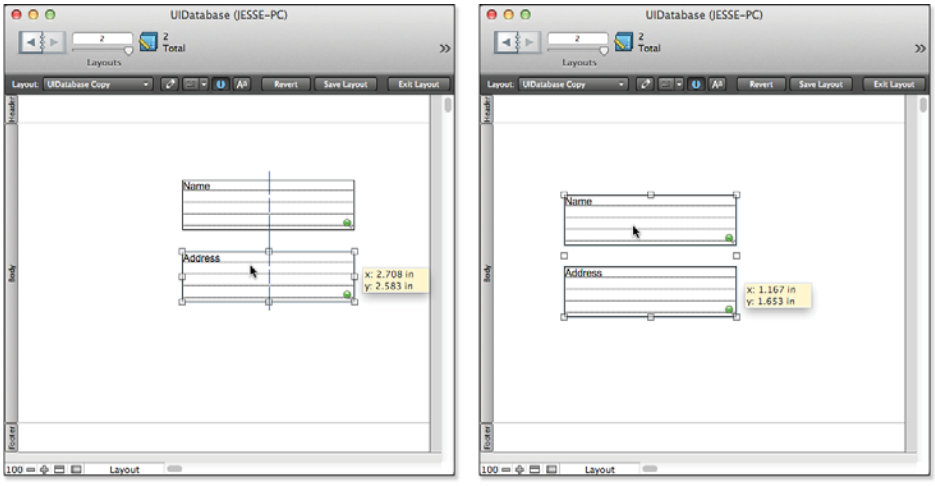


Figure 14.19 Use dynamic guides.

Using Screen Stencils

In addition to the guides and grid discussed previously, beginning with FileMaker Pro 12, you can use *screen stencils* to help you design your layouts.

In Layout mode, you can turn screen stencils on or off from the layout bar as shown in Figure 14.20. There are two controls. The one shown in Figure 14.20 lets you turn the provided stencils on and off. Just to its left, the button lets you show or hide stencils.

The stencils are yellow lines that appear over all your layouts in Layout mode until you turn them off. As you see in Figure 14.20, eight stencils are built in to FileMaker; you can add one more for a total of nine.

note

It is important to note that the dimensions in the names of the built-in stencils are represented in points rather than the commonly used pixels. You can verify this if you open the Custom Size dialog, as shown in Figure 14.21.

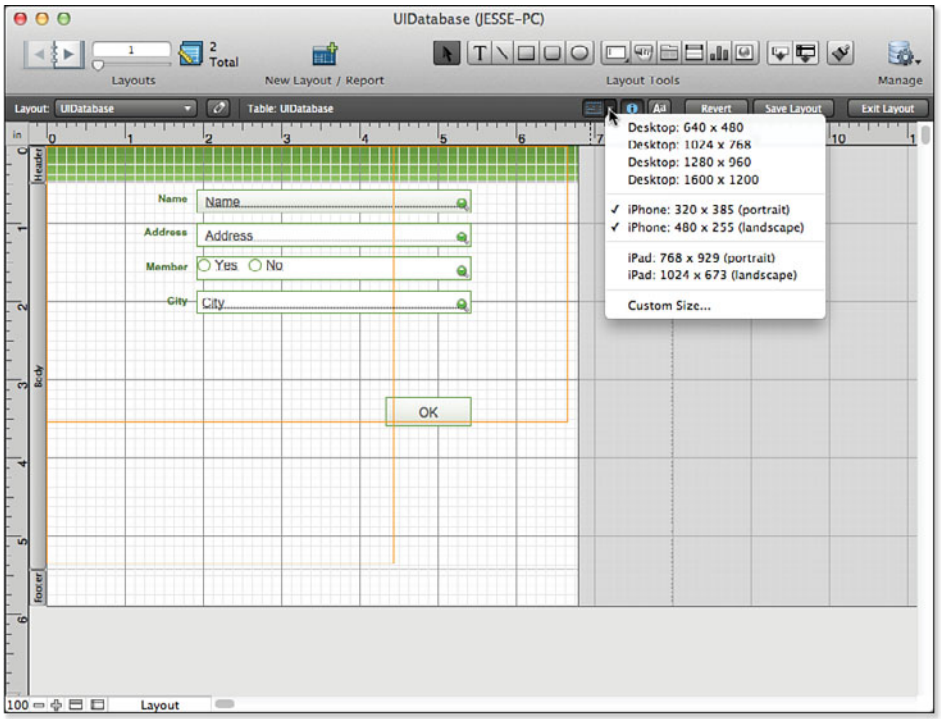


Figure 14.20
Use screen stencils.

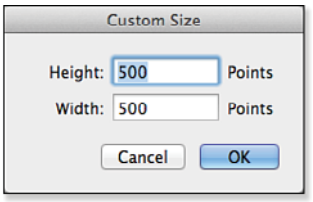


Figure 14.21
Create a custom screen stencil.

Points Versus Pixels

A point is a unit of length—specifically, 1/72 of an inch. Many people became used to talking about pixels as if they were units of length. This is because on many Apple (and other) displays, there were 72 pixels to an inch. It was not unreasonable to read 1024×768 as being measurements in pixels because that assumption turned out to be true. However, it only happened to be true when there were 72 pixels to the inch on a display.

With the advent of high-resolution screens, such as Apple's Retina Display, the resolution was achieved by *pixel doubling*. Where there had been a single pixel, there now were four pixels: two pixels across and four pixels down.

Now the distinction between pixels and points becomes important. An iPhone screen measures 320×385 points in Portrait mode whether it is a Retina Display or an earlier model. However, an iPhone screen that measures 320×385 pixels in an earlier model, measure 640×770 pixels in a Retina Display.

To measure dimensions and locations on the screen, the unit to use is points. One rule of thumb that many people use is that if you work with Photoshop or InDesign, you use pixels. If you work with Xcode or FileMaker, you use points.

Using GETLAYOUTOBJECTATTRIBUTE

The `GetLayoutObjectAttribute` function lets you interrogate layout objects about their current state. Many times you know this information already because you set these objects. However, for other objects, such as the Web Viewer, you do not necessarily know what page the user has navigated to.

The `GetLayoutObjectAttribute` function requires two parameters; it may take two additional ones. The first parameter is the name of the object, as set in the Inspector. Remember that FileMaker enforces the uniqueness of layout object names within a layout.

note

This is not a `Get` function in the sense of `Get ( ScriptParameter )`. The name of the function is `GetLayoutObjectAttribute` (with no space after `Get`).

The second parameter is the attribute you want returned. Table 14.1 shows the various attributes you can query and what the results are. All measurements are in points.

Table 14.1 Attributes for `GetLayoutObjectAttribute`

Name	Meaning (If Not Obvious)
`objectType`	Possible return values are `field`, `text`, `graphic`, `line`, `rectangle`, `rounded rectangle`, `oval`, `group`, `button group`, `portal`, `tab panel`, `web viewer`, and `unknown groups source`.
	Web Viewer: current URL.
	Field: table name::fieldname.
	Text object: text (without merge fields).
	Portal: related table name.
	Graphic: image data (Container data type, for example).
	Chart: XML description of the chart object
	Other objects: empty string

Table 14.1 Continued

Name	Meaning (If Not Obvious)
`content`	Web Viewer: HTML code. Field: data formatted as the layout object has specified. Text object: text (including merge fields). Graphics: same as source. Chart: bitmap representation of the chart object Other objects: empty string
`source`	Web viewers: returns current URL Fields: returns the fully qualified field name (table name::field name) Text objects: returns the text (does not return merge fields) Portals: returns the related table name Graphics: returns image data, such as Container data type; for all other objects, returns an empty string Charts: returns the XML description of a chart object.
`hasFocus`	True (1) if the object is active. For a portal, returns true if any row is selected.
`containsFocus`	True (1) if the object is active or contains an `isFrontTabPanel` active object.
`bounds left right top bottom width height`	Left, top, right, bottom.
`rotation`	Rotation in degrees of the object.
`startPoint, endPoint`	Lines: pair of values (horizontal/vertical) for start or end point.
`enclosingObject containedObjects`	The result is a list.

One of the common uses of this function is to get the HTML that is currently displayed in a Web viewer page to which the user has navigated.

Working with Custom Menus

To modify the menu sets in FileMaker, you have to develop using FileMaker Pro Advanced; however, anyone working with FileMaker Pro (or a bound runtime solution) can utilize the custom menus you create. The custom menus feature dramatically alters the user interface landscape for FileMaker: Developers can now control menus beyond simply turning them off and can drive a great deal of application logic.

Suppose that you have built a solution with a section for customers and another for orders. Assume that the system is somewhat complex and that you, as a developer, do not want users creating new order records or new customer records by selecting New Record from the Records menu in FileMaker—perhaps new record creation needs also to create child records in parallel or do some other bookkeeping within your system. Instead, you want scripts that you've written to manage the creation of these important records.

Using custom menus enables you to do the following:

- Change the names of menus or menu items
- Override or extend the functionality of native FileMaker menu items with your own scripts
- Change or add keyboard shortcuts to existing or new menu items
- Disable or remove individual menu items or entire menus
- Load custom menu sets on demand or tie menu sets to particular layouts, modes, or operating systems

note

In FileMaker Pro Advanced, you can create your own menu set and control at the most granular level when menu items appear, how they work, and even what keyboard shortcuts they use. Such customizations, if made, apply throughout the FileMaker interface, affecting contextual menus, the close box on Windows systems, and potentially every menu item in FileMaker.

Before delving further into custom menus, you have to be clear on the nomenclature used. There are four separate elements to consider when working with custom menus:

- **Menu Item**—This is a single item on a menu. A user can often select it, as is the case with the New Record and Save As menu items. Menu items can also be separators or submenus.
- **Command**—Commands refer to the native controls "baked into" FileMaker: They perform an action in FileMaker. In FileMaker's standard menu set, all the actions are predefined by the application—for example, entering Find mode or opening the Help system. You can also create custom commands that initiate a script you, as a developer, will have written. It is possible to tie a menu item to a command in FileMaker and thus offer the native functionality that the command controls.
- **Menu**—A *menu* is a collection of menu items and comprises the full set of menu items available, regardless of whether they are all active or visible in any given mode or situation. Examples of menus are File and Edit. The File menu starts with the New Database, Open, and Open Remote menu items.
- **Menu Set**—A *menu set* is the set of all menus currently active or potentially available in FileMaker. It is a collection of menus and is the element you will load or associate with layouts. The FileMaker Standard menu set includes all the menus we've worked with for years: File, Edit, View, Insert, and so on.

Note that any specific menu or menu item is present or not, grayed out or not, depending on certain conditions. For example, in Browse mode, the standard FileMaker Pro Advanced menu set includes the Records menu; but in Layout mode, the Records menu is not available and instead the Layouts

menu becomes available. These conditional states can be tied to FileMaker modes, layouts, or user platforms.

Specifying Custom Menu Elements

One important concept you will need to grasp is that FileMaker controls all custom menu elements at the menu set level. That means if you want your Records menu to show New Customer on a Customers layout and New Order on an Orders layout, you will need to create two additional custom menu sets—one to contain each new variant of the Records menu.

This does not mean, however, that you have to create duplicate menus or menu items. Menus and menu items can be used by multiple menu sets. You need only create menu items that are unique and require customization. In the preceding example, you would have to create two menu sets (a Customers set and an Orders set), two versions of the Records menu, and also two new menu items (New Customer and New Order).

When you change from one set to the other, all the user will see is that one menu item has changed; however, in the mechanics of working with custom menus, you will in fact have loaded a new menu set altogether.

Note that if you are starting from scratch, the menu set called My Menu Set will not yet exist: You create it in this section.

Using the Menu Sets Interface

In FileMaker Pro Advanced, choose Tools, Manage, Custom Menus, Manage Custom Menus to open the dialog shown in Figure 14.22.

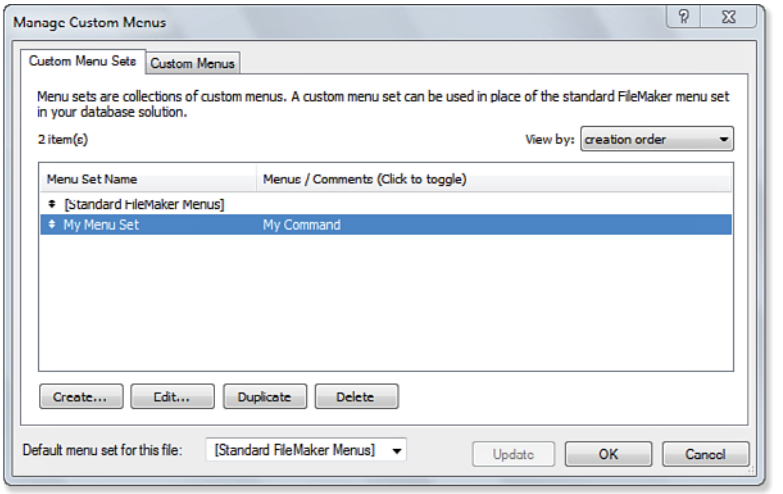
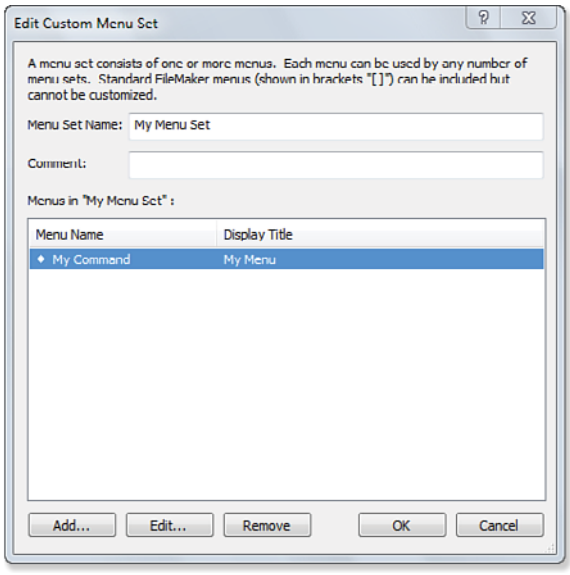


Figure 14.22
Manage your menu sets.

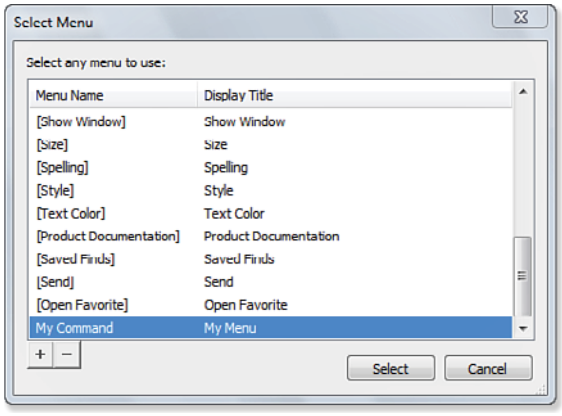
Click Create to create your menu set. The dialog shown in Figure 14.23 will open. Name your menu set. Notice that My Menu will not yet exist: That is what you will create in this step.

Figure 14.23
Name your menu set.



Click Add in the lower left to open the Select Menu dialog shown in Figure 14.24.

Figure 14.24
Select a menu to use.



Scroll down to the bottom. You will see the various FileMaker menus, but at the bottom you will be able to add your own. Figure 14.22 shows what it looks like when you have done your work.

Click the + button at the lower left to open the Create Custom Menu dialog shown in Figure 14.25.

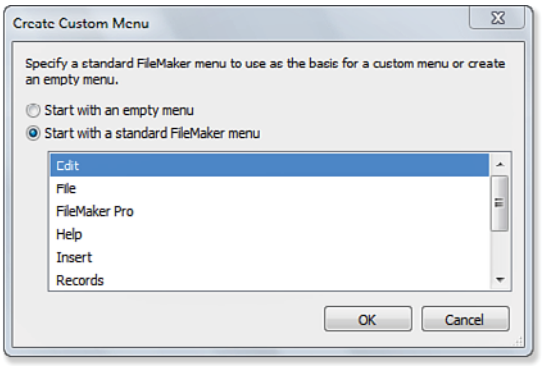


Figure 14.25
Create a custom menu.

Use the radio buttons at the top either to start with a standard FileMaker menu or to create a new empty menu. An empty menu is the choice to make for now.

The Edit Custom Menu dialog opens, as shown in Figure 14.26.

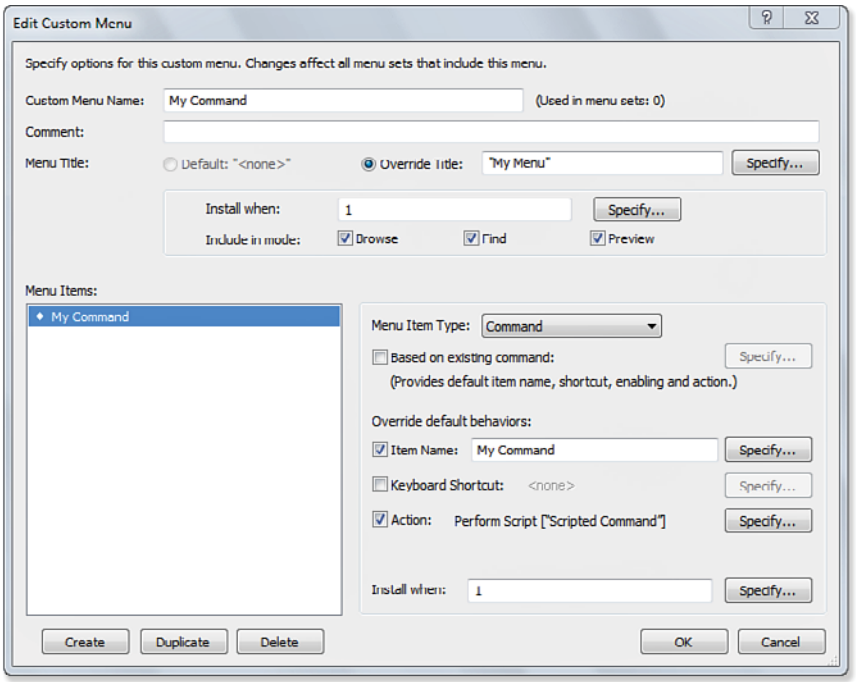


Figure 14.26
Edit your custom menu.

Provide a name for your new menu (this is the name you'll use in the Manage Custom Menus dialog. Don't confuse it with the menu title you specify on the third line down: That is what the user sees.

The Install When field is a calculation you set up to determine when this menu is to be installed. By default, it is set to 1 (true), but you can specify a calculation. Perhaps the most common calculation uses the `Get ( SystemPlatform )` function so that you install different menus on the different FileMaker platforms.

Next, check boxes let you determine which modes the menu should be installed in. When you click Create, a new menu item is created and selected, as you see at the lower left in the figure.

Your choices for Menu Item Type are

- Command
- Submenu
- Separator

In the basic case, you want a command. If you click Based On Existing Command, you can then select an existing FileMaker command and modify it by changing its name, its keyboard shortcut, or its action (or all three).

In this example, you build a totally new command, so you must specify its name and any keyboard shortcut you want to use. As you saw in Figure 14.26, in this example a script has been written. Click Specify for Action (or just click the check box) and you'll see a standard dialog for you to choose a script.

Finally, you have another Install When calculation: This controls the installation of the menu item.

The last part of installing the custom menu is attaching it to a layout or to a file. Use the Layouts, Layout Setup command in Layout mode to select the menu set for a layout, as shown in Figure 14.27.

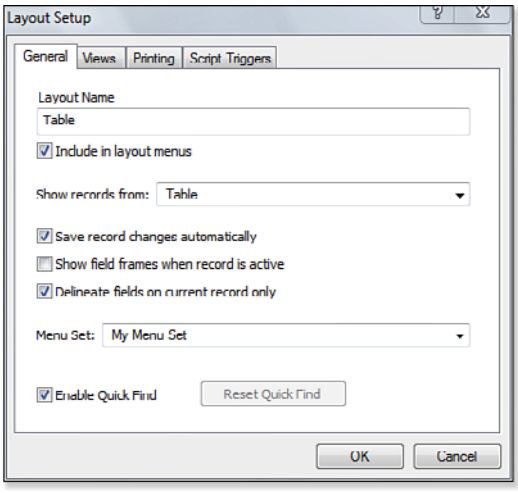


Figure 14.27
Use Layout Setup to attach a menu set to a layout.

To set up a custom menu set for a file, use the drop-down menu at the bottom of the Manage Custom Menus dialog, shown previously in Figure 14.22.

In addition, these techniques are also available to you:

- Load a menu set on demand by using a script.
- Load custom menu sets on demand, or tie menu sets to particular layouts, particular modes, or a particular operating system.
- Choose from among the available custom menus in the Tools, Custom Menus menu choices. This last option assumes that you're working with FileMaker Pro Advanced.
- Assign a custom menu set as the default menu set for an entire file.
- Assign a specific custom menu set to a layout.
- Load a menu set on demand by using a script.
- Load custom menu sets on demand, or tie menu sets to particular layouts, particular modes, or a particular operating system.
- Choose from among the available custom menus in the Tools, Custom Menus menu choices. This last option assumes that you're working with FileMaker Pro.

Providing Accessibility

FileMaker provides tools for the solution developer to improve access to solutions. This section covers the three aspects of providing improved accessibility for screen readers:

- Use the FileMaker Pro Accessibility Inspector to set up attributes for screen readers (developer)
- Turn on accessibility features in FileMaker and/or the operating system (end user)
- Use accessibility features (end user)

Set Up Accessibility Attributes in Layout Mode

In Layout mode, choose View, Accessibility to open the Accessibility Inspector as you see in Figure 14.28. As with all inspectors, the Accessibility Inspector reflects the values for the selected object. If no object in the layout is selected, the fields in the inspector are blank.

You can enter any of the three values in the Accessibility Inspector:

- **Name**—By default, this is the unqualified name of the field in the database that appears in the selected object. (Unqualified means that a field such as `Customers::Address` is shown as `Address`.)
- **Title**—This is the title that you want to be read by the screen reader. In this case, it might be *Customer Address*. This field can be entered as a calculation, which means you can incorporate data into it, such as a calculation like "Address for <*name of customer field*>". If <*name of customer field*> contains "Anatol," the spoken text would be "Address for Anatol."

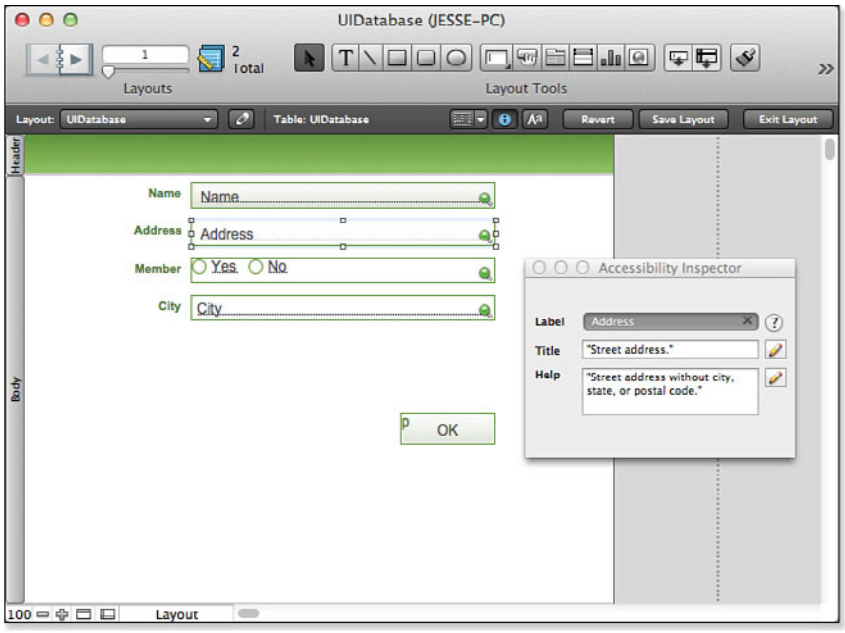


Figure 14.28
Use the Accessibility Inspector.

- **Help**—This is help text that will be spoken. It, too, can be a calculation field that incorporates data.

In the `title` and `help` fields, use punctuation in the text strings. The screen reader responds appropriately to commas and periods, so the resulting speech is easier to understand. Use Name, Title, and Help to provide different perspectives on the content: Do not simply repeat "Name" or "Address" in each label.

Turn On Accessibility Features

On OS X, use ⌘, System Preferences to turn on accessibility:

- In Universal Access, use the Seeing tab to turn on VoiceOver, as seen at the left in Figure 14.29.
- In Keyboard, use the Keyboard Shortcuts tab to select All Controls. This enables you to add accessibility to controls and fields. This is shown at the right of Figure 14.29.

Accessibility features can also be provided by third-party products, such as JAWS for Windows.

This step needs to be done only once for your computer unless you want to change the settings.

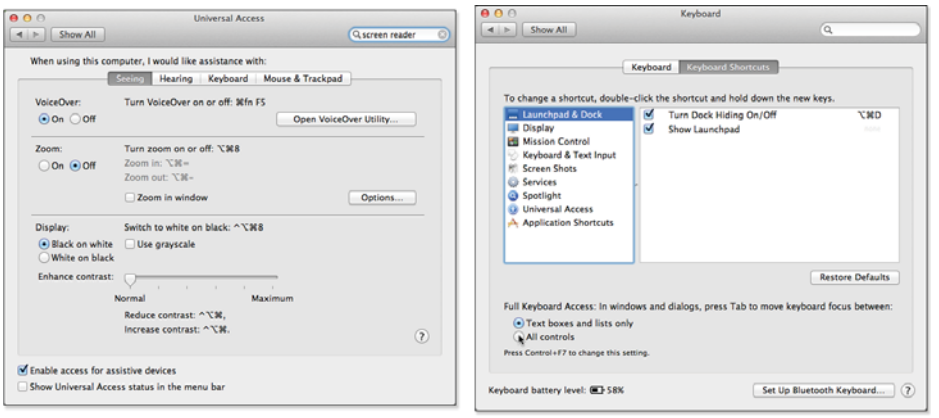


Figure 14.29
Turn on accessibility in OS X.

Use Accessibility Features

As people use the solution with the accessibility features turned on, the appropriate text will be displayed and read, as shown in Figure 14.30.

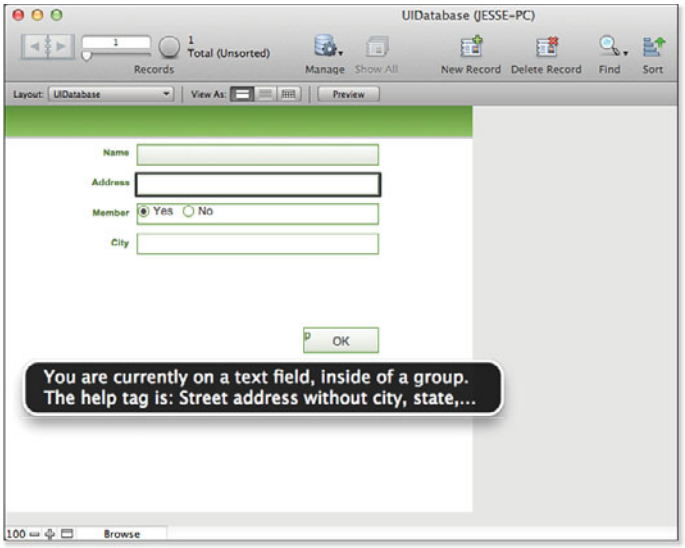


Figure 14.30
Use accessibility.

FileMaker Extra: User Interface Heuristics

We opened this chapter by saying we wouldn't preach to you about what makes a good interface and what doesn't. Well, we're breaking our word here. Although we won't argue about pop-up windows versus single-pane applications, or whether buttons should be 3D beveled or just text on the screen, here are a few guidelines we recommend to all our clients, students, and developers alike:

- **Use real-world terminology**—You should strive to speak your customers' language. Use terms they'll find familiar. In some cases, you might need to retrain them, but whenever possible, leverage the body of knowledge already in place in an organization to make your system more intuitive.
- **Impart meaning with more than just labels**—Text is only one of many things your users will see on a layout. They'll also see colors, shapes, headlines, subheads, footers, and so on. Use all the objects in your toolbox to impart meaning: Consider, for example, changing the background color of find layouts or perhaps making navigation buttons look different from functional buttons. Keep this in mind though: Don't rely too heavily on color. A great many people have varying degrees of colorblindness.
- **Give users the freedom to click around without fear**—Users should be able to cancel out of any destructive function (delete, for example) so that they can explore your application and learn by doing.
- **Be consistent**—We can't stress this point enough. Whatever the colors, shapes, sizes, styles, and so on that you prefer, make sure your layouts follow the set of rules you establish. Name fields and buttons consistently, place them in the same positions, and give your users a visual grammar for your system that they can learn.
- **Manage errors**—Errors happen. Handle them behind the scenes whenever possible, but when they're unavoidable, make sure you present the users with a graceful error routine that informs them, proffers a course of action, and then returns them to what they were doing.
- **Focus your screens**—Less is more. Whitespace is your friend. Leave the important bits on your layouts and dialogs and remove the objects that can be pushed elsewhere. If you offer focus to users, you will help them understand what to do on a given layout.
- **Remember your power users**—Contrary to all the earlier advice, don't forget your power users. On desktop and laptop layouts, offer keyboard shortcuts through "Are you sure?" dialogs, give them simple Table view access to your data, and don't bother them with assistants.

tip

With FileMaker Go, remember that touch-oriented interfaces, such as radio buttons, checkboxes, and pop-up menus, can be easier to use than the keyboard.

This page intentionally left blank

15

ADVANCED CALCULATION TECHNIQUES

Logical Functions

Chapter 8, "Getting Started with Calculations," presented an introduction to FileMaker Pro calculation formulas. This chapter deals not only with more specialized functions than the ones presented in Chapter 8, but also deals with the more programmatic functions—those that allow you to control operations of calculations. These are often called *logical functions*.

Chapter 8 discussed two of them: the `If` and `Case` conditional functions. This section continues with other logical functions.

The `Let` Function

The `Let` function enables you to simplify complex calculations by declaring variables to represent subexpressions. (In programming lingo, they are very much like subroutines.) These variables exist only within the scope of the formula, and you cannot reference them in other places. As an example, here is a formula presented in Chapter 8 for extracting the last line of a text field:

```
Right(myText; Length(myText) - Position(myText; "¶"; 1;
➥PatternCount(myText; "¶")))
```

With the Let function, you could rewrite this formula this way:

```
Let ([fieldLength = Length(myText) ;
      returnCount = PatternCount(myText; "¶") ;
      positionOfLastReturn = Position (myText; "¶"; 1; returnCount) ;
      charactersToGrab = fieldLength - positionOfLastReturn];

      Right (myText, charactersToGrab)
)
```

The Let function takes two parameters. The first is a list of variable declarations. If you want to declare multiple variables, you have to enclose the list within square brackets and separate the individual declarations within the list with semicolons. There are four variable declarations in the first parameter of the Let function shown here. The second parameter is some formula you want evaluated. That formula can reference any of the variables declared in the first parameter, just as it would reference any field value.

If you experience unexpected behavior of a Let *function, the trouble might be your variable names. For more information,* ***see*** *"Naming Variables in* Let *Functions" in the "Troubleshooting" section at the end of this chapter.*

Notice in this example that the third variable declared, positionOfLastReturn, references the returnCount variable, which was the second variable declared. This capability to have subsequent variables reference previously defined ones is one of the powerful aspects of the Let function because it enables you to build up a complex formula via a series of simpler ones.

It is fair to observe that the Let function is never *necessary*; you could rewrite any formula that uses the Let function, without using Let, either as a complex nested formula or by explicitly defining or setting fields or variables to contain subexpressions. The main benefits of using the Let function are simplicity, clarity, and ease of maintenance. For instance, you could write a formula that returns a person's age expressed as a number of years, months, and days, as shown here:

```
Year (Get (CurrentDate)) - Year(birthDate) - (DayOfYear(Get(CurrentDate))
➥ < DayOfYear(birthDate)) & " years, " & Mod ( Month(Get(CurrentDate))
➥- Month (birthDate) - (Day (Get(CurrentDate)) < Day(birthDate)); 12) &
➥" months, and " & (Get(CurrentDate) - Date (Month(Get(CurrentDate))
➥- (Day (Get(CurrentDate)) < Day(birthDate)); Day (birthDate);
➥Year (Get(CurrentDate)))) & " days"
```

This is a complex nested formula, and many subexpressions appear multiple times. The steps in writing and debugging this formula are difficult, even when you understand the logic on which it's based. With the Let function, you could rewrite the formula this way:

```
Let ( [   C = Get(CurrentDate);
          yC = Year (C) ;
          mC = Month (C) ;
          dC = Day (C) ;
          doyC = DayOfYear (C) ;
```

```
B = birthDate;
yB = Year (B) ;
mB = Month (B) ;
dB= Day (B) ;
doyB = DayOfYear (b) ;

num_years = ( yC - yB - (doyC < doyB)) ;
num_months = Mod (mC - mB - (dC <dB) ; 12) ;
num_days = C - Date (mC - (dC < dB) ; dB ; yC) ] ;

num_years & " years, " & num_months & " months, and " & num_days
➥& " days" )
```

Because of the extra space in the formula, this version is a bit longer than the original, but it's vastly easier to comprehend. If you were a developer needing to review and understand a formula written by someone else, we're sure you would agree that you'd prefer seeing this `Let` version rather than the first version.

The `Let` function's simplicity extends to fields that are similar to one another. For example, if you want to reformat a telephone number to insert standard symbols (parentheses and hyphens, for example), you can write a `Let` function to do so. You can then create calculation fields for a variety of phone numbers and simply paste the `Let` function into each one; all you have to do is change the first variable assignment statement to reference the particular phone number field you want to format.

In addition to simplicity and clarity, there are also performance benefits to using the `Let` function. If you have a complex subexpression that you refer to multiple times during the course of a calculation, FileMaker evaluates it anew each time it's referenced. If you create the subexpression as a variable within a `Let` statement, the subexpression is evaluated only once, no matter how many times it is subsequently referenced. In the example just shown, for instance, FileMaker would evaluate `Get(CurrentDate)` eight times in the first version. In the version that uses `Let`, it's evaluated only once. In many cases, the performance difference might be trivial or imperceptible. Other times, optimizing the evaluation of calculation formulas might be just the answer for increasing your solution's performance.

The more you use the `Let` function, the more likely it is to become one of the core functions you use. To help you become more familiar with it, we use it frequently throughout the examples in the rest of this chapter.

Quick Calculation Testing Using `Let`

The `Let` function makes it much easier to debug calculation formulas. It used to be that if you wanted to make sure that a subexpression was evaluating correctly, you had to create a separate field to investigate it. Using `Let`, you can just comment out the second parameter of the `Let` function and have the function return one or more of the subexpressions directly. When you've got each subexpression working as intended, just comment out the test code and uncomment the original code.

The Choose Function

The If and Case functions are sufficiently robust and elegant for most conditional tests that you'll write. For several types of conditional tests, however, the Choose function is a more appropriate option. As with If and Case, the value returned by the Choose function depends on the result of some test. What makes the Choose function different is that the test should return an integer rather than a true/false result. A number of possible results follow the test; the one chosen depends on the numeric result of the test. If the test result is 0, the first result is used. If the test result is 1, the second result is used, and so on. The syntax for Choose is as follows:

```
Choose (test ; result if test=0 ; result if test=1 ; result if test=2 ....)
```

A classic example of when the Choose function comes in handy is when you have categorical data stored as a number and you need to represent it as text. For instance, you might import demographic data in which an integer from 1 to 5 represents the ethnicity of an individual. You might use the following formula to represent it to users:

```
Choose (EthnicityCode; ""; "African American"; "Asian"; "Caucasian"; "Hispanic";
➥" Native American")
```

Of course, you could achieve the same result with the following formula:

```
Case (EthnicityCode = 1; "African American"; EthnicityCode = 2; "Asian",
➥EthnicityCode = 3; "Caucasian"; EthnicityCode = 4; "Hispanic";
➥EthnicityCode= 5; "Native American")
```

You should consider the Choose function in several other situations. The first is for generating random categorical data. Say your third-grade class is doing research on famous presidents, and you want to randomly assign each student one of the six presidents you have chosen. By first generating a random number from 0 to 5, you can use the Choose function to select a president. Don't worry that r isn't an integer; the Choose function ignores everything but the integer portion of a number. The formula would be this:

```
Let ( r = Random * 6;     // Generates a random number from 0 to 5
      Choose (r, "Washington", "Jefferson", "Lincoln", "Wilson", "Truman", "Kennedy"))
```

Several FileMaker functions return integer numbers from 1 to n, so these naturally work well as the test for a Choose function. Most notable are the DayofWeek function, which returns an integer from 1 to 7, and the Month function, which returns an integer from 1 to 12. As an example, you could use the Month function within a Choose to figure out within which quarter of the year a given date fell:

```
Choose (Month(myDate)-1; "Q1"; "Q1"; "Q1"; "Q2"; "Q2"; "Q2"; "Q3"; "Q3"; "Q3";
➥"Q4"; "Q4"; "Q4")
```

The -1 shifts the range of the output from 1–12 to 0–11, which is more desirable because the Choose function is *zero based*, meaning that the first result corresponds to a test value of 0. There

are more compact ways of determining the calendar quarter of a date, but this version is easy to understand and offers much flexibility.

Another example illustrating when Choose works well is when you need to combine the results of some number of Boolean tests to produce a distinct result. As an example, imagine that you have a table that contains results on Myers-Briggs personality tests. For each test given, you have scores for four pairs of personality traits (E/I, S/N, T/F, J/P). Based on which score in each pair is higher, you want to classify each participant as one of 16 personality types. Using If or Case statements, you would need a long, complex formula to do this. With Choose, you can treat the four tests as a binary number and then simply do a conversion back to base-10 to decode the results. The formula might look something like this:

```
Choose( (8 * (E>I)) + (4 * (S>N)) + (2 * (T>F)) + (J>P);
   "Type 1 - INFP" ; "Type 2 - INFJ" ; "Type 3 - INTP" ; "Type 4 - INTJ" ;
   "Type 5 - ISFP" ; "Type 6 - ISFJ" ; "Type 7 - ISTP" ; "Type 8 - ISTJ" ;
   "Type 9 - ENFP" ; "Type 10 - ENFJ" ; "Type 11 - ENTP" ; "Type 12 - ENTJ" ;
   "Type 13 - ESFP" ; "Type 14 - ESFJ" ; "Type 15 - ESTP" ; "Type 16 - ESTJ")
```

Each greater-than comparison is evaluated as a 1 or 0, depending on whether it represents a true or false statement for the given record. By multiplying each result by successive powers of 2, you end up with an integer from 0 to 15 that represents each of the possible outcomes. (This is similar to how flipping a coin four times generates 16 possible outcomes.)

As a final example, you also can use the Choose function any time you need to "decode" a set of abbreviations into their expanded versions. Consider, for example, a situation in which survey respondents have entered SA, A, N, D, or SD as a response to indicate Strongly Agree, Agree, Neutral, Disagree, or Strongly Disagree. You could map from the abbreviation to the expanded text by using a Case function like this:

```
Case (ResponseAbbreviation = "SA"; "Strongly Agree" ;
      ResponseAbbreviation = "A"; "Agree" ;
      ResponseAbbreviation = "N"; "Neutral" ;
      ResponseAbbreviation = "D"; "Disagree" ;
      ResponseAbbreviation = "SD"; "Strongly Disagree" )
```

You can accomplish the same mapping by using a Choose function if you treat the two sets of choices as ordered lists. You simply find the position of an item in the abbreviation list and then find the corresponding item from the expanded text list. The resulting formula would look like this:

```
Let ( [a = "|SA||A||N||D||SD|" ;
       r = "|" & ResponseAbbreviation & "|" ;
       pos = Position (a; r ; 1 ; 1) ;
       itemNumber = PatternCount (Left (a; pos-1); "|") / 2];

      Choose (itemNumber, "Strongly Agree"; "Agree"; "Neutral"; "Disagree";
              "Strongly Disagree")
)
```

In most cases, you'll probably opt for using the `Case` function for simple decoding of abbreviations. Sometimes, however, the list of choices isn't something you can explicitly test against (such as with the contents of a value list), and finding one choice's position within the list might suffice to identify a parallel position in some other list. Having the `Choose` function in your toolbox might offer an elegant solution to such challenges.

The `GetField` Function

When writing calculation formulas, you use field names to refer abstractly to the contents of particular fields in the current record. That is, the formula for a `FullName` calculation might be `FirstName & " " & LastName`. `FirstName` and `LastName` are abstractions; they represent data contained in particular fields.

Imagine, however, that instead of knowing in advance what fields to refer to in the `FullName` calculation, you wanted to let users pick any fields they wanted to. So, you set up two fields, which we'll call UserChoice1 and UserChoice2. How can you rewrite the `FullName` calculation so that it's not hard-coded to use `FirstName` and `LastName`, but rather uses the fields that users type in the two UserChoice fields?

The answer is the `GetField` function. `GetField` enables you to add another layer of abstraction to your calculation formulas. Instead of hard-coding field names in a formula, `GetField` allows you to place into a field the name of the field you're interested in accessing. That sounds much more complicated than it actually is. Using `GetField`, we might rewrite our `FullName` formula as shown here:

```
GetField (UserChoice1) & " " & GetField (UserChoice2)
```

The `GetField` function takes just one parameter. That parameter can be either a literal text string or a field name. Having it be a literal text string, although possible, is not particularly useful. The function `GetField("FirstName")` would certainly return the contents of the FirstName field, but you can achieve the same thing simply by using FirstName by itself. It's only when the parameter of the `GetField` function is a field or formula that it becomes interesting. In that case, the function returns the contents of the field referred to by the parameter.

There are many potential uses of `GetField` in a solution. Imagine, for instance, that you have a Contact table with the fields First Name, Nickname, and Last Name (among others). Sometimes contacts prefer to have their nickname appear on badges and in correspondence, and sometimes the first name is desired.

To deal with this, you could create a new text field called Preferred Name and format that field as a radio button containing First Name and Nickname as the choices. When doing data entry, a user could simply check off the name to use for correspondence. When it comes time to make a Full Name calculation field, one of your options would be the following:

```
Case ( Preferred Name = "First Name"; First Name;
       Preferred Name = "Nickname"; Nickname) &
       " " & Last Name
```

Another option, far more elegant and extensible, would be the following:

```
GetField (PreferredName) & " " & Last Name
```

> **note**
> When there are only two choices, the `Case` function certainly isn't cumbersome. But if there are dozens or hundreds of fields to choose from, `GetField` clearly has an advantage.

The `Evaluate` Function

The `Evaluate` function is one of the most intriguing functions in FileMaker. In a nutshell, it enables you to evaluate a dynamically generated or user-generated calculation formula. With a few examples, you'll easily understand what this function does. It might, however, take a bit more time and thought to understand why you would want to use it in a solution. We start by explaining the what and then suggest a few potential whys. The syntax for the `Evaluate` function is as follows:

```
Evaluate ( expression { ; [field1 ; field2 ;...]}  )
```

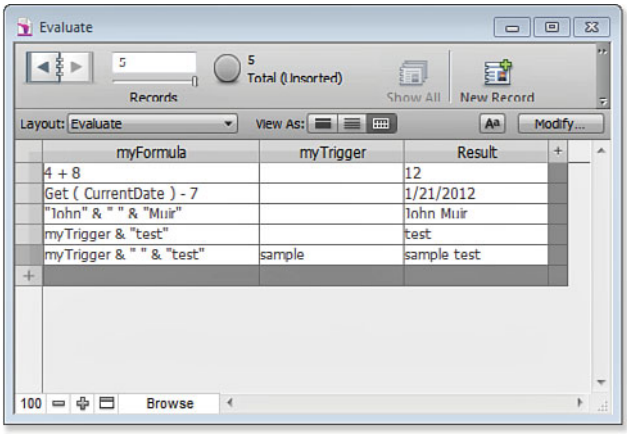
The `expression` parameter is a text string representing some calculation formula that you want to evaluate. The optional additional parameter is a list of fields whose modification triggers the reevaluation of the expression. Often, the expression itself uses these fields; if one of them changes, you want to reevaluate the expression.

For example, imagine that you have a text field named `myFormula` and another named `myTrigger`. You then define a new calculation field called Result, using the following formula:

```
Evaluate (myFormula; myTrigger)
```

Figure 15.1 shows some examples of what Result will contain for various entries in myFormula.

Figure 15.1
Using the `Evaluate` function, you can have a calculation field evaluate a formula contained in a field.



There's something quite profound going on here. Instead of having to hard-code calculation formulas, you can evaluate a formula that has been entered as field data. In this way, `Evaluate` provides an additional level of logic abstraction similar to the `GetField` function. In fact, if `myFormula` contained the name of a field, `Evaluate(myFormula)` and `GetField(myFormula)` would return exactly the same result. It might help to think of `Evaluate` as the big brother of `GetField`. Whereas `GetField` can return the value of a dynamically specified *field*, `Evaluate` can return the value of a dynamically specified *formula*.

Uses for the `Evaluate` Function

A typical use for the `Evaluate` function is to track modification information about a particular field or fields. A timestamp field defined to auto-enter the modification time triggers any time any field in the record is modified. Sometimes, however, you might want to know the last time that anyone modified the Comments field, without respect to other changes to the record. To do this, you would define a new calculation field, CommentsModificationTime, with the following formula:

```
Evaluate ("Get(CurrentTimestamp)" ; Comments)
```

The quotation marks around `Get(CurrentTimestamp)` are important, and are apt to be a source of confusion. The `Evaluate` function expects to be fed either a quote-enclosed text string (as shown here) or a formula that yields a text string (as in the Result field earlier). For instance, if you want to modify the CommentsModificationTime field so that rather than just returning a timestamp, it returns something like "Record last modified at: 11/28/2012 12:23:58 PM by Lily Bart," you would need to modify the formula to the following:

```
Evaluate ("\"Record modified at: \" & Get (CurrentTimeStamp) & \" by \" &
➥Get (AccountName)" ; Comments)
```

Because the formula you want to evaluate contains quotation marks, you must *escape* them by preceding them with a slash. For a formula of any complexity, this becomes difficult both to write and to read. Fortunately, a function named `Quote` eliminates all this complexity. The `Quote` function returns the parameter it is passed as a quote-wrapped text string, with all internal quotation marks properly escaped. Therefore, you could rewrite the preceding function more simply as this:

```
Evaluate (Quote ("Record modified at: " & Get (CurrentTimeStamp) & " by " &
➥Get (AccountName)) ; Comments)
```

In this particular case, using the `Let` function further clarifies the syntax:

```
Let ( [
    time = Get ( CurrentTimeStamp ) ;
    account = Get ( AccountName );
    myExpression = Quote ( "Record modified at: " & time & " by " & account ) ] ;

  Evaluate ( myExpression ; Comments )
)
```

Evaluation Errors

You typically find two other functions used in conjunction with the `Evaluate` function: `IsValidExpression` and `EvaluationError`. `IsValidExpression` takes as its parameter an expression; it returns a `1` if the expression is valid, a `0` if it isn't. An invalid expression is any expression that FileMaker Pro can't evaluate due to syntax errors or other runtime errors. If you plan to allow users to type calculation expressions into fields, be sure to use `IsValidExpression` to test their input to be sure that it's well formed. In fact, you probably want to include a check of some kind within your `Evaluate` formula itself:

```
Let ( valid = IsValidExpression (myFormula) ;
    If (not valid; "Your expression was invalid" ; Evaluate (myFormula) )
```

The `EvaluationError` function is likewise used to determine whether there's some problem with evaluating an expression. However, it returns the actual error code corresponding to the problem. One point to keep in mind, however, is that rather than testing the expression, you want to test the evaluation of the expression. So, as an error trap used in conjunction with an `Evaluate` function, you might have the following:

```
Let ( [result = Evaluate (myFormula) ;
     error = EvaluationError (result) ] ;
   If (error ; "Error: " & error ; result)
)
```

Lookup Functions

In versions of FileMaker before version 7, lookups were exclusively an auto-entry option. FileMaker 7 added two lookup functions, `Lookup` and `LookupNext`, and both are useful additions to any developer's toolkit.

The two lookup functions operate quite similarly to their cousin, the auto-entry lookup option. In essence, a lookup copies a related value into the current table. Lookups (all kinds) have three necessary components: a relationship, a trigger field, and a target field. When the trigger field is modified, the target field is set to some related field value.

It's important to understand the functional differences between the lookup functions and the auto-entry option. Although they behave similarly, they're not quite equivalent. Some of the key differences include the following:

- Auto-entry of a looked-up value is an option for regular text, number, date, time, or timestamp fields, which are subsequently modifiable by the user. A calculation field that includes a lookup function is not user modifiable.
- The lookup functions can be used anywhere—not just in field definitions. For instance, they can be used in formulas in scripts, record-level security settings, and calculated field validation. Auto-entering a looked-up value is limited to field definition.
- The lookup functions can be used in conjunction with other functions to create more complex logic rules. The auto-entry options are comparatively limited.

Lookup

The syntax of the Lookup function is as follows:

```
Lookup ( sourceField { ; failExpression}  )
```

sourceField is the related field whose value you want to retrieve. The optional failExpression parameter is returned if there is no related record or if sourceField is blank for the related record. If the specified relationship matches multiple related records, the value from the first related record is returned.

There are two main differences between using the Lookup function and simply referencing a related field in a formula. The first is that calculations that simply reference related fields must be unstored, but calculations that use the Lookup function to access related fields can be stored and indexed. The other difference is that changing sourceField in the related table does not cause the Lookup function to retrigger. Just as with auto-entry of a looked-up value, the Lookup function captures sourceField as it existed at a moment in time. The alternative, simply referencing the related field, causes all the values to remain perfectly in sync: When the related value is updated, any calculations that reference it are updated as well. The downside is that, as with all calculations that directly reference related data, such a calculation cannot be stored.

LookupNext

The LookupNext function is designed to allow you to map continuous data elements to categorical results. It has the same effect as checking the Copy Next Lower Value or Copy Next Higher Value option when specifying an auto-entry lookup field option. Here is its syntax:

```
LookupNext ( sourceField ; lower/higherFlag )
```

The acceptable values for the second parameter are Lower and Higher. These are keywords and shouldn't be placed in quotation marks.

An example should help clarify what we mean about mapping continuous data to categorical results. Imagine that you have a table containing information about people, and that one of the fields is the person's birth date. You want to have some calculation fields that display the person's astrological information, such as a zodiac sign and ruling planet. Having birth dates mapping to zodiac signs is a good example of continuous data mapping to categorical results: A range of birth dates corresponds to each zodiac sign.

In practice, two small but instructive complications arise when you try to look up zodiac signs. The first complication is that the zodiac date ranges are expressed not as full dates, but merely as months and days (for example, Cancer starts on June 22 regardless of what year it is). This means that when you set up your zodiac table, you'll use text fields rather than date fields for the start and end dates. The second complication is that Capricorn wraps around the end of the year. The easiest way to deal with this is to have two records in the Zodiac table for Capricorn—one that spans December 22–December 31, and the other that spans January 1–January 20.

Figure 15.2 shows the full data of the Zodiac table. The StartDate and EndDate fields, remember, are actually text fields.

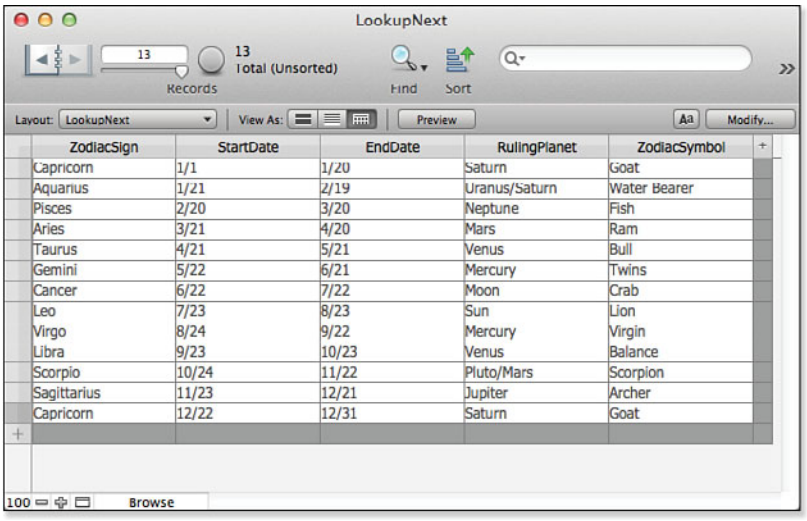


Figure 15.2
The data from the Zodiac table is looked up and is transferred to a person record based on the person's birth date.

In the Person table, you need information such as the name of the person as well as the birth date—a true date field (you can call it BirthDateAsDate). Create a calculation formula for a separate BirthDate field that generates a text string containing the month and date of the person's birth date from the BirthDateAsDate field. The BirthDateAsDate field is defined this way:

```
Month (BirthdateAsDate) & "/" & Day (BirthdateAsDate)
```

Next, create a relationship between the Person and Zodiac tables, matching the BirthDate field (the text field) in Person to the StartDate field in Zodiac. Figure 15.3 shows this relationship.

Obviously, many birth dates aren't start dates for one of the zodiac signs. To match to the correct zodiac record, you want to find the next lower match when no exact match is found. For instance, with a birth date of February 13 (2/13), there is no matching record where the StartDate is 2/13, so the next lowest StartDate, which is 1/21 (Aquarius), should be used.

In the Person table, therefore, you can grab any desired zodiac information by using the `LookupNext` function. Figure 15.4 shows an example of how this date might be displayed on a person record. The formula for ZodiacInfo is as follows:

```
"Sign: " & LookupNext (Zodiac::ZodiacSign; Lower) & "¶" &
"Symbol: " & LookupNext (Zodiac::ZodiacSymbol; Lower)
➥& "¶" &
"Ruling Planet: " & LookupNext (Zodiac::RulingPlanet;
➥Lower)
```

It would have been possible in the previous examples to match to EndDate instead of StartDate. In that case, you would simply need to match to the next higher instead of the next lower matching record.

note

An entirely different but perfectly valid way of approaching the problem would have been to define a more complex relationship between Person and Zodiac, in which `DateMatch` is greater than or equal to StartDate and less than or equal to EndDate. Doing this would allow you to use the fields from the Zodiac table as plain related fields; no lookup would have been required. There are no clear advantages or disadvantages of this method over the one discussed previously.

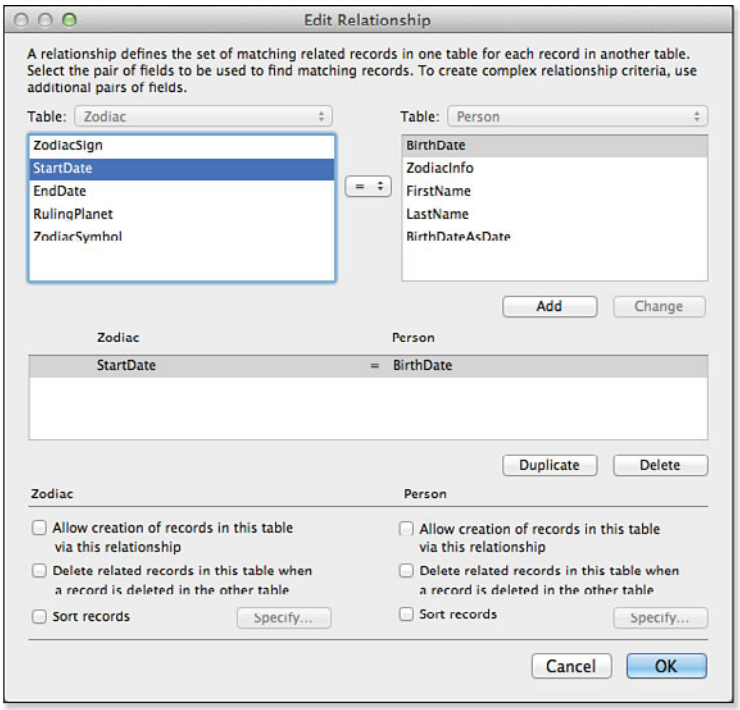


Figure 15.3
By relating the Person table to Zodiac, you can look up any information you want based on the person's birth date.

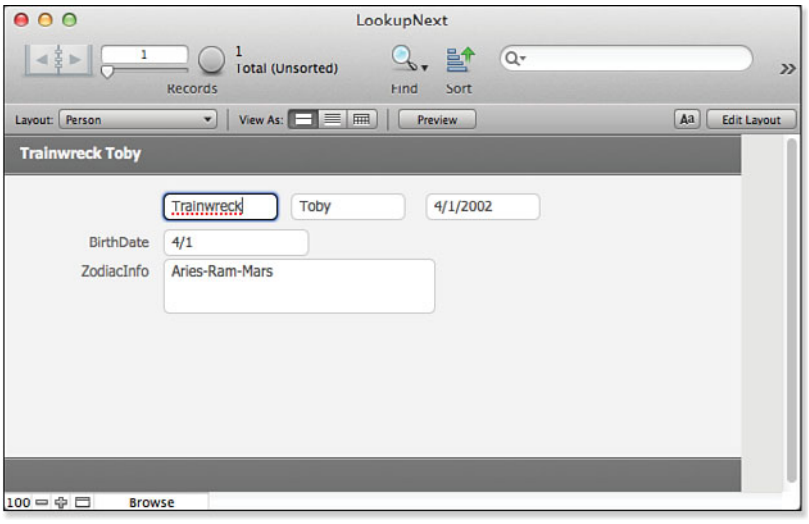


Figure 15.4
Using the `LookupNext` function, you can create a calculation field in the Person table that contains information from the next lower matching record.

note

Other typical scenarios for using `LookupNext` are for finding shipping rates based on weight ranges, determining price discounts based on quantity ranges, and defining cut scores based on continuous test score ranges, for example.

Text Formatting Functions

In versions of FileMaker Pro before version 7, there was no way to affect the display of a field (that is, color, size, font, style) via calculation formulas. Developers had to come up with workarounds for seemingly simple tasks, such as having the contents of a field change color based on some conditional test. For example, a typical workaround was stacking two calculation fields on top of one another, each formatted with a different text color on the layout, and then having a conditional test in each to turn it "on" or "off" to simulate the effect of the text changing color. Beginning with FileMaker Pro 8, nine text formatting functions obviate the need for many of these old workaround options.

Conditional Formatting Versus Multiple Layouts

Beginning with FileMaker Pro 9, conditional formatting in layouts (as discussed in Chapter 14, "Advanced Interface Techniques") can be used to dynamically change the appearance of fields. This gives you three ways to format fields: by formatting the field in a layout, by formatting the contents with a function as described here, and by using conditional formatting on the layout. The user, of course, can use FileMaker's text formatting commands when editing data; that is a separate matter from the developer's formatting of text.

In addition, you can use multiple versions of a layout to format the same field in different ways. This is most frequently used to provide alternate layouts for desktop and mobile devices. You use a startup trigger script to switch to the appropriate layout, and thereafter the navigation from one layout to the next is handled on generally parallel tracks for each device.

In general, conditional formatting is best used for formatting that is context sensitive (an error, for example). Formatting with a function is best used when the formatting is always to be used in displaying the text.

Text Color, Font, and Size

The `TextColor`, `TextFont`, and `TextSize` functions are quite similar. The first parameter of each function is the text string you want to act on; the second parameter contains the formatting instructions you want to apply.

For example, perhaps you have a Tasks table, and you want any tasks due within the next week to display in red. To accomplish this task, you would define a calculation field named TaskDisplay with the following formula:

```
Case (DueDate <= Get (CurrentDate) + 7;
      TextColor (TaskName; RGB (255; 0; 0));    // Red
      TextColor (TaskName; 0))                  // Black
```

The TaskDisplay field displays the task name in either red or black, depending on the due date.

The second parameter of the `TextColor` function must be an integer from 0 to 16777215 (which is $256^3 - 1$), which represents a unique RGB color. If you know the integer value of the color you want (for example, black is 0), you can simply use that integer. More typically, you'll use the `RGB` function, which returns the integer representation of the color specified. Each of the three parameters in the `RGB` function must be an integer between 0 and 255. The first parameter represents the red component of the color; the second, the green component; the third parameter represents the blue component. The `RGB` function determines the integer representation by the following formula:

```
((255^2) * Red) + (255 * Green) + Blue
```

Text Style

The next two text formatting functions are `TextStyleAdd` and `TextStyleRemove`. Each of these functions takes two parameters. The first is a text string to act on; the second is a style or styles to apply to the text string. If listing multiple styles, you have to separate them with a plus sign (+). The style names are keywords and should not appear in quotation marks. They also must be hard-coded in the formula; you can't substitute a field that contains style instructions. Here is a list of the valid styles for both `TextStyleAdd` and `TextStyleRemove`:

```
Plain
Bold
Italic
Underline
Condense
Extend
Strikethrough
SmallCaps
Superscript
Subscript
Uppercase
Lowercase
Titlecase
WordUnderline
DoubleUnderline
AllStyles
```

To remove all styles from a chunk of text, you can either *add* `Plain` as a style or *remove* `AllStyles`. Additionally, there are numeric equivalents for each of the text style keywords. Unlike the keywords themselves, the numeric equivalents can be abstracted as field values.

Removing Text Formatting

In addition to functions for selectively adding formatting to text strings, FileMaker has functions for removing formatting from text. In addition to `TextStyleRemove`, mentioned previously, there are functions named `TextFontRemove`, `TextColorRemove`, `TextSizeRemove`, and `TextFormatRemove`.

The first three of these remove some specific styling attribute from the designated text. `TextFormatRemove` removes all formatting from the selected text in one operation.

For most of these functions, you can specify an optional second parameter that specifies exactly what value you want to remove. For example,

```
TextSizeRemove( text )
```

removes all text sizing from `text`, causing all of `text` to return to whatever text size was specified for the field in Layout mode, whereas

```
TextSizeRemove( text; 14 )
```

removes only the 14-point size from `text`, causing any characters in a 14-point size to revert to the field default size.

`TextFormatRemove`, as mentioned, is the exception to this pattern. `TextFormatRemove` takes just one parameter, the text string to be reformatted, and strips all formatting from the field.

You might have difficulty applying text formatting functions within calculations that return something other than plain text. ***See*** *"Text Formatting in Nontext Calculations" in the "Troubleshooting" section at the end of this chapter.*

Array Functions

Arrays are a powerful and extremely useful programming concept. If you've done any programming in languages such as C++, Perl, PHP, or Visual Basic, you're probably familiar with both the concept of arrays and some uses for them. We think it likely, however, that most FileMaker Pro developers out there haven't had much experience with arrays and will benefit from both a formal and a practical discussion of them.

Abstractly, an *array* is essentially a structure that can hold multiple values. The values are ordered within the structure and can be referenced by their position or index number. Figure 15.5 shows a representation of a simple array. The array has been defined to hold up to seven values, but only four values are present. The first element in the array is the value `red`.

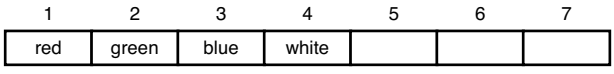


Figure 15.5
An array is a structure that can hold multiple values. Each value can be identified and referenced by an index number.

Arrays are useful for a wide variety of tasks, including storing lists of data, efficiently moving multiple values through a system, and dealing with variable-size data structures where it's impossible to define separate fields for each individual data element. FileMaker Pro doesn't have an explicit "array" data type, but fields defined to hold multiple repetitions can be regarded as arrays. More commonly, if you want to use arrays in FileMaker, you can create your own by placing into a text field multiple values separated by some delimiter.

Return-delimited lists pop up all over the place in FileMaker Pro. Many functions and operations in FileMaker generate return-delimited lists, including most of the `Design` functions and the `Get (ExtendedPrivileges)` function. When a user selects multiple values in a check box–formatted field, FileMaker stores that data as a return-delimited list of the selections. Additionally, the `Copy All Records` script step generates a return-delimited list of the data elements on the current layout for the current found set (the tab character separates elements within a record).

note

In FileMaker Pro, you can use "array notation" to refer to data in a repeating field. `myField[3]`, for instance, refers to the data in the third repetition of `myField`. It's really just a shorthand notation for `GetRepetition(myField, 3)`, but it makes formulas much easier to read.

Working with Return-Delimited Data Arrays

FileMaker Pro has five functions that greatly facilitate working with return-delimited data arrays such as the ones just described. They are `ValueCount`, `LeftValues`, `MiddleValues`, `RightValues`, and `GetValue`. Syntactically, they are similar to the four "word" functions (`WordCount`, `LeftWords`, `MiddleWords`, and `RightWords`) as well as to the four "character" functions (`Length`, `Left`, `Middle`, and `Right`).

Briefly, the syntax of these functions is as described here:

- `ValueCount (text)`—Returns the number of items in a return-delimited list. Unlike its cousin the `WordCount` function, which interprets sequential word delimiters as a single delimiter, if you have multiple carriage returns in a row, even at the beginning or end of a list, `ValueCount` treats each one as a delimiter. For example, `ValueCount ("++Red+Blue+Green++White+")` returns `7`. It's immaterial whether the list contains a single trailing return; the `ValueCount` is unaffected by this. Multiple trailing returns affect the `ValueCount`.
- `LeftValues (text; numberOfValues)`—Returns a list of the first *n* elements of a return-delimited text string. The list always has a trailing return, even if you are requesting only the first item of the array.
- `MiddleValues (text; startIndex; numberOfValues)`—Returns a list of *n* elements from the middle of a return-delimited array, starting from the position specified in the second parameter. As with `LeftValues`, the output of this function always contains a trailing return.
- `RightValues (text; numberOfValues)`—Returns a list of the last *n* elements from a return-delimited array. This function, too, always generates a trailing return at the end of its output.
- `GetValue (listOfValues; valueNumber)`—Returns a single value from a return-delimited list of values. This value will not contain a trailing carriage return. This function is useful when you want to loop through a set of values and perform some operation using each value in turn.

*If you ever use arrays that use delimiters other than return characters, **see** "Working with Arrays" in the "Troubleshooting" section at the end of this chapter.*

To demonstrate how you might use these functions in a solution, we present an example of iterating through a user's selections in a check box–formatted field and creating records for each selection

in another table. Imagine that you have a table containing information about kids coming to your summer camp, and that one of the pieces of information you are capturing is a list of sports in which the child wants to participate. When you originally set up the table, you simply created a check box–formatted field in the CamperInfo table for this information. You now realize that it's impossible to run certain reports (for example, a subsummary by sport) with the data structured this way and that you should have created a separate table for CamperSport data. You'd like not to have to reenter all the data, so you want to create a script that loops through all the CamperInfo records and creates a record in the CamperSport table for each sport checked for that camper.

We can approach a challenge such as this in many ways. We might, for instance, temporarily set data from CamperInfo into variables, navigate to a layout based on the CamperSport table, create records, and populate data from the variables. We've chosen instead to use a portal from the CamperInfo table to the CamperSport table that allows the creation of related records. This way, we avoid having to navigate between layouts for each camper, and the CamperID field is automatically set correctly in the CamperSport table.

Stepping Through an Array

A user's selections in a check box field are stored as a return-delimited array, in the order in which the user checked them. You can step from element to element in such an array in two ways. One method is to iteratively "lop off" the first element of the array until there's nothing left to process. This requires first moving the data to be processed into a temporary location where it can be cut apart without harming the original data.

The other method is to use a counter to keep track of what element is being processed. You continue processing, incrementing the counter as you go, until the counter exceeds the number of elements in the array. To some extent, which method you use is personal preference. Some developers preferred the first method in earlier versions of FileMaker Pro because it was simpler syntactically, but the newer "value" functions (introduced in FileMaker 7) make the second method appealing now. Listings 15.1 and 15.2 present both versions of the script so that you can decide for yourself which approach is preferable.

Listing 15.1 Method 1: "Lop off" the Top Element of the Array

```
Go to Layout ["CamperInfo" (CamperInfo)]
Go to Record/Request/Page [First]
Loop
    Set Variable [$sportArray; Value: CamperInfo::SportArray]
    Loop
        Exit Loop If [ValueCount ($sportArray) = 0]
        Go to Portal Row [Select; Last]
        Set Field [CamperSport::Sport; GetValue ($sportArray; 1)
        Set Variable [$sportArray; Value: Let (count =
        ➥ValueCount($sportArray);
        ➥RightValues ( $sportArray; count-1))
    End Loop
    Go to Record/Request/Page [Next; Exit after last]
End Loop
```

Notice that in line 8, the first element of the SportArray field is pushed through the portal, where it becomes a record in the CamperSport table. In the next line, the `$sportArray` variable is then reset to be everything *after* the first line. It gets shorter and shorter with each pass through the loop, until finally there aren't any more items to process, concluding the inner loop.

Listing 15.2 Method 2: Walk Through the Elements One by One

```
Go to Layout ["CamperInfo" (CamperInfo)]
Go to Record/Request/Page [First]
Loop
Set Variable [$counter; Value: 1]
    Loop
        Exit Loop If [$counter > ValueCount (CamperInfo::SportArray)]
        Go to Portal Row [Select; Last]
        Set Field [CamperSport::Sport;
                  ➥GetValue (CamperInfo::SportArray; $counter)
        Set Variable [$counter; Value: $counter  + 1]
    End Loop
    Go to Record/Request/Page [Next; Exit after last]
End Loop
```

Again, the main difference in this method is that the inner loop steps through the elements of the SportArray field based on a counter variable.

The "FILTER"-ing Functions

The `Filter` and `FilterValues` functions, introduced in FileMaker 7, are nifty tools for complex text comparison and manipulation. The following sections provide an example of each.

The `Filter` Function

The syntax for the `Filter` function is as follows:

```
Filter (textToFilter; filterText)
```

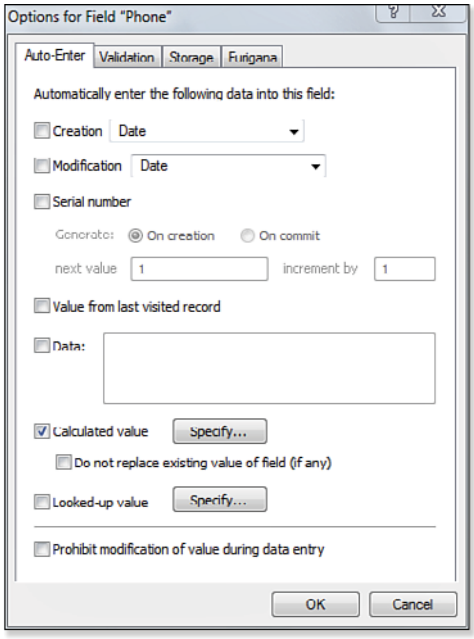
The `filterText` parameter consists of a set of characters that you want to "protect" in `textToFilter`. The output of the `Filter` function is the `textToFilter` string, minus any characters that don't appear in `filterText`. For example:

```
Filter ("This is a test" ; "aeiou") = "iiae"
```

Here, the filter is the set of five vowels. Therefore, the output from the function contains all the vowels from the string `"This is a test"`. The filter is case sensitive, so if you want to include both uppercase and lowercase vowels in your output, you have to make the `filterText` parameter `aeiouAEIOU`. The output is ordered according to the order in which characters in the filter are found in the first parameter. The order of the characters in the filter itself is irrelevant.

The Filter function is useful any time you want to constrain the domain of possible characters that a user can enter into a field. The most common use of Filter, therefore, is as part of an auto-entry calculation for text fields. Figure 15.6 shows the auto-entry options dialog for a field named Phone. Note that the option Do Not Replace Existing Value of Field (If Any) has been unchecked. That means the auto-entry calculation does not trigger only when the record is created, but also when the Phone field is modified. Essentially, this means that whenever a user modifies the Phone field, the result of the specified calculation formula immediately replaces his entry.

Figure 15.6
The Filter function is often used as part of the auto-entry of a calculated value.



You can use the Filter function as part of the auto-entry calculation for the Phone field to remove any non-numeric characters entered by the user. A nice thing about the Filter function is that you don't need to anticipate all the incorrect things a user can enter (text, punctuation, spaces), but rather you can specify what the acceptable characters are. The actual function you use to reformat the user's entry in the Phone field depends on your needs and preferences, but one option would be the following:

```
Let ( [
    ph = Filter (Phone; "0123456789");
    len = Length (ph) ;
    areaCode = Case ( len = 10; Left (ph; 3); "");
    exchange = Case ( len = 10; Middle (ph; 4; 3); Left (ph; 3)) ;
    end = Right (ph; 4) ];
```

```
    Case (
        len =10 ;  "(" & areaCode & ") " & exchange & "-" & end ;
        len =7 ;   exchange & "-" & end ;

       "Error: " & TextStyleAdd ( Phone ; Bold)
    )
)
```

The formula starts by stripping out any non-numeric characters from the user's entry. Then, if the length of the remaining string is either 7 or 10, the number is formatted with punctuation and returned to the user. If it's not, the function shows the user an error message, complete with the original entry presented in bold text.

The `FilterValues` Function

The `FilterValues` function is similar to the `Filter` function, except that it filters the elements in one return-delimited set by the elements in a second return-delimited set. When each set consists of unique elements, the `FilterValues` function essentially returns the intersection of the two sets. In Figure 15.7, you can see that `FilterValues` returns the items common to the two sets. Had the two parameters been reversed and the formula been written as `FilterValues (Set B; Set A)`, the only difference would have been the order of the elements in the resulting list.

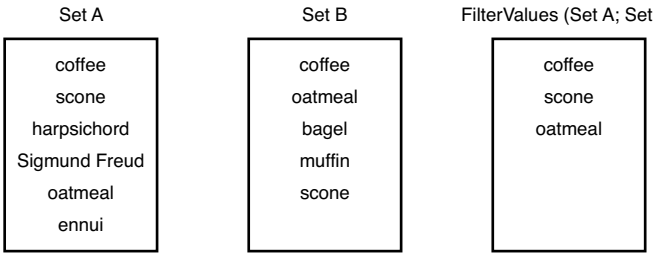


Figure 15.7
The `FilterValues` function returns a list of all the items of Set A that are also in Set B.

note

The result list always is ordered based on the first set. If an element appears multiple times in the first set, and it's included in the filter set, it appears multiple times in the result set.

`FilterValues` comes in handy any time you want to see whether two lists contain any of the same elements. For instance, if you've defined any extended privileges as part of your security settings, you can see a list of all the privileges granted to the current user with the `Get (ExtendedPrivileges)` function. If you have some routine that only users with `PrivSetA` or `PrivSetC` should have access to, you can use the formula `FilterValues("PrivSetA+PrivSetC"; Get (ExtendedPrivileges))`. If the result is not empty, the user has at least one of those two privilege sets.

As another example, imagine that you are a third-grade teacher and that you have given your students a ten-question true/false test. Rather than setting up a related table for their answers, you've

entered all their responses into a return-delimited text field. By also putting the answer key into a global text field, you can use the `FilterValues` function to determine the number of correct answers each student had. Figure 15.8 shows how this might look when you're finished. The formula for the NumberCorrect field is the following:

```
ValueCount (FilterValues (TestResults; AnswerKey) )
```

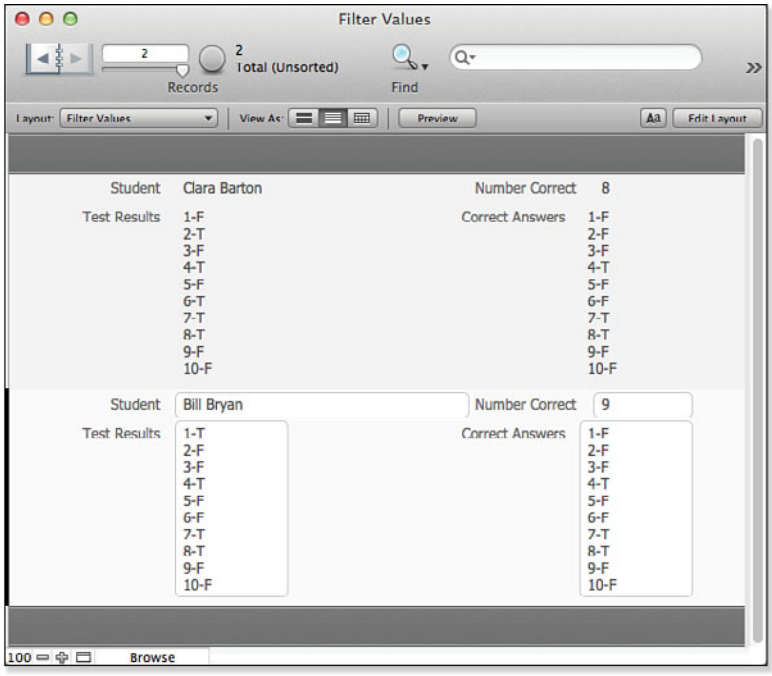


Figure 15.8
By using the `FilterValues` and `ValueCount` functions, you can count how many items in one array are contained within some other array.

Custom Functions

In addition to all the wonderful and powerful calculation functions built into FileMaker Pro, you can create your own custom functions. To create custom functions, you must have a copy of FileMaker Pro Advanced. Any custom functions you create using FileMaker Pro Advanced remain in the database file and are fully usable when the regular FileMaker Pro client or FileMaker Go application subsequently uses that file. You just can't edit the formula of a custom function unless you have FileMaker Pro Advanced.

As with other objects, such as scripts, tables, and user account information, custom functions live in a particular file. There is, unfortunately, no easy way to move or import custom functions defined in one file into another one. The implications of this are obvious: If you have a solution that consists of multiple files, you have to define custom functions redundantly in all the files that need to access them, thus complicating maintenance and troubleshooting. This fact shouldn't scare you off from using custom functions—they're really quite wonderful—but it's certainly a constraint you must be aware of.

Alternatively, if you use several custom functions, you can add a file containing those functions and relations to the relevant tables in other files. You then just have to reconnect those tables in copies if you want.

Custom functions created for a particular file show up with all the built-in functions in the list of functions within the calculation dialog. To see only the custom functions, you can choose Custom Functions from the filter above the function list. Custom functions are used in a formula just as any other function. The person who writes the custom function defines the function name and the names of its parameters.

Uses of Custom Functions

There are several reasons for using custom functions in a solution. Custom functions enable you to abstract snippets of calculation logic so that they become reusable. Abstracting out bits of logic also makes your code easier to read and eliminates redundancy.

Simplifying Complex Formulas

The best place to begin understanding the potential uses of custom functions is with a simple example. Imagine that you need to generate a random integer from 10 to 50. Knowing, as you do from reading Chapter 8, that the `Random` function returns a random number between 0 and 1, you eventually conclude that the following formula solves this particular problem:

```
Int(Random * 41) + 10
```

With the problem solved, you write your formula and go on your merry way. Now, imagine that the next day you come back and discover that you need to write another function that requires a random integer from 1 to 6. After a bit more thinking, you come up with the following:

```
Int(Random * 6) + 1
```

About this time, you'd be wishing that the engineers at FileMaker, Inc., had thought to create a function that would return a random integer from *x* to *y*. Using FileMaker Pro Advanced, you can write your own custom functions for situations such as this. Rather than continuing to solve particular problems, you can solve the general case and never again need to divert your attention to the particular.

So, what would a generalized solution to the random number problem look like? First, you need to have some way of abstractly representing the "from" and "to" numbers. Let's call these two numbers `lowNumber` and `highNumber`. Then the function that satisfies the general condition would be this:

```
Int (Random * (highNumber - lowNumber + 1)) + lowNumber
```

For any `lowNumber` and `highNumber` you feed this function, you get back an integer between the two. In a moment we'll look at how you would go about setting this up as a custom function, but for now the important point is the concept that custom functions, just like the built-in functions you use all the time, have inputs (called *parameters*) and an output. Let's say that you decide to call this

function `randomInRange`. Now, to solve the first problem we looked at, finding a random integer from 10 to 50, you could just use the following function:

```
randomInRange (10; 50)
```

And to find a number from 1 to 6, you could use this function:

```
randomInRange (1; 6)
```

You've simplified your code by replacing a complex expression with a single function, thereby making it easier to read and maintain. You've abstracted that bit of logic out of whatever larger formula you were working on, leaving you with one fewer thing to think about.

Custom Functions as System Constants

A few different schools of thought exist regarding when you should write a custom function to abstract your programming logic and when you should use existing tools to solve the problem. Some developers hold that you should always write custom functions. Even if you use a given custom function only a single time, you made your code more modular, thus making it easier to track down and troubleshoot problems. Plus, if you ever need that function again, it's there, ready and waiting.

Other developers find that they use custom functions more sparingly. Their attitude is this: If you find yourself solving a particular problem more than once, go ahead and write a custom function for it, and go back to change the original occurrence to reference the custom function instead. This process, often called *refactoring* as a general programming concept, has a certain pragmatism to it: Write a custom function as soon as it's more efficient to do so, but not sooner.

Whatever camp you find yourself falling into, you should be aware of two other common uses for custom functions. The first is for defining system constants. As an example, imagine that the commission rate is 15% in your sales organization. In calculations where you determine commission amounts, you might find yourself writing numerous formulas that multiply sales figures by 0.15. If, heaven forbid, you ever need to change that figure to, say, 0.18, you'd need to sift through all your code to find all the instances where you had hard-coded the commission figure.

As an alternative, you might consider defining custom functions to represent systemwide constants such as these. In this example, you would simply have a custom function called `CommissionRate` that had no parameters and returned a value of `.15`. By abstracting out the hard-coded value, you're able to quickly and easily make global changes by editing a single function. You should never refer directly to the magic number in a formula; use the custom function instead. Other examples of numbers and strings that should be abstracted out of your formulas include IP addresses, URLs, and colors.

note

There's a subtle pitfall here. Note that stored values that reference custom functions do *not* automatically update when a custom function definition changes. For example, if you implement a system constant called `commissionRate` as a custom function and then go on to create one or more stored calculations that reference `commissionRate`, the values in those calculations do not update if you later redefine `commissionRate` to be 18%. The same would hold true of data that's auto-entered into a field. If you want these stored values to take account of the new commission rate, you have to force the fields to explicitly refresh their contents—perhaps by using a field for rate.

Creating Recursive Functions

The final common situation in which custom functions are used is for making recursive functions. One of the limitations often lamented by developers over the years has been the fact that you can't create looping constructs within calculation formulas. That is, you can't instruct a regular calculation formula to keep doing something until some condition holds.

Custom functions, on the other hand, can contain recursive logic, which mimics the effects of a looping control structure. This means that a class of problems can be solved only by the creation of custom functions. This stands in stark contrast to the "custom functions as vehicles for abstraction" idea discussed previously. As an abstraction tool, custom functions can always be replaced in a formula by the logic they abstract. No such substitution can be made when dealing with recursive functions. In those cases, using custom functions is not a convenience; it's a necessity. In the section that follows, we develop and discuss several recursive functions.

Creating Custom Functions

Now that you understand what custom functions are and why you might want to use them, it's time to turn to the mundane subject of how to actually create them. First, recall that custom functions can be created and edited only with FileMaker Pro Advanced, and that custom functions live in a specific file. To see a list of custom functions defined in a particular file, and to define new ones, choose File, Manage, Custom Functions. Figure 15.9 shows the resultant Manage Custom Functions dialog.

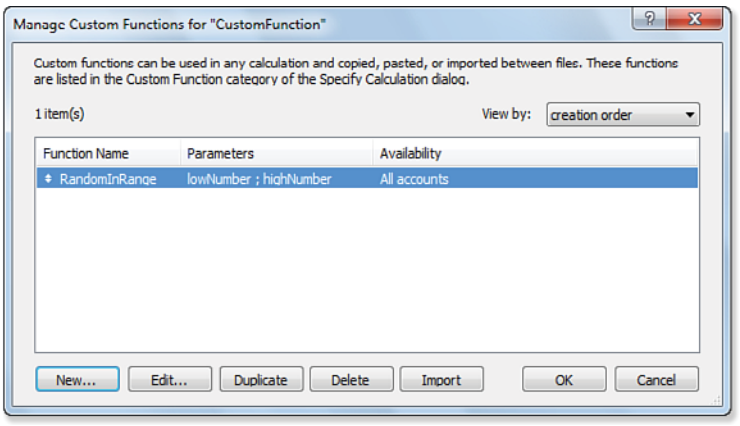
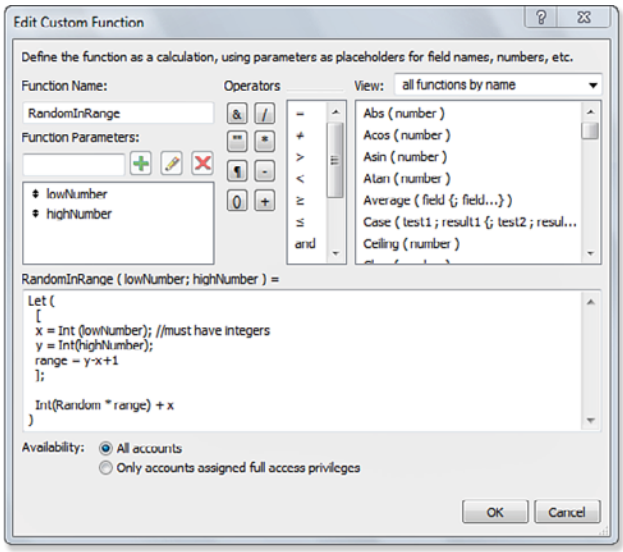


Figure 15.9
With FileMaker Pro Advanced, you have access to a Manage Custom Functions dialog.

Buttons from this dialog enable you to create, edit, and delete a custom function. The dialog shows the names of the parameters defined for each function, as well as whether a function is available to all accounts or just those with the Full Access privilege set. When you go to create or edit a custom function, you're taken to the Edit Custom Function dialog, shown in Figure 15.10.

Figure 15.10
You define the parameters and formula for a custom function in the Edit Custom Function dialog.



This dialog is similar in many ways to the standard calculation formula dialogs, so it shouldn't seem terribly unfamiliar. The main difference is the upper-left portion of the dialog where, instead of seeing a list of fields, you can instead name your function and its parameters. The restrictions for function and parameter names are the same as those for field names: They can't contain any mathematics symbols (such as + - * / ^ =); they can't contain the word `AND`, `OR`, `XOR`, or `NOT`; they can't begin with a digit or period; and they can't have the same name as an existing function or keyword.

There is no practical limit to the number of parameters you can define for a function, but most functions require anywhere from zero to a handful. The order of the parameters is important: When you use a function and specify the input parameters, they are interpreted as being in the order in which they are listed in the Edit Custom Function dialog.

The other significantly new and different portion of this dialog is the Availability section at the bottom. By default, a function is available to all user accounts. Any time a user or developer has access to a calculation dialog, he or she will see and be able to use all the unrestricted custom functions. The other option available to you is to restrict the use of the function to only those users who have the Full Access privilege set. The latter can be referred to as *private* functions, and the former can be thought of as *public* functions.

tip

When you are naming your custom functions and parameters, we think it's best to follow the same naming conventions used in the built-in functions. The initial letter of each word in a function name should be capitalized, and the name should contain no spaces or other punctuation. Parameters should be in *camel case*, with the first letter in lowercase and the first letter of subsequent words capitalized (for example, `numberOfCharacters`, `textString1`). Some developers prefer the function name itself in camel case as well.

note

If you find yourself writing a function that requires more than four or five parameters, that's a pretty good signal that you should break the function down into two or more smaller functions.

If access to a function is restricted, users who don't have full access never see that function. If those users ever view a calculation dialog that references a private function (say, in a script), `<Private function>` replaces the name of the function in the calculation dialog.

You might want to restrict access to a function for several reasons. As you will see in some of the examples in the section that follows, when you define recursive functions, you often need to define two functions to accomplish one goal. In those cases, the first function is often a public function, whereas the other is restricted, thereby keeping users from accidentally calling it directly.

Another reason to define a function as private is simply to keep from confusing novice developers. Your function might not be documented anywhere, and it might not contain adequate error trapping to handle improper parameter values. By making the function private, you reduce the risk that it will be used improperly.

tip

We find it helpful to place an underscore or a z (or zz) at the beginning of the name of private functions so that they are quickly and obviously identifiable. The z or zz at the beginning of the name of a private field, table, layout, function, or other object is a FileMaker convention. The underscore at the beginning of the name of a private variable is a programming convention in many languages, including Objective-C.

note

Declaring a function as private has no impact on what data is displayed or accessible to a user. The functions still do their jobs and work properly. It's just the functions themselves that can't be viewed.

Examples of Custom Functions

We think the best way to learn how to write your own custom functions is to study examples so that you can get ideas about uses in your own solutions. Some of the sample functions that follow might have intrinsic value to you, but the ideas and techniques are more important than the specific formulas. To that end, following each of the examples presented here, we provide commentary about the syntax and/or use of the function.

Hypotenuse (leg1Length ; leg2Length) =

```
Let ( [
   a2 = leg1Length * leg1Length;
   b2 = leg2Length * leg2Length;

   c2 = a2 + b2] ;
   Sqrt (c2)
)
```

Although FileMaker Pro provides built-in functions for many common mathematical formulas and operations, a number of common equations are missing. The preceding `Hypotenuse` function uses the Pythagorean Theorem (a2 + b2 = c2) to find the length of the hypotenuse of a right triangle given the lengths of the two legs.

Examples:

```
Hypotenuse (3 ; 4) = 5

Hypotenuse (5 ; 12) = 13
```

NthRoot (number ; root) =

```
Exp (Ln (number) / root )
```

This is another example of creating a custom function to provide an abstraction for a mathematical formula. There is a built-in function that returns the square root of a number but no function that returns the *n*th root of a number. The `NthRoot` function uses logarithms to find this number.

Examples:

```
NthRoot (8 ; 3) = 2

NthRoot (64; 4) = 4
```

Quarter (myDate) =

```
Ceiling ( Month (myDate) / 3)
```

This function returns the calendar quarter (1–4) of `myDate`. This function exemplifies the idea of using custom functions to substitute for code chunks, making your code easier to read and maintain. The `Month` function returns a number from 1 to 12, so taking the ceiling of that number divided by 3 yields an integer from 1 to 4.

Examples:

```
Quarter ("12/11/08") = 4

Quarter ("4/1/10") = 2
```

WeekEndingFriday (myDate) =

```
myDate + Mod (6 - DayOfWeek(mydate); 7)
```

Given a date, this function returns the date of the following Friday. This sort of functionality is often necessary in time-tracking systems so that you can summarize records by week. It would be easy to alter or extend this function to be referenced to some day other than Friday. To extend it, you would just specify a second parameter in the function and replace the hard-coded `6` (which is the `DayOfWeek` of any Friday) with a value derived from the parameter.

Examples:

```
WeekEndingFriday ("3/30/2013") = "4/5/2013 // the 30th is a Saturday

WeekEndingFriday ("3/24/2013") = "3/29/2013     " // the 24th is a Sunday
```

RepeatText (text ; numberOfRepetitions) =

```
text & Case (numberOfRepetitions>1; RepeatText (text; numberOfRepetitions - 1))
```

This is the first example of a recursive function. The `RepeatText` function returns *n* repetitions of the text string passed in the first parameter. For instance, `RepeatText ("t"; 3)` returns the string `ttt`. If the concept of recursive functions isn't clear to you, this is a good place to begin experimenting. Figure 15.11 traces through exactly what the function is asked to do when it evaluates this simple example. `RepeatText ("t"; 3)` is first evaluated as `t` and the result of `RepeatText ("t"; 2)`.

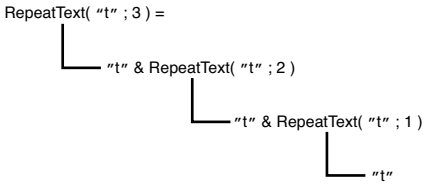


Figure 15.11
This diagram shows how the recursive custom function `RepeatText ("t" ; 3)` is evaluated.

Of course, the latter is then evaluated as `t` and the result of `RepeatText ("t" ; 1)`, which is simply `t`. The iteration stops at this point because `numberOfRepetitions` is not greater than 1. This is known as the function's *exit condition*; without one, you have endless recursion (picture a dog endlessly chasing its tail), which fortunately FileMaker Pro is smart enough to recover from after some large number of iterations.

caution
Be sure that any recursive function you write has some exit condition that is guaranteed to be reached.

Possible uses of the `RepeatText` function include creating progress bars or bar graphs. If you ever tried to do this sort of thing in previous versions of FileMaker, you know what a kludgy workaround was required to get a repeating string of characters. Another use is for padding out spaces when generating fixed-length data formats. Say that you need to pad out a FirstName field to 15 characters by adding spaces at the end. In previous versions of FileMaker, you would have used this formula:

```
Left (FirstName & "               " ; 15)
```

Using `RepeatText`, you could simply use this:

```
FirstName & RepeatText (" " ; 15 - Length(FirstName))
```

Of course, if you have a lot of padding to do, you might decide to abstract this one more layer and build the PadCharacters function shown next.

Examples:

```
RepeatText ("|" ; 10) = "||||||||||"

RepeatText ("hello"; 3) = "hellohellohello"
```

PadCharacters (text ; padLength; characterToPad; side) =

```
Let ( [
   padString = RepeatText (characterToPad; padLength - Length(text));
] ;
Case (
  Length (text) > padLength ; Left (text; padLength);
  side = "start"; padString & text;
  side = "end"; text & padString
)
)
```

Building on the preceding example, the PadCharacters function pads either leading or trailing characters onto a string. We used four parameters here to gain flexibility. The third and fourth parameters specify, respectively, the pad character and whether the padding should be at the start or end of the string. If you knew you always wanted to pad leading zeros, you could define this function with just two parameters and then hard-code the location and character within the formula.

Notice that this function makes a call to the RepeatText function to generate the padString. We could have included the formula for RepeatText, but by abstracting it out, we centralize the code for RepeatText (making it easier to troubleshoot) and also make the formula easier to read.

Examples:

```
PadCharacters ("foo"; 8 ; "x"; "end") = "fooxxxxx"

PadCharacters ("123"; 10; "0"; "start") = "0000000123"
```

TrimChar (text; removeCharacter; location) =

```
// valid locations are "start", "end", "all"
Let ( [
  leftChar = Left (text; 1);
  rightChar = Right (text; 1);
  remainderLength = Length(text) -1
```

```
] ;
Case (
  (location = "start" or location = "all") and leftChar = removeCharacter;
     TrimChar (Right(text; remainderLength) ; removeCharacter; location) ;
  (location = "end" or location = "all") and rightChar = removeCharacter;
     TrimChar (Left(text; remainderLength) ; removeCharacter; location) ;
   text
)
)
```

FileMaker Pro's built-in `Trim` function removes any leading and trailing spaces from a text string. Sometimes, however, you might need a more generalized way of removing a specific leading or trailing character from a string. The `TrimChar` function does just this. The first parameter is the string you want trimmed; the second is the character you want removed. The third parameter, `location`, specifies whether you want the character removed from the start or the end of the string, or from both. Valid inputs are `start`, `end`, and `all`.

This function works by checking whether the first or last character in the string needs to be lopped off. If so, the remainder of the string is fed back recursively to itself. Each iteration removes at most a single character; the "loop" continues until no more characters have to be removed, at which point the shortened text string is simply returned.

Examples:

```
TrimChar ("xxThis is a testxxx", "x", "all") = "This is a test"

TrimChar ("Another test¶¶¶", "¶", "end") = "Another test"
```

CrossProduct (array1; array2) =

```
_CrossProductGenerator (array1; array2; 1)
```

This, the final custom function example, looks at a more complex recursive function. In the recursive examples shown previously, the exit condition for the recursion was based on either an explicitly passed parameter reaching a certain value (`RepeatChar`) or a condition no longer being true (`TrimChar`). Other situations exist in which you want to be able to increment a counter with every iteration and base the exit condition for the loop on that counter reaching some threshold. The interesting part is that because the counter has to be passed along from iteration to iteration, it must be defined as a parameter. This means, however, that anyone using the function must initialize the counter for you, most likely setting it to 1.

The other solution is that you have a private function with a counter parameter called by a public function without one. In this case, the public function `CrossProduct` takes only two parameters, both expected to be return-delimited arrays. The function is defined merely to call another function, `_CrossProductGenerator`, which has three parameters. The first two inputs to `_CrossProductGenerator` are simply passed along based on the user's input. The third, however, is hard-coded to `1`, hence initializing a counter used there.

The syntax for the private function is as follows:

```
_CrossProductGenerator (array1; array2; counter)
```

It has the following formula:

```
Let ( [
  array1count = ValueCount (array1);
  array2count = ValueCount (array2);
  limit = array1count * array2count;

   pos1 = Ceiling (counter / array2count) ;
   pos2 = Mod (counter - 1; array2count ) + 1;

  item1 = TrimChar (MiddleValues (array1; pos1; 1); "¶" ; "end");
  item2 = TrimChar (MiddleValues (array2; pos2; 1); "¶" ; "end")
] ;

Case ( counter <= limit ;
   item1 & item2 & "¶" & _CrossProductGenerator (array1; array2; counter + 1))

)
```

The cross-product of two sets is a set containing all the two-element sets that can be created by taking one element of each set. For example, if Set1 contains { A, B }, and Set2 contains { P, Q, R, S }, their cross-product would consist of { AP, AQ, AR, AS, BP, BQ, BR, BS }. The number of elements in the cross-product is the product of the number of elements in each of the two sets.

The `_CrossProductGenerator` function "loops," incrementing a counter as it goes, until the counter is no longer less than the number of elements expected in the result set. Each time it iterates, it figures out what element number to grab from each list. With Set1 and Set2 of the example, the function would iterate eight times. If you were on iteration 5, the function would realize that it needed to grab the second item from the first list (because `Ceiling (5 / 4) = 2`), which is B, and the first item from the second list (because `Mod (4; 4) + 1 = 1`), which is P. That's how BP becomes the fifth element of the result set.

Notice also that this function, in addition to recursively calling itself, also calls the `TrimChar` function created earlier in this section. From the section on working with arrays, you'll remember that the `LeftValues`, `MiddleValues`, and `RightValues` functions return a trailing return after the item list; that trailing return has to be removed before the item is processed.

Examples:

```
CrossProduct ("A¶B¶C" ; "1¶2¶3¶4") =
"A1¶A2¶A3¶A4¶B1¶B2¶B3¶B4¶C1¶C2¶C3¶C4¶"

CrossProduct ("Red¶Yellow¶Blue" ; "-fish") = "Red-fish¶Yellow-fish
➥¶Blue-fish¶"
```

GETNTHRECORD

GetNthRecord merits its own discussion. In general, in FileMaker, if you are situated on one record and you want to see data from some other record, you need a relationship of some kind. This is intuitive; if you are on, say, a customer record and want to see data from an invoice, you need some kind of relationship between Customer and Invoice to accomplish this task. But this has also been true if you want to see data from somewhere else in the same table.

However, relational access has never covered all the possible scenarios in which you might want to access data from other records. What about when you're situated on a customer record, and you also want to know the names of the customers immediately before and after the current record? Or what if, when you're looking at a set of related invoices from the viewpoint of a customer, you want to get some specific information from the second related invoice record, or the third? It has always been *possible* to do these things in previous versions of FileMaker, but it has sometimes involved some cumbersome techniques. GetNthRecord solves these problems as well as a number of others. Its syntax looks like this:

```
GetNthRecord( fieldName; recordNumber )
```

Here, fieldName is the name of a field in the current table or a related table, and recordNumber is the number of the specific record from which to fetch data. Let's look at some examples:

```
GetNthRecord( CustomerName; 17 )
```

returns the value of the CustomerName field in the seventeenth record in the found set in the current table. The two expressions

```
GetNthRecord( CustomerName; Get(RecordNumber) + 1 )
```

and

```
GetNthRecord( CustomerName; Get(RecordNumber) - 1 )
```

return the value of the CustomerName field from the records immediately succeeding and preceding the current record, respectively.

```
GetNthRecord( InvoiceLineItem::ProductName; 3 )
```

returns the product name from the third line item related to a given invoice.

These applications are useful enough, but when you use some other advanced calculation techniques, some very interesting things are possible. For example, you'll often see cases in which you want to collect or aggregate non-numeric data from some set of records. Say, for example, you wanted to extract the personal names from a found set of records and present them in a comma-separated list. In the past, it would have been necessary to write a looping script to run through all the records and collect the results into a list. In FileMaker, a recursive custom function that invokes GetNthRecord can accomplish the same thing more economically. Consider a function that looks like this:

```
allNames( recordNum, currentList)
```

Consider its definition as shown here:

```
Case( recordNum > Get ( FoundCount ); currentList;
allNames( recordNum + 1; currentList &
Case( recordNum > 1;  ", "; "" ) &   Evaluate( "GetNthRecord ( name ;" &
➥recordNum & ")" )  ) )
```

Initially, you'd need to call this function with a `recordNum` value of 1 and a `currentList` value composed of an empty string. From there, the function keeps calling itself until `recordNum` is equal to the current found count. With each fresh function call, the value of the name field returned by `GetNthRecord` is appended to the list, and the list is passed back into the function again for the next iteration.

Note that it was necessary to use the `Evaluate` function here. The reason is that a custom function cannot directly access record data, such as the name field. Without the `Evaluate` function, when you attempt to save the function definition, FileMaker warns you that the name `name` is unknown. As a result, you have to build up the call to `GetNthRecord` as a text string, incorporating the current value of `recordNum`, and then pass that entire text string off to the `Evaluate` function.

As written, the function is designed to operate on data within the current table. It's rather limited in that sense, and we could certainly recast the function to be more extensible. It might be better to determine the total count of records from somewhere outside the function and pass that in along with the name of the field to be aggregated. Such a function might be called like this:

```
aggregateRecords( field; start; end )
```

And it might be defined something like this:

```
Case (start <= end ; GetNthRecord (field ; start ) &
➥Case (start < end; "¶"; "" ) & GetRelated (field ; start+1 ; end) ; "")
```

In this case, you need to decide for yourself what the end value would be; this is simply the total number of records you're trying to aggregate, and it could be the result of a `Get(FoundCount)` on the current file or a `Count()` operation against a related file. The function needs to be called with a start value of 1, unless you want to begin aggregating from a later record for some reason. So, a call to this function would look like this:

```
aggregateRecords( firstName; 1; Get(FoundCount) )
```

This would aggregate the firstName field across all the records in the current found set of the current table.

A recursive custom function, with or without an `Evaluate`, is probably one of the more complex pieces of coding you would need to do in FileMaker, but the results can be quite striking.

Troubleshooting

Text Formatting in Nontext Calculations

I want some of my dates to come out in red. I created some calculations that apply text formatting to certain dates, but they just don't work.

For a calculation containing text formatting functions to work correctly, the calculation must have an output type of Text or Number. Calculations defined to output a data type of Date, Time, or Timestamp will not show the effects of text formatting calculations.

However, if you use conditional formatting and the `Self` function, you can perform the necessary manipulation regardless of the field type.

Naming Variables in `Let` Functions

Can I use spaces in the names of variables used in `Let` *functions? Are the variable names case sensitive? What happens if I give a* `Let` *variable the same name as an existing field name, variable name, or function name?*

First off, yes, you can use spaces in the names of variables used in `Let` functions. Variable names can't begin with numbers, nor can they contain certain reserved characters (such as ; , \ / + - * = ( ) [ ] < > & and). You can, however, use characters such as $ and % in variable names.

Some complexity arises when we look at the possible use of script variables (variables beginning with $ or $$) within `Let` statements. We explore this complex topic in the following chapter. For now, suffice it to say that because various parts of a `Let` statement can work with script variables, you should avoid using $ or $$ in naming any of your `Let` variables.

Variable names within `Let` statements are not case sensitive. You can use a particular name several times within a function, and names can also be reused in separate functions.

There are no restrictions against giving variables the same names used for fields and functions. Be aware that any subsequent use of the name within the function refers to the local variable, not the field or function. With most functions, you don't need to worry about this, but names of functions that don't take parameters, such as `Random`, `WindowNames`, and `Pi`, should not be used for variables within a `Let` function. For instance, the formula `Let (Pi = "Hello"; Pi)` would return the string `Hello`, not the trigonometric constant pi that you might expect. As a rule, it's wise to avoid any overlap of names with reserved FileMaker names or names of objects elsewhere in the system. Even if the logic works, it might be confusing and hard to read.

Working with Arrays

I use arrays that have pipe characters as delimiters. Can I use the "values" functions to extract elements from these arrays?

The five "values" functions (`ValueCount`, `LeftValues`, `MiddleValues`, `RightValues`, and `GetValue`) operate only on return-delimited lists of data. If you have lists delimited by other characters, such as pipes or tabs, you would first need to do a substitution to change your delimiter into a

return. For example, if `myArray` is a pipe-delimited array, you could count the number of values in it with the following formula:

```
Let (tempArray = Substitute (myArray; "|"; "¶"); ValueCount (tempArray))
```

Of course, one of the reasons you might not have used returns as your delimiter in the first place is that your data elements could possibly contain return characters. If that's the case, you can't swap in returns as your delimiters and expect the structure of the array to remain unchanged. Before turning pipe characters into carriage returns, turn any existing carriage returns into something else—something that's guaranteed not to be found in an element and that's easy to turn back into a return character if necessary. You might, for instance, use the `Substitute` function to turn returns into the string `***RETURN***`.

FileMaker Extra: Creating a Custom Function Library

If you or your organization uses custom functions across several solutions, you'll likely want to create some sort of centralized library of the functions you develop. That way, when you find yourself in need of a particular function, you won't have to rack your brain remembering where you used that function before. In addition, centralizing the function library is one way to create a knowledge base that can help your organization leverage its past work and can aid in the training of new developers.

Your library can take many forms. One option, of course, is to create a FileMaker Pro file for your function library. Minimally, you'll want to include fields for the function name, its parameters, its formula, and a brief description. You might also use a container field to store a sample file for a particular function. Another "nice to have" is a related table for storing information about where you used the function.

As of the time of this writing, there's no way to move custom functions from one file to another using tools in the FileMaker product line, although cutting and pasting formulas to and from the library isn't terribly time-consuming. Custom functions are, however, part of the Database Design Report (DDR) that you can produce with FileMaker Pro Advanced. If you're handy with XML or are looking for a fun first XML project, you might want to use the XML output of the DDR to create your function library.

Finally, if you always want to have a particular set of custom functions in your files, create a sparse template file that has them in it. Then, rather than creating new files from scratch, you can just duplicate and develop on top of your template.

Matching Multiple Values

As you saw in Chapter 2, "Using FileMaker Pro," the Quick Start screen lets you filter the various Starter Solutions by typing in the field at the upper right. This feature is useful, and it can be implemented in a number of ways; the following is one of them.

First, create a field (it is usually a global) in a table to contain what is typed into the filter field. This can be a field in the table itself, and you can use a self-relationship, but it could be a field in another table. Now you need to create a relationship between the filter field and the field containing the words you want to filter. The problem is, what operator do you use?

One way to do this is to create a calculation field that contains each word separated by a carriage return. Because an equijoin matches based on any of the values in a field, this will do the trick. For example, if the value of the field containing the data is `computer furniture`, it will match `computer furniture` in an equijoin. However, if the data in the field is

```
computer
furniture
```

it will match either `computer` or `furniture`.

If the field containing the data is called `thing`, here is the calculation:

```
thing & ¶ &
MiddleWords ( thing ; 1 ; 1 ) & ¶ &
If ( WordCount ( thing )  >= 2; MiddleWords ( thing ; 2 ; 1 )  ; "" ) & ¶ &
If ( WordCount ( thing )  >= 3; MiddleWords ( thing ; 3 ; 1 )  ; "" ) & ¶ &
If ( WordCount ( thing )  >= 4; MiddleWords ( thing ; 4 ; 1 )  ; "" ) & ¶ &
If ( WordCount ( thing )  >= 5; MiddleWords ( thing ; 5 ; 1 )  ; "" )
```

This example creates a calculation, each value of which has something to match: the whole name, the first word, and then, for each of the next words, a value for each word.

16

ADVANCED SCRIPTING TECHNIQUES

What Is Advanced Scripting?

Chapter 9, "Getting Started with Scripting," presented an introduction to FileMaker Pro scripting techniques. It covered such topics as error trapping, linking scripts together via subscripts, conditional branching, looping scripts, and using custom dialogs. You should become familiar with all these essential scripting techniques.

This chapter explores several additional scripting techniques, including working with script variables, and script input/output techniques. Although we think that everyone can potentially benefit from learning these techniques, they do require a solid familiarity with general scripting techniques, calculation formulas, and the Relationships Graph. For this reason, we have opted to present these as advanced scripting techniques.

*To put these topics together with advanced layout and calculation techniques, **see** Chapter 18, "Advanced FileMaker Solution Architecture."*

Script Parameters

FileMaker 7 introduced script parameters, a means of passing inputs into a script. FileMaker 8 added to the picture by adding *script results*, the capability for a script to output a piece of data after it's finished executing. In FileMaker 9, an enhanced script debugger and a significant redesign of the script-editing interface brought everything together so that a truly powerful programming language is part of FileMaker.

The capability to move data in and out of scripts is desirable because it means that scripts can be written more abstractly and thus can be reused.

By *abstractly*, we mean scripts written to solve general problems rather than specific ones. Using script input/output saves you time, reduces the number of scripts that are necessary in your files, and makes your scripts easier to maintain.

By abstracting the information, you can create a script that, for example, goes to a layout as specified by a script parameter. Such a script can replace a multitude of other scripts such as `GoToCustomerLayout`, `GoToGroupLayout`, and so forth. If you are converting an older FileMaker solution to a current version of FileMaker, you will probably find that you can remove a lot of scripts by converting them to use parameters (*parameterizing* them).

Much of what there is to say about script input/output applies equally well to script parameters (inputs) and script results (outputs). We discuss script parameters first and then delve into a consideration of script results.

Script Parameters

Before we get into the details of how and why to use script parameters, a short example will give you a concrete sense of what script parameters are all about and why you want to learn this information. Imagine that you want to create several navigation buttons that take users to various layouts. One way to do this is to create a separate script that's hard-coded to go to a particular destination. You would need as many scripts as you have destination layouts, and every time you wanted to add a new destination, you'd create a new script.

Another way to accomplish this task is to create a generic Navigate script that navigates to a layout specified by the script parameter passed to it. Then, when setting up the buttons themselves, you would simply call the Navigate script, specifying the destination as the parameter. This approach has the advantage of requiring only a single script. To add another destination in the future, you simply specify the new destination as the parameter. There is no need to add a new script or to edit the original Navigate script.

It's clear from this example that extracting hard-coded values from a script and placing them instead into script parameters has a tangible benefit. Keep this example in mind as you read further about script parameters.

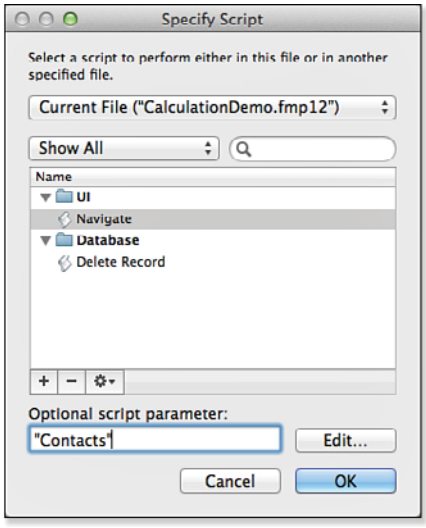
Parameters Versus Triggers

As you will see in Chapter 17, "Working with FileMaker Triggers," there is some overlap between script parameters and triggers. For example, you can attach a script to a button and launch a script with a parameter indicating which button has been clicked. With a field, you can attach a script to it that will be triggered when that field is entered. The trigger can specify the event and the script to run; it also can set the parameter for the script that indicates what object has been the trigger. In that case, the trigger sets the parameter to `Get ( ActiveLayoutObjectName )`. There is a certain simplicity in using the trigger because nowhere do you hard-code the name of the triggering object as you would with a button. Setting a script parameter automatically rather than hard-coding it is a more durable style of programming, although it is not possible to do so in every case. For that reason, you must be aware of both technologies.

Specifying Script Parameters

Script parameters can be set in several places: as part of a button definition, as an option for invoking a subscript within the `Perform Script` script step, or as part of the definition of a custom menu item. Figure 16.1 shows the first of these: the dialog for specifying which script should run when a button is clicked. The interface for specifying a subscript is the same; it, too, gives you a place to specify a parameter when calling a script.

Figure 16.1
When attaching a script to a button, you can also specify an optional script parameter, which is passed into the script.



At the bottom of this dialog, you have the option to specify a script parameter. The parameter can be some text string you type into the space provided, or you can enter a calculation formula as the parameter. Clicking the Edit button brings up a standard calculation formula dialog. If you use a calculation formula as your script parameter, when the button (or subscript) is triggered, the formula is evaluated and the results are used as the parameter.

The actual string or formula you use as your parameter completely depends on what you're trying to accomplish. Later in this section, you see some sample applications of script parameters.

note

Only scripts launched by buttons or triggers, launched through Custom Web Publishing, or called as subscripts of other scripts can have script parameters passed to them. Scripts launched through the Scripts menu or as startup/shutdown scripts (under File, File Options) cannot have script parameters passed to them.

Retrieving a Script Parameter

The `Get(ScriptParameter)` function can be used to retrieve the value of the parameter passed to a script. If no parameter was

specified, this function simply returns an empty string. The value of the script parameter can be accessed anywhere from within the script in this way. It can't be changed or altered in any way, and it expires as soon as the script is complete.

Any subscripts called by a script can have their own independent script parameters; they do not inherit the parameter of the script that calls them. As an example, say that the string `abc` was designated as the parameter to be passed to a script called Main Script. Assume further that Main Script called a subscript called Child Script as its second step, and that the parameter `xyz` was specified as part of the `Perform Script` step. Within Main Script, `Get(ScriptParameter)` always returns `abc`. Within Child Script, `Get(ScriptParameter)` always returns `xyz`.

The parameter passed to a script can be the result of a calculation, so by using `Get(ScriptParameter)` as the script parameter, as shown at the bottom of Figure 16.1, you can pass a script's parameter down to the subscripts it calls.

Passing Multivalued Parameters

The interface for specifying script parameters allows only a single value to be passed to a script. For many situations, that is sufficient to achieve the desired outcome. Other times, however, you will find that you want to be able to pass multiple parameters to a script. Although this isn't directly possible, there are several methods to achieve such a result.

Parsing a Text Array

The simplest way to pass multiple values in a script parameter is to specify a delimited array as the script parameter. For instance, if you wanted to send a parameter that contained the values `Fred`, `123`, and `Monkey`, you could send the string `Fred¦123¦Monkey`, or even `Fred+123+Monkey`.

To retrieve a portion of the passed parameter, use the built-in text-parsing functions of FileMaker Pro. If you've used carriage returns as your array delimiter, the `GetValue` function is the easiest way to extract a particular value. Say that you want to grab the third value from within your script, any time you wanted access to this value, you would use the following formula:

```
GetValue (Get (ScriptParameter) ; 3 )
```

note

The delimiter you use (here we've used pipe characters and carriage returns) is up to you; just choose something you know won't be found in the data you're passing.

*For more details on text-parsing functions, **see** Chapter 8, "Getting Started with Calculations," and Chapter 15, "Advanced Calculation Techniques."*

The nice thing about using delimited lists to pass multiple values is that you can set them up easily. Even if some of the values are derived as calculated results, it's still quite easy to set up a formula that concatenates all the appropriate pieces together. For instance, if you wanted to pass the current layout name and the current time as the two values of your script parameter, you would use the following formula:

```
Get (LayoutName) & ¶ & Get (CurrentTime)
```

note

The main drawback of this method is that the burden is on you, the developer, to know what each position in the array represents. There's nothing in the parameter itself that offers any assistance. This can (and should!) be clarified with script and/or calculation comments.

Using the `Let` Function

Another method for passing multiple values in a script parameter involves the `Let` and `Evaluate` functions. If you have a good understanding of those functions, you'll likely appreciate the elegance of this technique.

For more information on the `Let` *and* `Evaluate` *functions,* **see** *"Logical Functions,"* **p. 407.**

Imagine that you pass as your script parameter the following string:

```
"First Name = \"Fred\"; Favorite Number = 123 ; Favorite Animal = \"Monkey\""
```

What you have here is a set of name/value pairs, separated by semicolons. Immediately you can see one of the benefits of this method over the previous one: When you pass both names and values, the parameter becomes more meaningful. In six months when you need to troubleshoot or enhance your script, you won't have to rack your brain to remember what the elements in your parameter represent. Another benefit of this method is that the order of the values doesn't matter. They're retrieved by their name rather than by their position within the parameter.

You'll notice that within the parameter are backslashes before all the internal quotation marks. This process, known as *escaping* your quotes, is necessary any time you want to pass a string that contains internal quotation marks. For this technique, you need to escape the quotation marks surrounding any text values in your parameter; numeric values (such as the `123`) do not need quotation marks and hence don't need to be escaped.

You might recognize that the parameter specified previously is structured similarly to the first parameter of a `Let` function. This isn't a coincidence. Recall that the `Let` function allows you to set variables within a calculation formula. Imagine you had the following formula:

```
Let ([First Name = "Fred";
  Favorite Number = 123 ;
  Favorite Animal   "Monkey"
  ]
  ;
Favorite Animal)
```

This formula sets three variables (`First Name`, `Favorite Number`, and `Favorite Animal`) and then returns the value of the `Favorite Animal` variable. It would, in fact, return `Monkey`.

By combining the `Let` and `Evaluate` functions, you can build a formula that pulls out a named value from within a script parameter. The `Evaluate` function executes a dynamically constructed calculation formula. Therefore, within your script, any time you want to retrieve the value of the variable `Favorite Animal`, you would use the following formula:

```
Evaluate ( "Let ([" & Get(ScriptParameter)  & "];
➥Favorite Animal)")
```

As you can see, a string containing a `Let` function is dynamically assembled from the value of the script parameter. The `Evaluate` function is used to execute it. To return one of the other variables within the script parameter, you would simply need to change the end of the formula to reference the proper variable name.

tip

If you foresee a need to do much parsing of multivalued script parameters, you should consider creating a custom function to simplify the process even more. That way, you won't have to remember the syntax for the `Let` and `Evaluate` functions every time you need to retrieve a parameter value. Figure 16.2 shows the definition for a custom function called `GetParam`.

You need FileMaker Pro Advanced to create custom functions. For more details on creating custom functions, ***see*** *"Custom Functions,"* ***p. 427****.*

Figure 16.2
The custom function `GetParam` abstracts the script parameter-parsing routine even more.

The `GetParam` function takes a single argument, `paramName`. The formula for the function is the same as the `Evaluate` formula shown previously, but with the `paramName` inserted in the place of the hard-coded parameter name:

```
Evaluate ( "Let ([" & Get(ScriptParameter)  & "]; " & paramName & ")")
```

Now, within your script, to retrieve the value of the variable `Favorite Animal`, you just need the following formula:

```
GetParam ("Favorite Animal")
```

This final abstraction provided by the `GetParam` custom function certainly makes the parameter parsing more convenient. After it's in place, you can pass and retrieve multivalued script parameters with ease.

Passing Structured Data Elements in the Style of XML

The final method in this discussion for passing multivalued script parameters involves creating your own structured data elements. It's really a hybrid of the other two methods in that it requires standard text parsing to retrieve an element (like the first method), but the elements are meaningfully named (as in the second method).

The syntax you create for naming elements is up to you. We generally prefer an XML-like structure because it's easy to use and organize. For instance, to pass the same three values discussed in the preceding section, you might specify the following as your script parameter:

```
"<First Name>Fred</First Name><Favorite Number>123</Favorite Number><Favorite
Animal>Monkey</Favorite Animal>"
```

This is, of course, simply another way of specifying element names and values. But you don't need to worry about escaping any quotation marks, as you do with a string that will be used in an `Evaluate` statement. To retrieve the value of a particular element of the script parameter, you would need to use standard text-parsing functions. This is best accomplished with the creation of a custom function; you then need to write the parsing logic just once. The following formula could be used as the definition for such a custom function; the function's only parameter is `paramName`:

```
Let ( [
    openElement = "<" & paramName & ">";
    closeElement = "</" & paramName & ">" ;

    startPos = Position (Get(ScriptParameter) ; openElement ; 1; 1) +
    ➥Length (openElement);
    endPos = Position (Get (ScriptParameter) ; closeElement ; 1; 1)] ;

Middle (Get(ScriptParameter) ; startPos ; endPos - startPos)
)
```

If this function were called `GetParamXML`, the value of one of the script parameter elements could then be retrieved with the function `GetParamXML("First Name")`. The custom function is hard-coded to parse out a value from a script parameter.

Andy Knasinski has an XML-parsing custom function posted at www.briandunning.com/cf/1 that you can explore.

Strategies for Using Script Parameters

Using script parameters can greatly reduce the number of scripts in a file and can make your database much easier to maintain. You should consider using script parameters in several common programming scenarios.

Modularizing Scripts

The first—and most important—reason for using script parameters is to add a layer of abstraction to your scripts, thereby making them more modular and reusable. Instead of writing scads of single-purpose scripts, if you can generalize your scripts by using script parameters, you will need fewer scripts and your solution will be easier to maintain.

You will know if you have encountered a situation that can potentially be simplified and strengthened by using script parameters if you find yourself writing several scripts that do basically the same thing, differing only in some specific value. In place of that specific value, use `Get (ScriptParameter)` and then have the buttons or other scripts that trigger the script specify the particular value.

For example, say that you've developed a system that contains a calendar and that one of your layouts shows all the scheduled appointments for a given week. You'd like to be able to place a button above each of the seven days of the week (Sunday through Saturday) that users can click when they want to create a new appointment on that particular day. Assume that you have a field that contains the date of the Sunday of the week. Therefore, a script that would create a new appointment on Wednesday would do something like the following:

```
New Record/Request
Set Field [Appointments::AppointmentDate ; SundayDate + 3]
```

The scripts for creating appointments on the other days of the week would differ from what's shown in the preceding formula only by the constant that's added to the `SundayDate`. You could therefore write seven scripts, link them to your buttons, and move on to your next task.

We hope you can already see how and why script parameters can be used here. In the sample script, if you change the `+ 3` to `+ Get (ScriptParameter)`, you need only a single script to do the work of the seven required without script parameters. Each of the seven buttons calls the generic version of this `Add Appointment` script, passing as a parameter an integer from `0` to `6` to differentiate them from each other. By using this method, you replaced seven hard-coded scripts with a single generalized one.

Passing Data Between Files

Another situation in which script parameters can be beneficial is for passing data between files. Using script parameters for this purpose saves you from needing to create extra fields and relationships in your files.

As an example, imagine that you have a file called `Transactions` and another called `TransactionArchive` (each with a single table with the same name as the file). You periodically archive old transactions into the archive file, but occasionally you need to pull a record back from the archive into the main production file. Further, you'd like to avoid placing a table occurrence from the archive file in the main file because the two need to be able to function independently.

Because you can call scripts in another file without having a relationship to that file, script parameters make an ideal transfer mechanism for moving data between unrelated files. In the sample scenario, you might set up a script in the `TransactionArchive` file that calls a subscript in the `Transaction` file, passing a multivalued parameter (using one of the methods described in the preceding section) that contains the pertinent data elements from the transaction. In the `Transaction` file, then, your subscript would create a new record and populate it using the parsed-out parameter data.

In this example, importing the record from one file to the other would have been another solution within the defined constraints. Nonetheless, this example still clearly demonstrates the role that script parameters can play in moving data around. It's certainly preferable to copying and pasting data, or even parking data in global fields for later retrieval (both of which were common techniques with versions of FileMaker before version 7).

Script Results

Script results are, if you like, the flip side of script parameters. A script parameter lets you feed data into a script; a script result lets you pass data back out of a script. In the past, you might have done this by putting some data into a global field or a global variable for other scripts to look at later. But the best choice now is generally to use a script result.

Using a script result is the best choice because the script result is not stored automatically unless you choose to do so. Thus, a subscript can return a value, and the calling script can test that value and decide what to do next. If you use a global variable or global field, the value persists after you test it. In the future, you or someone else might look at the global field or variable and use its value for an unrelated purpose, possibly jeopardizing the logic of the script that relies on it.

To return a result from a script, you use the `Exit Script` script step to specify a result to return when the script exits. Much as when specifying the value for a `Set Field` or `Set Variable` script step, you can create a calculation expression that defines the result to return.

That takes care of how to return a script result. To access the returned result, you have to use the `Get(ScriptResult)` function, a sort of twin to `Get(ScriptParameter)`. `Get(ScriptResult)` hands back whatever result was returned by the most recently completed script or subscript.

Let's consider a full example. As we've suggested, one of the main reasons to use script input/output is to increase the reusability of your scripts. Consider a solution with a large number of reports. When you allow users to print reports, it's common to display the report in Preview mode first, pause the script, and then, on resuming, pop up a dialog box asking whether the user wants to print the report. The task of prompting the user for print confirmation might happen over and over again in a report-intensive solution. Using script results, you can write a single script to query the user and then return the user's choice as a script result. Here's what such a script might look like:

```
Show Custom Dialog [ Title: "Print Confirmation"; Message: "Would you
➥like to print the report?"; Buttons: "Yes", "No" ]
Exit Script [ Result: Let (
        [
        msg = Get(LastMessageChoice) - 1;
        choiceText = Choose ( msg ; "Yes"; "No" )
        ];
    choiceText ) ]
```

Notice the difference in the `Exit Script` step. As part of this step, the script specifies that a calculated result be returned from the script. The calculation looks at the numeric result of the dialog box choice, converts it into text using the `Choose` function, and returns the corresponding text result.

To use this script's modular functionality, another script has to call it. A script to display and optionally print a single report might look like this:

```
Go to Layout [ "Report" ]
Sort Records [ Specified Sort Order: Reporting::Region; ascending ]
➥[ Restore ]
Enter Preview Mode
Pause/Resume Script [ Indefinitely ]
Perform Script [ "Print Confirmation Dialog" ]
If [ Get ( ScriptResult ) = "Yes" ]
    Print [  ] [ No dialog ]
End If
Go to Layout [ original layout ]
Enter Browse Mode
```

This script performs all the usual sort of management common to previewing reports: navigating to a layout, sorting the records in some way, entering Preview mode, pausing for the user to look over the report. When the user resumes the script, though, the script goes straight into the confirmation subscript. Thereafter, the outer script uses `Get(ScriptResult)` to determine the result of the confirmation dialog and then prints the displayed report, or not, accordingly.

With such a script, instead of having a dozen print dialogs coded all over your system, you now have just one. If a user reports a problem with the print dialog, you now know where to start looking. And any changes or improvements made to the print confirmation process immediately benefit all reports that use this functionality.

Final Thoughts on Script Input/Output

The script input and output capabilities of FileMaker Pro represent a major advance in the capability to construct streamlined, reusable routines within a FileMaker solution. Mastering the use of these techniques is critical to getting the most out of FileMaker Pro. We recommend that you study these features carefully and that you aggressively look for opportunities to use them. Any time a script does similar work with different input values, consider using script parameters. Any time a script might be better structured as a tool that does some work and then reports on the results, consider reporting those results via a script result. Your solutions will become cleaner, simpler, and more elegant.

Script Variables

If you've worked with other languages or development environments, you're familiar with a variable as a type of named, temporary storage. For example, in the PHP programming language, you might write this:

```
$x = 7;
$y = 9;
$z = $x + $y;
```

Here `$x`, `$y`, and `$z` are all *variables*—temporary storage elements to which values are then assigned (the `$` in PHP indicates that these are variable names, and FileMaker uses a similar convention for variable names). They contrast with the permanent storage of fields in databases. In later expressions, the variable names stand in for the values stored in them. So, you'd expect that when the preceding program runs, the variable `$z` will end up storing a value of `16`.

Often, as you build up a program or routine, you'll find yourself wanting to rely on named, temporary storage elements like these. In previous versions of FileMaker, the only place to cache such data was within FileMaker's database structures, by putting the data into one or another kind of field.

Consider the simplistic example of a script that beeps ten times in succession. Previously, you might have defined a field with global storage to act as a counter and written the script like this:

```
Set Field [Loop::gCounter; 1]
Loop
    Beep
    Pause/Resume Script [Duration (seconds):1]
    Set Field [Loop::gCounter; Loop::gCounter + 1]
    Exit Loop If [Loop::gCounter > 10]
End Loop
```

This approach has always worked fine, but it has some drawbacks:

- Even though the counter field is used only in this script, it has to be defined within the field definitions for the table as a whole. It will always be there, cluttering up the list, even though it might apply to only a single script.

- The storage is not as temporary as you would like. The field `gCounter` goes on holding its value and being accessible after the script completes. This is one reason you need to reset the field to `1` at the start of the script. If the script has run previously, it might still have its old value of `11`, or it might have some other value altogether if someone edited the field directly and stored it in the database.

note

If you are working with a FileMaker database that has its roots in the past, you might encounter a number of global fields. Converting them to variables is a routine part of conversion to the current versions of FileMaker; doing so can make the databases smaller and the scripts simpler and most robust. One of the great recommendations of variables is that they are temporary, so they cannot hang around to take up storage and confuse future developers who work on the database.

About Local Variables

A *local variable* is one that exists and has meaning only within a single script—exactly what you want for the loop-counting example shown previously. If you were to rewrite the looping script using script variables, it might look like this:

```
Set Variable [$counter; Value:1]
Loop
    Beep
    Pause/Resume Script [Duration (seconds):1]
    Set Variable [$counter; Value: $counter + 1]
    Exit Loop If [$counter > 10]
End Loop
```

Local variables are named using a single dollar sign ($), and they're created and manipulated using the `Set Variable` script step. Figure 16.3 shows the options for the `Set Variable` script step.

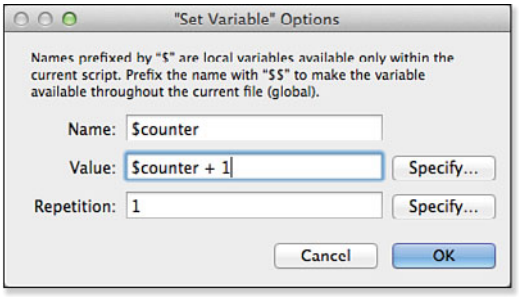


Figure 16.3
The capability to set variables is a powerful feature in FileMaker Pro.

You'll notice that the `Set Variable` script step enables you to set the variable's value dynamically, using the Specify option. This means that a variable can be used to hold the results of any expression you can define using FileMaker's Calculation dialog. You could store the current username, the current date, or the results of an expression such as `$counter + 1`. You'll note also, by the way,

that variables can be referenced from within such calculations just by using the variable name. For example, the following are perfectly valid calculation expressions:

```
$counter + 1

"Name: " & $userName
```

If a variable of the specified name is not currently defined, a reference to the variable returns an empty string (or a *null result*). The first expression of the preceding two would give a result of `1`, whereas the second would give a result of `"Name: "` if the respective variables were not defined.

About Variable Scope

Variable scope is the area in which a variable has meaning and can be accessed. Variables in FileMaker have one of two kinds of scope: either local scope or global scope.

So far, the variables we've examined have local scope. It's most common to refer to them as *local variables*. A local variable exists and has meaning only within the context of a single script. Consider the example of the `$counter` variable discussed previously. This variable exists only within the script in which it's defined and manipulated. After the script completes, the `$counter` variable is undefined and cannot be accessed. Likewise, if you called a subscript from within that script, the subscript would not have access to the value of `$counter` contained in the parent script.

tip

Because local variables exist only within the context of a single script, this can lead to subtle confusion. Consider a parent script that uses a `$counter` variable that then also calls a subscript. If you were to try to access the value of `$counter` within the subscript, you'd get a null value because you'd be trying to access a variable that had never been set within the context of the subscript. And if you were to try to *set* the value of `$counter` within the subscript by using the `Set Variable` script step, this would create a new variable, local to the subscript, with the same name, `$counter`. There would consequently be a total of two `$counter` variables: one local to the parent script and one local to the subscript. The two exist simultaneously and independently; they don't conflict, and they don't affect one another.

Local Variables Summary

So, let's summarize what has been said about local variables:

- They are set using the `Set Variable` script step.
- They must have names beginning with `$`.
- They can be referenced within calculation expressions.
- They are limited in scope to the script in which they are defined (via the `Set Variable` script step). Neither subscripts nor parent scripts can access the value of a local variable.
- They do not appear in the Manage Database dialog.

About Global Variables

Global variables, denoted with a double dollar sign (`$$userName`, `$$currentLayout`), share many features with local variables. The only difference is in their scope. Whereas local variables are limited in scope to a single script, global variables retain their value no matter what script is running or whether a script is running at all. They can store values that persist across any or all scripts for the duration of a user's session.

The last point bears repeating. Whereas local variables have *script scope*, meaning that they are limited in scope to a single script, global variables have *file/session scope*. Like globally stored fields, global variables are unique to an individual user: Each user has his own copy of a global variable, so the variable `$$userName` can have a different value for each active user. In addition, global variables cease to exist when a user session ends. If you work with a global variable, quit FileMaker, and then open the same file again, the global variable will disappear, until some logic in the files creates it again.

Global variables also have scope only within a single file. There is no way to "reach across" to pull a global variable's data from one file into another. Such a thing *is*, by contrast, possible with globally stored fields. Global variables from other files cannot be accessed via relationships because they don't appear in the field list.

So, what good are global variables? When does it make sense to use them? We recommend that, by and large, you use global variables for *user session data*: data specific to one user that is intended to persist for just that user session. Examples include the name of the currently logged-in user, user preferences such as a user's chosen default layout, or any other user-specific data you might be storing, such as a user department or sales region—particularly when this information is not available from a database.

Global variables cannot completely obviate the need for globally stored fields. Globally stored fields have several capabilities not shared by global variables:

- Globally stored fields can be accessed across files by using relationships.
- Globally stored fields can accept user input.
- Globally stored fields can be used to drive relationships.
- Globally stored fields can be used to store the content of an input field from a custom dialog.

For example, if you were implementing a *filtered portal* (a portal whose contents change in response to user input), you would have to use a globally stored field to do so, both because you would need to capture user input and because you would need to use that input to drive the portal relationship.

Other Ways to Work with Variables

When you're first starting out with variables, we recommend you try to stick to the following precepts until you feel you've mastered the basics:

- Use local variables for temporary storage used within the context of a single script.
- Use global variables to store user-specific session data (with the exceptions noted in the next point).

- Use globally stored fields, not variables, to store user-specific session data that must be captured directly from the user, must drive a relationship, or must be shared heavily across files.

Now that we've said all that, if you have mastered the basic concepts of variables, there are some advanced points to be made about them.

About Dynamic File Paths

There's another nice feature of variables in FileMaker Pro that's very much worth mentioning. Certain script steps, such as `Export Records`, as well as the `Save Records as Excel/PDF` script step, allow you to specify the location of an output file by typing in a file reference. In FileMaker Pro, that file reference can be taken from a variable, rather than being hard-coded.

If such usefulness isn't obvious, let it sink in for a moment. In the past, it wasn't possible to create names for exported files on the fly: You either had to let the user enter a filename or had to hard-code a single specific filename into the script step. If you wanted to name exported or saved files dynamically (say, you wanted to include the current date in the filename), you were out of luck, unless you chose to use a third-party plug-in.

To save files to a dynamically specified file path, you need to create that file path in your script and put it into a variable. (The path begins with the user's desktop, so that is where the file is placed.) That variable can then be used in specifying a file path, as the following script example illustrates:

```
Go to Layout [ "Contacts" ]
Show All Records
Set Variable [ $filePath; Value: Let (
  [
    theDate    = Get(CurrentDate);
    theYear    = Year(theDate);
    theMonth = Month(theDate);
    theDay     = Day(theDate);
    dateText  = theYear & "_" & theMonth & "_" & theDay;
    filePath    = Get ( DesktopPath ) & "Export_" & dateText
  ];
  filePath ) ]
Save Records as PDF [ File Name: "$filePath"; Records being browsed ]
```

Viewing Your Variables

One final note on variables in FileMaker Pro. We've made the point a few times that variables are beneficial in that they don't add clutter to the database schema: They don't appear in the Manage Database dialog, nor do they appear in the field lists that go along with operations such as `Sort` or `Import Records`. There's a disadvantage to this as well: There's currently no way to see a list of all the variables currently active in a FileMaker solution.

It is possible to view the values of individual variables in the FileMaker Pro Advanced Data Viewer, but you must enter the variable names one at a time, as with any other expression.

FileMaker Extra: Recursive Scripts

Chapter 15 discusses how you could make custom functions recursive by including calls to themselves within their formulas. In a similar manner, you can use script parameters to create recursive scripts. Although this isn't something you need to do on a daily basis, there are some interesting applications for recursive scripts.

A *recursive script* is one that calls itself repeatedly until some exit condition is satisfied. Each time the script calls itself as a subscript, it passes a script parameter that can be used as part of an exit condition test. In many ways, recursive scripts are quite similar to looping scripts, and many tasks you can accomplish with one can be done as easily by the other. As an example of a recursive script, consider this `Recursive Add` script:

```
If [Get (ScriptParameter) >= 100]
    Exit Script
End If
New Record/Request
Perform Script ["Recursive Add"; Parameter: Get (ScriptParameter) + 1 ]
```

This script adds 100 new records to the current table. It's first called without a script parameter, so the first time through, the script calls itself as a subscript, passing a parameter of `1`. The parameter increments each subsequent time through until eventually the exit criteria (`Get (ScriptParameter) >= 100`) are met.

If there are any steps in the script after the recursive subscript call, they are all executed, from the inside out, after the exit criteria have been met. Try to predict what would happen if you added the following steps to the end of the preceding script:

```
Beep
Show Custom Dialog ["The parameter is:" ; Get (ScriptParameter)]
```

The 100 records would be created exactly as they were originally. But after they were all created, you would hear a beep and see a message telling you that the script parameter value is 99. After clicking OK, you would then hear another beep and see a message telling you that the parameter is 98. This would continue for some time, and eventually the last message you'd see would be that the parameter is empty, which, of course, was the condition on the first trip through the script.

As a final example of recursive scripting, consider the following script, which flags duplicates among a set of records. Assume that the set contains a list of names, which has been sorted by name before this script is called:

```
If [IsEmpty (Get (ScriptParameter))]
    Go to Record/Request/Page [First]
Else
    Go to Record/Request/Page [Next; Exit after last]
    If [Get (ScriptParameter) = Contacts::Name]
```

```
        Set Field [Contacts::DuplicateFlag; "Duplicate"]
    End If
End If
Perform Script ["Mark duplicates"; Parameter: Contacts::Name]
```

During each iteration through the script, the current record's name is compared against the value of the script parameter, which was set to the value of the previous record's name. The exit condition here is the `Exit after last` option on the fourth line; the script continues through the set of records, stopping only when there's no next record to go to.

This page intentionally left blank

17

WORKING WITH FILEMAKER TRIGGERS

Introducing FileMaker Triggers

Triggers are nothing new to FileMaker or to programming in general. They are operations (scripts, in the FileMaker world) that run when certain events occur. These events can be a click in a field, a click out of a field, a change from one mode to another (Browse to Preview to Layout to Find), the opening or closing of a file or window, the selection of a tab, or a change in a data value. Once the trigger is established—and that means defining the action to be taken as well as the event that causes the trigger to run—everything works by itself.

note

A trigger is said to "fire" when the event occurs and the action is started.

FileMaker Triggers Before FileMaker Pro 10

Some triggers have always been present in FileMaker, although the term was not always used. The previous FileMaker triggers are as follows:

- **Calculations**—Like spreadsheets, FileMaker has logic that makes certain that calculations are updated as needed when fields that they reference are modified.
- **Lookups**—Likewise, FileMaker keeps track of when a Lookup field needs to be updated.

- **File Open and Close**—You can use File, File Options to specify scripts to run when the file is opened or closed.
- **Buttons**—In Layout mode, you can use Format, Button Setup to attach a script to the selected interface element (the interface element can be a button, a graphic, or even text).

Triggers in FileMaker Pro Today

As described previously, a trigger consists of an event that, if it occurs, causes a script to run. The trigger must have both components: Without a script, there is nothing to run, and without the event, there is nothing to cause the trigger to fire.

The scripts associated with triggers are just like any other scripts, although they may use some new functions, described later in this chapter in "Trigger Functions," **p. 468.**

A trigger has several components:

- **Name**—This is the name of the type of trigger, not the name of a trigger associated with a specific field, button, or the like.
- **Target**—This is the thing that is acted upon—a field, a button, a layout, a file, or the like.
- **Event**—This is the thing that is done to the target—a tap or mouse click, for example.
- **Timing**—This determines whether the trigger fires before the event is processed (but after it has been initiated) or after the action has taken place. In other words, when an event is intercepted, is the trigger the first or the last thing to happen? The two values are Pre and Post. For Pre triggers, you can supply a script that returns false (0) to stop the event from occurring. A return value of 1 lets the event continue.
- **Result**—A trigger may have a result. Pre triggers most often will return a result that can determine whether the action should proceed.

Triggers, FileMaker Pro, and FileMaker Go

Like layouts and the Database Relationship Graph, triggers are created and modified in FileMaker Pro, but they are active in FileMaker Pro and FileMaker Go.

Trigger Targets

There are three primary type of targets in FileMaker Pro and FileMaker Go: objects, layouts, and files. Layouts and files are common concepts, but a precise discussion of trigger object targets is in order.

In general, trigger objects are objects in layouts that can have keyboard focus: You can control them from the keyboard. These objects include fields, portals, web viewers, radio buttons, pop-ups, buttons, and check boxes. If you can type into it, tab into it, or activate it with the Return/Enter key, it can obtain keyboard focus and it can be an object target of a trigger.

note

This section describes general behavior; some triggers implement a subset of these features.

These types of objects can have triggers attached to them, but unless the specified event occurs, the triggers will not fire. This means that if you have a field on a layout that is set not to allow entry, an associated trigger will not necessarily fire because the event cannot occur.

Trigger Events

A single user action can trigger a variety of trigger events. For example, clicking in a field in a portal row can trigger `OnObjectEnter` events for both the field and the portal, but this is not always the case. If the portal row is selected (perhaps because another field has been clicked), only the field's `OnObjectEnter` event is triggered. In deciding how to set up your trigger settings, consider what you really want to capture: events at the field level or events at the portal row level. Perhaps you want both types of events to fire triggers, but they would probably be very different types of triggers.

A Typical FileMaker Event Handler Chain

Today's operating systems are generally event driven. At various levels, event handlers (triggers in FileMaker Pro) can respond to events. Usually, each event handler can pass the event that triggered it on to the next handler in the chain; it also can stop the chain so that events are dropped. An example of taps, clicks, and keystrokes demonstrates this. Here is a conceptual view of what happens to taps, clicks, and keystrokes made to a FileMaker window:

- The operating system gets to respond. It may decide not to pass certain events along. An example of this is a key that increases or decreases speaker volume on a laptop: That keystroke event goes to the operating system, which adjusts the speaker volume. It is not passed along.
- If the event is passed along, FileMaker Pro receives that event. Similarly to the speaker volume keystroke, it may decide that this keystroke should be handled by FileMaker Pro and not passed along. FileMaker Pro checks to see which layout object should receive the keystroke (if it is in a field that does not allow entry in the current mode, it is passed along to the containing object).
- The layout object (such as a field) to which the tap, click, or keystroke was directed can act. If an `OnLayoutKeystroke` trigger is installed, it runs. If it returns a result of true, processing continues with the next step.
- If there is no `OnLayoutKeyStroke` trigger or if no layout object received the event, the layout itself receives the keystroke and fires its own `OnLayoutKeyStroke` trigger. If it returns a result of true, processing continues with the next step.
- Navigation keystrokes are handled by FileMaker (for example, by tabbing into the next field). Other keystrokes are handled by the active object.
- If no object has handled the tap, click, or keystroke, FileMaker Pro posts an alert, beeps to indicate that the keystroke is inappropriate, or, in some cases, does nothing. (For example, a tap or click in a data-entry-enabled field selects it. A tap or click in an already selected field has no effect.)

Note that if triggers are installed for both layout objects and the layout itself, those triggers can stop processing of Pre events by returning false as the trigger script's result. In most cases, the trigger

script will perform its own action in response to the keystroke. That action should be a logical and consistent action. Do not redefine standard modifier keys or keystrokes.

Triggers and Underlying Data

In the case of layout objects, it is clear that the triggers associated with them fire only when someone interacts with the layout itself either directly or through a script. However, you may think that a trigger such as `OnObjectModify` would be triggered when the object's data is modified. As you think it through, you will realize that a trigger such as `OnObjectModify` will fire only when the layout object itself is modified. A modification to the underlying data (such as displaying the result of a changed calculation in a field) is not a modification to the layout object itself. (As noted previously, the updating of the data display is a form of conceptual trigger, but it is not part of the FileMaker Pro 10 triggering mechanism.)

Triggers and Web Publishing

When Instant Web Publishing (IWP) or Custom Web Publishing (CWP) is used, the FileMaker web experience is different in one very important way from the basic FileMaker Pro experience: The window with which the user interacts is a browser window, not a FileMaker Pro window. One of the important consequences of this is that whereas FileMaker Pro can keep track of user events such as entering fields and changing data, on the Web, FileMaker sees only the overall result of such events when a user clicks a Submit button or its equivalent.

Because FileMaker cannot know what the user is doing in browser interactions, triggers on the Web occur only if the event (such as entering a field) is caused by a script step. Because FileMaker is running the script (even on the Web), it can know whether a field is entered and can fire a trigger appropriately.

File Open/Close triggers do fire on the Web because FileMaker has to actually open the file so that it knows that it has to fire a trigger.

Attaching Triggers

Triggers can be attached to different objects. The next sections explain how you can attach triggers to layouts, objects, and windows.

Layout Triggers

You can attach triggers to layouts using the Layout Setup dialog, as shown in Figure 17.1.

As you see in the dialog, each event for the layout can have a script attached to it. The combination of event and script creates a trigger. You can attach only one script to an event, although that script can call other scripts, as is the case with any script.

The Layout trigger events are shown in Table 17.1.

Figure 17.1
Set triggers for layouts.

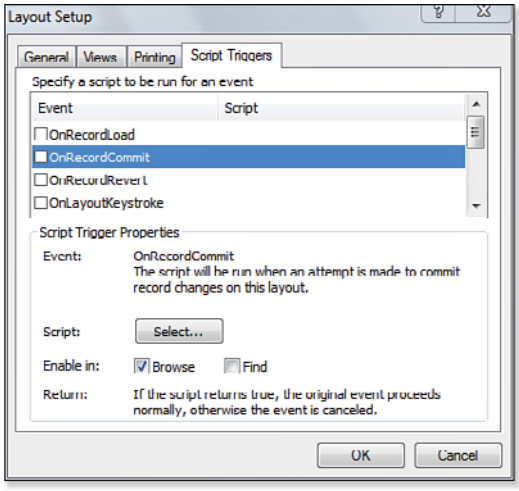


Table 17.1 Layout Trigger Events

Name	Pre/Post
`OnLayoutEnter`	Post
`OnLayoutExit`	Pre
`OnLayoutLoad`	Post
`OnLayoutKeystroke`	Pre
`OnModeEnter`	Post
`OnModeExit`	Pre
`OnViewChange`	Post
`OnRecordLoad`	Post
`OnRecordCommit`	Pre
`OnRecordRevert`	Pre

Object Triggers

You attach triggers to an object in a layout by selecting it and choosing Format, Set Script Triggers in Layout Mode, as shown in Figure 17.2.

The Set Script Triggers command opens the dialog shown in Figure 17.3, which is similar to the layout dialog shown in Figure 17.1. In the case of events that are available both at the layout and object level, make certain you know which will fire first.

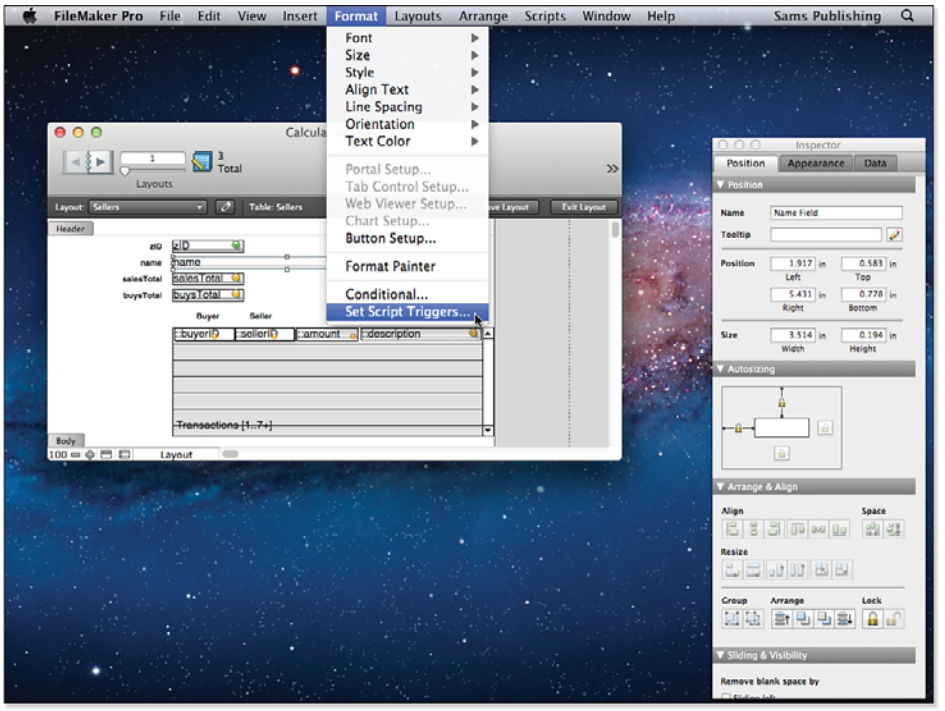


Figure 17.2
Set an object's triggers.

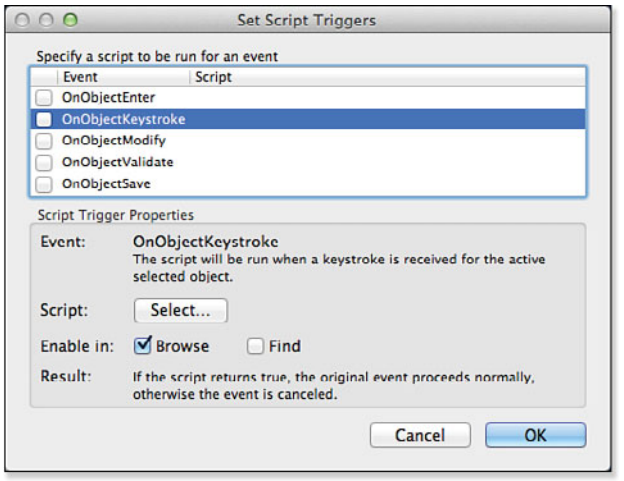


Figure 17.3
Specify the script for each trigger.

The triggers shown in Table 17.2 apply to objects on layouts.

Table 17.2 Object Trigger Events

Name	Pre/Post
`OnTabSwitch`	Pre
`OnObjectEnter`	Post
`OnObjectExit`	Pre
`OnObjectModify`	Post
`OnObjectValidate`	Pre
`OnObjectKeystroke`	Pre
`OnObjectSave`	Post

Window Triggers

A set of script triggers is available to you in the File Options command in the File menu, as shown in Table 17.3.

To review information on script triggers, **see** *Chapter 9, "Getting Started with Scripting."*

Table 17.3 Window Trigger Events

Name	Pre/Post
`OnFirstWindowOpen`	Post
`OnLastWindowClose`	Pre
`OnWindowOpen`	Post
`OnWindowClose`	Pre

Renamed Triggers Starting in FileMaker Pro 12

Note that in versions of FileMaker Pro before FileMaker Pro 12, `OnFirstWindowOpen` was called `OnFileOpen`, and `OnLastWindowClose` was called `OnFileClose`.

Using a Timer

There is a script step, Install On Timer Script, that installs a timer for a window. In some ways, the timer is like a trigger. The timer has two components:

- A calculation that determines how many seconds to wait before firing
- A script to execute when the timer fires

Each window has its own timer (and only one timer). When you install a timer with the new script step, it overwrites any previous timer for that window.

The default timer has a wait time of zero, which means that it will never run until it is replaced by a timer you set.

Trigger Functions

Several functions implement triggers in a variety of ways. Triggers reverse the traditional flow of scripting in some ways. Before the existence of triggers, layout objects that caused scripts to run had to have scripts attached to them in the Format Button dialog. Parameters can be specified in that dialog to control the execution of the script.

Scripts can be triggered when specific events occur: when a file is opened, when an object is modified, and when a layout is loaded, for example. In many ways, the trigger is a side effect of a user's action. The user enters some data in a field on a layout, thereby modifying that object, and with no additional action on the user's part, a trigger causes a script to run. In part because there is no button whose sole purpose is to start a script in these cases, the script that is started cannot always know how or why it was started. This is the reverse of a common scripting paradigm in which parameters are passed in to the script when a button is clicked, and those parameters indicate to the script what has happened (what object has been clicked) and what should be done.

The functions in the following section help implement the mechanism by which a script needs to find out how it was triggered. In addition, several field functions that previously were classified in another area have been moved here because of the commonality they have with the trigger-supporting functions.

The `Self` Function

One of the features added in FileMaker Pro 9 was the `Self` function. This function lets you access the value of an object. It is applicable to calculations defined within fields. The function takes no parameters: The single word `self` provides the object's value.

If you want to use conditional formatting to change the appearance of a field, you can use the `Self` function to do so. For example, if you set up conditional formatting where the formula to be used is `Self = "test"`, typing **test** into the field will trigger the conditional formatting as soon as you click out of the field.

So far, there is no improvement over using the field's value itself for the conditional formatting. However, you can demonstrate the value of this function by copying the field (which also copies its formatting) and then changing the field's content to another field in the database. The original field might show the database name field, and the copy might now show the database address field. But

the conditional formatting for both fields is triggered if the field's content is `test`—regardless of what the underlying database field is.

By setting conditional formatting in this way, you can create a layout field object that can be used for a variety of database fields but displays the same conditional formatting regardless of the field. One real-world application of this would be a layout field with conditional formatting attached to it that always flags numbers outside a specific range.

tip

You can also use the `Self` function in auto-enter and validation calculations.

Char and Code Functions

FileMaker Pro has two character functions that are common in programming languages but relatively new to FileMaker Pro: `Char()` and `Code()`. In their simplest form, they let you convert a character to its numeric code, and vice versa. This capability is particularly useful in dealing with special characters such as the Return key when working with script triggers. Because FileMaker Pro supports Unicode, these functions actually work on Unicode code points, but many people refer to ASCII codes, which are a subset of Unicode code points.

- **`Char ( number )`**—This function returns the character for a Unicode code point. It may return more than one character if the number string describes multiple characters.
- **`Code ( text )`**—This is the reverse of the preceding function; it provides a text string (often a single character), and the function returns its numeric value. This function is often used with `Get ( TriggerKeystroke )`, which is described in Chapter 8, "Getting Started with Calculations."

The `GetFieldName` Function

This function lets you find out the name of a field. This matters particularly when you are using a parameter in a custom function or when you are using `self` in conditional formatting. Neither `self` nor the parameter discloses what the underlying field's name is. For that purpose, use `GetFieldName`—note the absence of parentheses.

The `Get (TriggerKeystroke)` and `Get (TriggerModifierKeys)` Functions

Two functions support the implementation of triggers discussed in Chapter 4, "Working with Layouts": `Get ( TriggerKeystroke )` and `Get ( TriggerModifierKeys )`.

`Get ( TriggerKeystroke )` returns the character that triggered a script triggered by `OnObjectKeystroke` or `OnLayoutKeystroke`. Frequently, you want to check for a special character such as the Return key. A simple way of doing so is to use the `Code` function to convert the character to a numeric code and to test for that. The following line of code tests for the Return key (the numeric value of which is 13):

```
If [ Code ( Get ( TriggerKeystroke ) ) = 13 ]
```

`Get ( TriggerModifierKeys )` returns whatever modifier keys were pressed when the script was triggered.

These functions are used frequently in conjunction with script triggers when you are trying to determine what keys have triggered the script. The numeric values of modifier keys are shown in Table 17.4.

Table 17.4 Numeric Values of Modifier Keys

Key	Number
Backspace	8
Tab	9
Shift-Tab	9
Enter	10
Return	13
Escape	27
Left arrow	28
Up arrow	29
Right arrow	30
Down arrow	31
Space	32
Forward delete	127

FileMaker Extra: Using Triggers for an Interactive Interface

Triggers are not good for performing data validation, in part because they do not fire when certain batch processes (or web updates) are done. But they are great to use in many other cases, including in an interactive interface.

Consider the traditional way of enforcing certain fields to be entered: You require them not to be blank by setting an option for the field in the database. This is still the best and most reliable method because if you do not allow it to be overridden, that edit will ensure that there are values in all the fields.

But you can supplement that edit with an interactive interface. There are many ways of doing this. One way is to use the `OnObjectModify` trigger (a Post trigger) to check not only what has been modified, but also the status of other fields on the layout. You can then modify a field on the layout that displays suggestions for what might be entered next. This enables people to fill in a form in whatever sequence they want without skipping any required fields.

18

ADVANCED FILEMAKER SOLUTION ARCHITECTURE

This chapter focuses on a variety of architectures and techniques used in FileMaker solutions. You might encounter them as you modify and update existing FileMaker solutions; you also might adopt them in new solutions that you create. Many of these techniques involve managing windows, which, after all, are at the heart of traditional graphical user interfaces. As time and technology have changed, the role of windows in interfaces has changed. Most significantly, on mobile devices the concept of multiple windows has changed. What might have been implemented as multiple windows on a desk- or laptop computer may now be implemented by several views within a single window or even by a sequence of views. Building a solution to run on FileMaker Go means that you will not have multiple windows visible at the same time.

*For more information on interface design, **see** Apple's iOS Human Interface Guidelines and Mac OS X Human Interface Guidelines at developer.apple.com.*

Window Management Techniques

Among the many important features of FileMaker, the capability to have multiple windows showing data from the same table stands out as one of the most important. To aid developers with managing this feature, several window management script steps are present in ScriptMaker, including the following:

- `New Window`
- `Select Window`

- `Close Window`
- `Move/Resize Window`
- `Adjust Window`
- `Set Window Title`

Also, 11 `Get` functions return data about the active window, ranging from its size and location to its name and the mode it's in. Another function that plays a role in window management is `WindowNames`, which returns a list containing the names of all the open windows, ordered according to the stacking order of the windows. (`WindowNames` is listed as a Design function rather than a Get function.)

These script steps and calculation functions provide you with tremendous ability to control the user experience. The amount of window management you do might vary widely from solution to solution, but having a good grounding in the options available to you is important.

When you create, move, and resize windows, you have the opportunity to specify both a location for the window and its size. The unit of measure for all window manipulation is the point. Figure 18.1 shows the options for the `Move/Resize Window` script step.

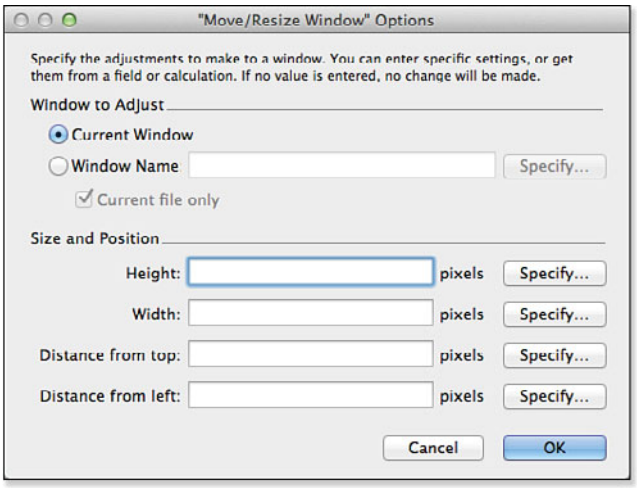


Figure 18.1
The `Move/Resize Window` script step enables you to specify the exact coordinates (in points) and size for any given window.

For each parameter of the `Move/Resize Window` script step, you can either specify a literal number or supply a calculation formula whose result determines the parameter's value. If you leave any of the parameters empty, their values are inherited from the current active window. For instance, if you merely want to move the current window without changing its size, you don't have to specify anything for the `Height` and `Width` parameters.

The `Adjust Window` script step includes a Resize to Fit option, which resizes the window to the layout in which it is displayed. If you use a script to go to the appropriate layout and immediately adjust the window with Resize to Fit, the window will be just the right size for the layout.

Multiwindow Interfaces

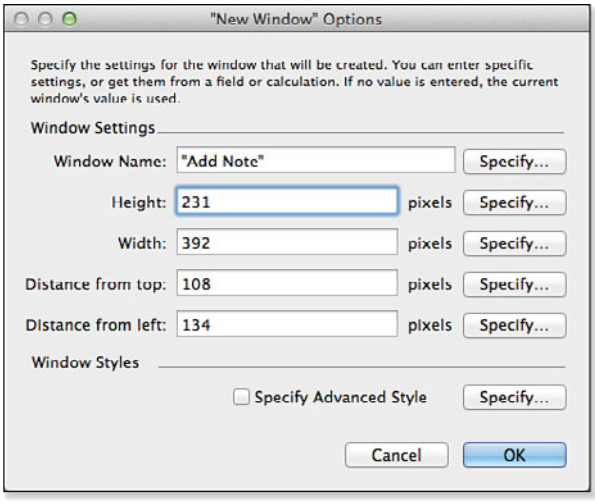
Opening a new window for your navigation or other button elements is only the tip of the iceberg when it comes to working with multiple windows. It is possible in FileMaker Pro to strictly control multiple windows—their positions, sizes, and titles.

The simple nuts and bolts of these features can be found in the `New Window` script step options. With them, you can create new windows, close windows, select (bring to front) a specific window by name, adjust and resize windows, tile and cascade multiple windows at once, and control the availability of the Status toolbar as well. Figure 18.2 shows a simple example of the script options for the `New Window` script step.

caution

Remember that FileMaker Go and Instant Web Publishing do not support multiple windows. If you are designing a solution that will run in those environments, you must address the issues addressed with multiple windows in this section using other techniques.

Figure 18.2
This 231-by-392 window opens with a title of Add Note.



Using Window Styles

Advanced styles, which can be specified at the bottom of the dialog, let you use new interface options starting in FileMaker Pro 12, as you see in Figure 18.3.

As you can see in Figure 18.3, you can specify not only the style of a window but also which controls are available. This provides a degree of interface customization that dramatically increases your flexibility in building interfaces.

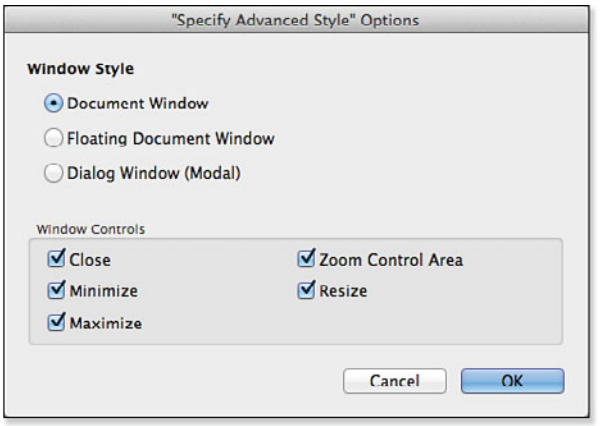


Figure 18.3
Set advanced styles for a new window.

Working with Document Windows

Document windows are traditional windows that people can move and resize, although you will see ways to constrain that behavior later in this section. A floating document window is always on top of other windows even if the users activates another window. A dialog window is *modal* (that is, users cannot activate other windows).

The possible uses for multiple windows are quite varied:

- To view as many layouts at once as your screen real estate allows
- To create multiple List view windows of the same table, with different found sets, at once
- To use a form for editing a single record while still viewing multiple records via either List or Table view
- To create a pop-up window, similar to a dialog
- To keep navigation, function, and other palettes off to the side of your workspace
- To view reports while not having to leave the windows/layouts in which you're working

tip
Remember that with FileMaker Go on iOS devices, you can create multiple windows, but only one window is visible at a time. The control at the left of the toolbar shows you how many windows are open and lets you switch among them.

Creating a Modal Dialog with a Window Style

Modal dialogs—windows that stay open in the foreground while waiting for some action to be performed by the user—are a common user interface standard that users will find familiar. Certainly, the `Show Custom Dialog` script step will take care of some of your basic needs, but in cases where you would like to control the look and feel of a dialog or need more than three simple text-entry fields, you will need to turn to crafting your own window dialogs.

With the window styles, it is easy to manage dialog windows. You simply use the new window script step and choose a dialog window. You can provide a layout with the required information and/or data entry fields. By omitting a Close control, you can prevent users from closing the window, and users will not be able to activate other windows. They are trapped in the modal dialog, which is actually what you want. However, you need to provide some means of escape such as an OK or Cancel button (or both) that at the very least performs the Close Window script step for the current window. Of course, you can perform other tasks with scripts for OK, Cancel, and any other interface elements you provide. It is common to use a single script with a parameter indicating whether the action is Close or OK (possibly by using the name—not the title—of the selected button. That keeps your actions together.

tip

This "trapping" in modal dialogs has been recognized as an unfriendly user interface device, but it is a good way of stopping the action until a user clicks an OK or Cancel button in the model dialog.

Cancel buttons imply that whatever action the user has taken in the modal dialog window can be undone. That can be problematic, especially if you've allowed the user to add and remove records from a portal, so be careful with the use of that term. One technique for managing the undo process is to use global fields for data entry and to populate true fields only when the user clicks Submit. Other techniques involve record-level rollbacks.

note

A *rollback* essentially undoes a transaction in a database, returning it to a previous state.

*To learn more about rollbacks and undo operations, **see** Chapter 11, "Developing for Multiuser Deployment."*

Creating a Modal Dialog Using a Script Pause State

FileMaker solutions that predate FileMaker Pro 12 did not have access to window styles. As a result, they typically use a different technique to create modal windows. You might find this code in solutions that you are maintaining or modifying. In addition, you might find this code written in new solutions either because the developer (who may be yourself) is not familiar with window styles or for one reason or another prefers not to use them.

To build a modal dialog without using window styles, follow these steps:

1. Build a layout intended to act as your dialog. You can add whatever functions and layout objects to it that you want. The layout can be as simple as a single field, or it can be as complex as one that displays a subsummary report in Preview mode.

2. Add a button that will close the window. If it is to be used like a dialog, you will need two buttons: one for Cancel and one for Submit (or whatever terms you prefer). Both will close the window, but one or the other (or both) will have additional functionality. What is important is that only the scripts behind those buttons can close the window.

3. Now place a button on your main layout to open the subsidiary window. This script assumes that the layout and window are both named Add Note, although the window name has a space in it and the layout name does not. Attach it to the following script:

```
Allow User Abort [ Off ]
New Window [ Name: "Add Note"; Height: 231; Width: 392;
➥Top: 108; Left: 134 ; Style: Dialog]
//note that although the script step doesn't report it,
➥the Close button is disabled in Window Controls
Go to Layout [ "AddNote" (Notes) ]
Adjust Window [Resize to Fit]
New Record/Request
Show/Hide Status Area [ Hide, Lock ]
Pause/Resume Script [ Indefinitely ]
```

It's important to disallow user abort; otherwise, users can close your window without performing the action you're attempting to require. It's also a good idea to lock the Status toolbar. (For backward compatibility, the script step still refers to the Status area.) Finally, you have to hold FileMaker in a paused state so that users can't perform any other action while attending to the dialog.

Keep in mind that your users will still be able to run scripts that are visible in the Scripts menu or elsewhere. In solutions that use this technique, developers often opt *not* to set scripts to display in the Scripts menu and to control or change any custom menu sets in use. Alternatively, they write their scripts such that all scripts visible in the Scripts menu take into account this paused state by either refusing to run or ending gracefully so that the user's state in the modal dialog window is not disrupted.

note

Generally, it's a bad idea to leave a script paused—users can get stuck in limbo—but in this case it is exactly the behavior you want. The script ends, leaving the user in a paused state. You need to remember that a paused state is active when performing any additional scripts or when providing other functions in your pop-up window.

4. Now write the Cancel and Submit scripts. For this example, the Cancel script is this:

```
Close Window [ Name: "Add Note" ; Current file]
```

The Submit script is as follows:

```
Commit Records/Requests[]
Close Window [ Name: "Add Note" ; Current file]
```

5. Finally, attach the script from step 3 to a button.

Adding a Pause State

You'll notice we haven't yet dealt with the pause state. If you add a `Pause/Resume` script step to the `Done` script, FileMaker won't know that you want it to resume a currently paused script. The behavior it normally implements is to overlay a new pause state on top of the earlier pause state. This is entirely as it should be because this allows you to build routines with multitiered pause states.

Multitiered Pause State

A *multitiered pause state* can occur when you have one routine running, paused, while another runs and then hits a pause state of its own. For example, you might be running a report that pauses for a user to enter some find criteria. In performing the find subscript, your process might turn up zero records and pause again to have the user respond to some options on what do to about the situation. These multilayered pause routines fold into each other like Russian dolls: Each pause needs its respective resume script step performed before the outer pause state can itself be resumed.

But in the case where you want to resume a previously paused script, the solution to dealing with your pause state lies with the button options attached to each button object. Select your Done button object and either right-click or navigate to the Format menu (in Layout mode) and choose the Button Setup option. Another technique is to simply double-click the button object in Layout mode. Refer to the Current Script options shown in Figure 18.4.

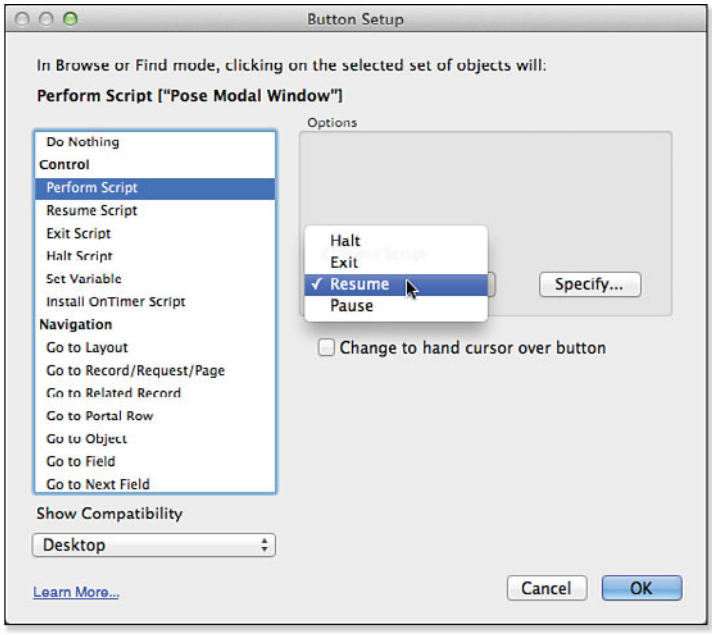


Figure 18.4
Notice the rarely used Current Script pop-up menu in the Button Setup dialog with its choices about what to do with the current script—Halt, Exit, Resume, or Pause.

The Current Script option for the `Perform Script` button behavior is almost never changed. Most often its default state of pausing a currently running script while performing whatever new script is necessary will meet your needs. In this case, however, you need it to resume the current script (which will simply continue from the pause state, effectively ending it) before proceeding through the `Done` script and closing the pop-up window.

This then closes the pause state without creating a nested second one and allows the user back into the state of using the database solution normally. This, combined with the `Close Window` script step, gives the user the experience of clicking Done and seeing the window close. Clicking the Open button sends users back to the layout from which they began.

For another example of working with modal dialogs using script results, ***see*** *"Script Results," p. 451.*

GO TO RELATED RECORD

`Go to Related Record` is one of the most useful and important script steps. In this discussion of scripting, we've focused for the most part on categories of tasks that you can perform with scripts rather than on specific steps, but `Go to Related Record`, which we'll refer to as *GTRR*, merits a discussion entirely its own.

Figure 18.5 shows the Go to Related Record Options dialog. Essentially, GTRR lets you navigate to one or more records related to whatever record or records you're currently viewing. As we discuss in this section, there are several options for how and where that related set will be displayed. It might take awhile for all the nuances of GTRR to sink in, but mastery of this script step is crucial for becoming an experienced script writer.

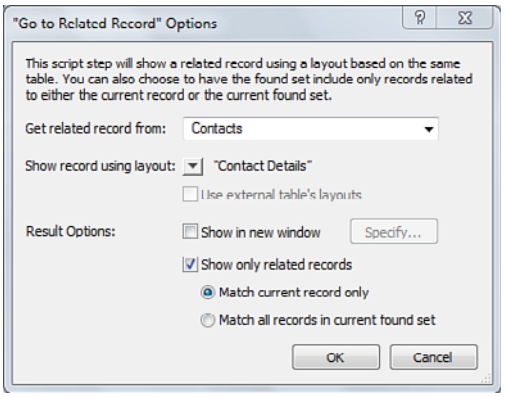


Figure 18.5
`Go to Related Record` is one of the most useful script steps. It's also one of the most complex.

GTRR Basics

It might be helpful to think of GTRR as a way to move or jump from one point on the Relationships Graph to another point. But *from* where and *to* where? In the GTRR options dialog (shown previously in Figure 18.5), the first thing you specify is the destination table occurrence for this move. The script's *context* determines the starting point for the move. We use the terms *origin* and *destination* to refer to these table occurrences.

Whenever a script executes, it does so in a context determined by the active window, the active layout, the active found set, and the active record. All these things can, of course, be changed during the course of a script by using a wide variety of script steps. Whatever layout is active at the point

in the script at which the GTRR occurs determines the origin for a GTRR script step. The active layout situates you at a particular point on the Relationships Graph. So, managing the origin of the jump is done not in the GTRR step itself, but rather through navigation (if necessary) to the appropriate layout beforehand.

As the destination for the GTRR, you can select any table occurrence on the graph, including table occurrences tied to external tables, table occurrences unrelated to the origin, and even the origin itself. This last option produces a special result that's discussed in the "Jumping to Disconnected Table Occurrences" section a little later in this chapter.

The other pop-up list within the GTRR dialog is for specifying a layout to use for displaying whatever set of records the GTRR returns. Unlike the choice of a destination table occurrence, you are restricted in your choice to selecting among layouts tied to the same table (*not* table occurrence) as the destination table occurrence. That's a convoluted way of saying that you're expected to specify an appropriate layout to display the related set of records. We therefore refer to this layout as the *display layout*. If and only if the destination table occurrence is from an external file, you'll have the option to select the Use External Table's Layouts check box. The choices for the display layout consist of those layouts, in the external table, that are tied to the same table as the destination table occurrence.

Another option in the GTRR dialog enables you to specify that the related set of records will appear in a new window. If you select this option, you have access to the same setup parameters that you do when using the `New Window` script step (window name, location, size). If you don't check the Show in New Window option, one of two things happens when the GTRR executes:

- If the display layout is in the current file, that becomes the active layout.
- If the display layout is in a different file, another window must be activated (windows are file specific). If there are no windows for the required file currently open, a new window is created regardless of whether you've checked this option. If there are windows belonging to the external file (even hidden ones), the frontmost of those in the stacking order becomes the active window.

The final option on the GTRR dialog is Show Only Related Records. Your choice here partially determines what found set the display layout contains. It's easier to discuss the possible implications of selecting this option in the course of a specific example, which we do in the example that follows. For now, know that in most cases, you'll want to enable this option.

If you choose the Show Only Related Records option, you also have the choice to navigate to only those records related to the current record or to records related to *any* record in the current found set. For example, if you've isolated a subset of customer records, it is now possible to use GTRR to navigate to a found set of all products ordered by any of those customers. This was possible in previous versions of FileMaker but required a complex workaround.

Predicting the Found Set

The origin and destination table occurrences must be connected on the graph for the GTRR to function. If they aren't, the user sees an error stating, "This operation could not be completed because the target is not part of a related table." The actual error generated is Error 103: Relationship is missing.

Assuming that there is some unique path from the origin to the destination, you really need to know just three rules to determine what found set will appear if you do a `Go to Related Record` script step:

- Every relationship along the path is evaluated.
- The found sets are cumulative.
- The sort setting of the final hop determines the sort order.

If any of the individual hops in a multihop GTRR yield a null set, the entire GTRR behaves the same as a single-hop GTRR that yields a null set. If you're ever in doubt about what records would appear, or in what order, simply create a portal that displays records from the destination table occurrence. The same set of records that shows up in the portal would end up as the found set after a GTRR. Assuming that the portal itself wasn't sorted, the order of the records would even be the same.

Jumping to Disconnected Table Occurrences

There's one final behavior of the `Go to Related Record` step that's worth noting: It can be used to move a found set from one table occurrence of a base table to another. This even works for disconnected table occurrences. In a given window, all the layouts associated with a given table occurrence share the same found set and sort order. This is good because it means that moving back and forth between, say, a list view and a form view based on the same table occurrence doesn't require any found set manipulation.


note

Keep in mind that if two layouts are attached to different table occurrences, their found sets and sort orders are independent of each other, even if they're both occurrences of the same base table.

By using the same table occurrence for both the origin and the destination of a GTRR, you can move the current found set to another layout and/or window. There's something about this behavior that defies intuition, but it's very handy nonetheless.

Dedicated Find Layouts

Entering Find mode and performing find requests is a crucial part of FileMaker Pro, but it's also one of the more difficult things to manage at the user interface level. As your solutions become more complex, Find mode will not be as intuitive for users: They might not have all the fields by which they want to search on one layout, or they might want to perform find requests on related data. Although FileMaker Pro can manage this task quite easily, users might be disoriented or confused by the results.

Say you've created a utility relationship that displays related data based on selected criteria or some temporary condition in the database. The fields sitting on your layout are not a structural


tip

Before launching into dedicated find layouts, remember to explore Quick Find at the right of the Status toolbar. For a great many finds, users can get to what they want just by using Quick Find without specifying the specific field(s) to search. This brute-force finding is what drives search engines, and most people are comfortable with it for basic searching.

one-to-many representation of your primary data architecture. Nonetheless, human users will intuitively want to hop into Find mode and have the process act on the primary relationship rather than your utility relationship.

Here's another example: Imagine looking at an author table with a related book-title field showing the most recent book written by that author. By definition, only one book can be the most current. Now imagine that someone is searching for an author who wrote a given book a long time ago. She is likely to click into the related book-title field in Find mode and be baffled as to why her search returned zero results—or worse yet, she might not realize her mistake and might conclude wrongly that the data doesn't exist (the book she's looking for is not the most recent, so the search fails). Given that the fields on the right relate to only the most current book for an author, the search would be accurate but yield undesirable results. Furthermore, there might be dozens of fields in your database, related and otherwise, but users will want to search on only a small handful of these 90% of the time.

To make the find process as intuitive as possible, you can create a separate find layout. An additional nicety is setting it up to open in a pop-up window. Your users will remain in context—in other words, they'll see where they were in the window behind the current one—and will intuitively understand the process going on. You can build find processes generally in two ways, each of which is covered in the following sections.

Dedicated Find Mode Layouts

The first process is perhaps the simplest. Create a separate layout and populate it with all the appropriate fields specific to the table in which a find is to be performed. Take care to place primary related fields on these layouts: Using the book example again, you would place a book title from a primary-key-to-foreign-key relationship between the Book and the Author tables. The find result would then properly return authors who wrote books—any books, not just the most current—that matched the find criteria.

You can rely on users navigating to these find layouts themselves, along with entering Find mode and performing finds, or you can script the process. The scripted process would involve a button on your standard layouts to take the user to the special Find layout and enter Find mode. A second button on the Find layout itself would perform the request and return the user to the original layout and Browse mode.

This approach is a great way to give your users an intuitive process and shield them from unpredictable results. It's also a nice way to reduce the sheer volume of fields from which they have to choose in Find mode.

Script-Driven Finds

A more complex Find routine replaces the fields in the preceding example with global fields. Providing a dedicated Find layout will likely be something you might want to deliver in Browse mode. Instead of having users work with the related fields themselves (which in Browse mode would display actual data and could potentially pose a problem if users didn't realize they had access to actual data), you can control access and the entire process using a script and offer users empty global fields for entering find criteria.

This approach is labor intensive, and it relies on heavy scripting. As in the example in the preceding section, you have to bring users to the Find layout. This time, leave them in Browse mode. After their find criteria are entered, they have to click a Find button that then takes the system into Find mode, populates and performs the find request by using `Set Field` script steps, and then returns the users to some proper result layout.

caution

The difficulty here lies in replicating all the Find functionalities: inserting omit requests, extending found sets, constraining found sets, and working with multiple requests. We recommend using this technique only in rare cases when you want to fully control the user experience.

Troubleshooting

Pop-up Window Issues on a Windows PC

Pop-up windows don't appear in front of the current window when the current window is maximized.

On the Windows platform, when a window is maximized to fill the application window, no other windows can also be visible on the screen. That is, only a single window can be maximized, and it must be the foreground window. This means that if you try to pop up a window in front of a maximized window, the background window cannot remain maximized. It instead reverts to its reduced state.

If you plan to build a user interface that makes use of multiple windows, be aware of this potential pitfall. It would be better in such cases never to have any windows maximized, even though this means you have to work within a reduced space. Users can still manually maximize a window, so test your routines thoroughly to see what effect this action would have. You'll likely need to add some control routines such as `Adjust Window [Resize to Fit]` to your navigation scripts to get the windows back to the size at which you intend them to be viewed.

Creating New Windows Loses My Found Sets

Whenever I create a new window, all the found sets of the nonvisible layouts are reset to show all records. What causes this behavior?

When a new window is created, either manually from the Window, New Window menu command or via script, it inherits many characteristics of the currently active window. Specifically, it keeps the same size (except when opened via script and specified otherwise), active layout, found set, sort order, and active record. To all appearances, it's as if it's an exact duplicate of the currently active window.

In fact, only the settings of the active layout are retained when a new window is created. All layouts that are not visible (except those tied to the same table occurrence as the active layout) lose any sense of the found set, active record, and sort order. All records are displayed, unsorted, and the first record in the table is the active record.

Incomplete Highlighting Rectangle

My row highlight is showing in its container field, but it doesn't fill the entire portal row well. Where should I first look to address this problem?

If you place an image in a container field and then have a calculation display the contents of that container field in a portal row, even when Maintain Original Proportions is enabled, your rectangle might show whitespace on either side. This is further complicated if you are trying to put something more complex than just a colored rectangle in the highlight field. FileMaker's resizing of images can be unpredictable at times. The best way around this situation in many cases is simply to make the image larger than you need it to be and set the graphic format to Crop.

This page intentionally left blank

19

DEBUGGING AND TROUBLESHOOTING

What Is Troubleshooting?

This chapter introduces you to some of the broader systematic problems that can occur in a FileMaker system. We discuss how to spot these and fix them, and we cover some useful debugging tools that the FileMaker product line offers you.

In addition to *reactive* troubleshooting—the art of finding and fixing problems *after* they happen—we're also going to spend some time talking about *proactive* troubleshooting. To us, this means designing systems that are simply less error-prone and designing them in such a way that any errors that do appear are caught and handled in a systematic way. The better you become at this kind of proactive troubleshooting, the less often and less severely your reactive skills are likely to be tested.

Staying Out of Trouble

The following sections give you a few of the proactive steps you can take to avoid problems.

Understand Software Requirements

Even if you are the user as well as the developer (and, perhaps, particularly in this case), you need to understand what your solution is supposed to do. Draw the lines clearly as to what is in and out of the scope of the project. These lines might shift (as in a phased project), but at any given moment, make certain that you know what your current objective is.

Understanding the requirements also means understanding the data to be used. Avoid sample data; look at real data if you possibly can. People tend to remember extreme cases and might exaggerate their frequency. Likewise, routine errors that are easily corrected, particularly in a manual system, might be ignored. Grill your user (or yourself) with the limits of data: "Can part of an order be returned?" "Will you ever allow someone to register for two classes at the same time—even if the overlap is only five minutes?"

Avoid Unclear Code

If you want to avoid unclear code, two acts in particular are important: giving descriptive names to the components of your program (databases, tables, fields, layouts, and scripts, to name a few) and using comments liberally throughout your program.

Choosing Good Names

As much as possible, the names you choose should be descriptive and follow clear conventions where possible. We offer some suggestions, but you should take them as just that—suggestions. Think of them more as examples on which you could base your own naming conventions. The most important point here is consistency: Try to adopt clear rules for naming things and do your best to stick to them.

Databases and Tables Each database file (a collection of tables) should be named for its overall function. If one file contains all the tables for an invoicing module, call the database Invoicing, not Module A.

If the tables in a single database file do not seem to have anything in common that you can use as a name, you might rethink the file structure.

For tables, we recommend that you name them according to the types of things they store. For an intermediate join table, you should give thought to the function of the table and then decide what thing it represents. Therefore, a join table between Students and Classes could be called StudentClass but is better called Enrollment or StudentsForClasses. A join table between Magazines and Customers is called Subscriptions or MagazinesForCustomers, and so forth.

Some people like their table names to be in the singular form. Therefore, a table of customers is called Customer, a table of pets is called Pet, and so forth. Others (including many Starter Solutions) use the plural form.

Some join tables don't really evoke a natural function, in which case you may need to fall back on a less descriptive name that just incorporates the names of each file: ProjectsForEmployees, for example, or OrdersForPayments. This naming convention easily handles the case in which a variety of relationships is created for tables: ProjectsForEmployees, ProjectsForDepartments, and so forth. If there is a relationship between Employees and Departments, it is quite possible that the Relationships Graph will not allow you to use it to get from Projects to Departments, so you will need the explicit ProjectsForDepartments table. Of course, remember that FileMaker relationships are bidirectional, so ProjectsForDepartments is equally meaningful as DepartmentsForProjects; the name does not imply direction.

Some tables are naturally line item files. The children of other files, which are generally accessed through portals, are characteristic of certain kinds of business documents. Order line items and invoice line items are common examples. You can use a variety of abbreviations as long as you are consistent: OLI (for Order Line Item) or LI for Line Item (as in OrderLI, InvoiceLI, and so forth).

For additional discussion of field naming conventions, **see** *"Field Naming Conventions,"* **p. 84.**

Fields One of the main issues with fields in FileMaker is that they're a superset of what we normally think of as database fields. FileMaker fields include, of course, the classic fields, which are those that store static data, generally entered by users. But they also include fields with global storage, which are not data fields at all, but programming variables; calculation fields, which are in fact small functions, or units of programming logic; and summary fields, which are actually aggregating instructions intended for display in reports.

As a developer, you must decide what things you need to be able to distinguish quickly in this thicket of fields, and devise a suitable naming scheme. Generally, we like to be able to pick out the following database elements quickly:

- User data
- Globally stored fields (often prefixed by g)
- Calculation fields (often prefixed by c)
- Summary fields (often prefixed by s)
- Developer or internally used fields (often prefixed by z or zz)
- Structural database keys

note

If you identify keys (not all programmers do), you can prefix them with kf for foreign keys and kp for primary keys. We sometimes go one step further in naming key fields. For primary keys, we precede the field name with a double underscore (__), and then kp to signify a primary key. For foreign keys, we precede the field name with a single underscore and the designation kf. The effect of this convention is to cause all the key fields to sort to the top of an alphabetized field list in FileMaker, and for the primary key to sort to the very top, above all foreign keys. This makes it easy to access the keys when you are building relationships in the Relationships Graph.

Other developers consider keys in FileMaker as less important than they may be in other database environments and do not bother to identify them at all.

tip

Look at the Starter Solutions for some ideas about naming fields. You might also want to consider what information about a field properly belongs in the field name and what information belongs in the comment that you can create in the Manage Database dialog. Many database designers prefer to put information about key status of a field into the comment section because as database tables are shared among different environments, what is a key for one environment may not be one for another environment.

Making a broad distinction between user fields and developer fields is harder. Those that try to do this generally adopt some kind of overall field name prefix. It's not uncommon to see a scheme where all developer fields are prefixed with two z's. This puts them all together at the end of the field list and the use of the double z means that fields that actually begin with z (ZIP code, for example) are not part of the developer fields group.

Layouts With naming layouts, again, we advocate that you have some clear naming scheme to distinguish between layouts your users interact with directly and those that you build for behind-the-scenes use. One general rule is to prefix the names of all "developer" layouts with Dev_ or a similar tag (you can even use the double-z convention or a variation thereof).

If you follow the Starter Solutions general guidelines, you will find that you have two types of layouts: forms and lists. You can easily name these layouts Client Form and Client List, for example.

Scripts The scripting tools in FileMaker Pro provide major advances in script organization. Grouping scripts together in folders makes sense both for users and developers. You can access these features either from File, Manage, Scripts or Scripts, Manage Scripts.

For additional discussion of scripts, **see** *Chapter 9, "Getting Started with Scripting," and Chapter 16, "Advanced Scripting Techniques."*

Other Elements There are, of course, still other areas where improper names can sow confusion, such as the naming of value lists, extended privileges, and custom functions. Function and parameter naming are especially important, so we'll touch on that area as well.

It pays to take care when naming custom functions, custom function parameters, and also the temporary local variables you create in a `Let` statement. A few simple choices here can greatly add to the clarity of your code or greatly detract from it.

Suppose that you have a custom function intended to compute a sales commission, with a single parameter, intended to represent a salesperson's gross sales for the month. To be fully descriptive, you should call this parameter something like `grossMonthlySales`. That might seem like a lot to type, but if you call this parameter something short and efficient, such as `gms`, you'll be scratching your head over it in a few months' time. The longer name will stay descriptive.

For internal elements (script variables, field names, value lists, and the like), we like to use a style called *camel case*, popular among Java programmers, in which the first letter of the first word is lowercase and all other words in the name begin with uppercase. We don't use this convention for names that the user sees—layout and script names, for example.

For additional discussion of custom functions, **see** *"Custom Functions,"* **p. 427.**

Using Comments Wisely

A comment is a note that you, the programmer, insert into the logic of your program to clarify the intent or meaning of some piece of it. You can use comments many different ways, but we strongly suggest you find ways that work for you and use them.

FileMaker Pro offers a number of useful commenting facilities. You can add comments onto field definitions and inside the body of calculations as well.

To add a comment to a field, just type your note into the Comment box in the field definition dialog. To view comments, you need to toggle the Comments/Options column of the field list; the list can display comments or options, but not both at once.

Comments can be useful for almost any field. They can be used to clarify the business significance of user data fields or to add clarity to the use of global and summary fields.

Also present in FileMaker Pro is the capability to insert comments into the text of calculations and custom functions. We recommend you make use of this feature as well as spaces and indentation to clarify complex calculations.

Finally, FileMaker enables you to add comments to your scripts. Some developers have elaborate script-commenting disciplines. They might create an entire header of comments with space for the names of everyone who has worked on it, the creation date, and even a full modification history.

Other developers use script comments more sparingly, reserving them for places where the flow of the script is less than self-explanatory, or for guiding the reader through the different cases of a complex logic flow. Short, pointed comments throughout a lengthy script can add a great deal to its clarity.

tip

Commenting increases the longevity and reusability of your code, and we recommend you learn about the different commenting options that FileMaker allows.

Writing Modular Code

Modularity is one of those popular buzzwords for which it seems every programmer has a different interpretation. To us, a modular program is one that avoids unnecessary duplication of effort. Much as the concept of database normalization encourages that each piece of information be stored once and only once in a database, you should try to program in such a way that you avoid (as much as possible) writing multiple routines that do the same or similar things. Try instead to write that routine or piece of logic once and then draw on it in many places. Furthermore, separating interactive code from code that does not interact with users increases the reusability of scripts and often simplifies testing.

FileMaker Pro offers several powerful features that can greatly increase the modularity of your code if used with discipline. Three of the most important are custom functions, script parameters, and script results. These topics have been covered thoroughly in their respective chapters, but it's worthwhile to bring them up here again. You should thoroughly understand the mechanics and uses of custom functions and script parameters and use them aggressively to make your code more general and extendable.

note

Bear in mind that custom functions can be created only with FileMaker Pro Advanced, not with the regular FileMaker Pro product. However, after they're created and added to a database, they can be used in FileMaker Pro.

For more information on custom functions, **see** *"Creating Custom Functions,"* **p. 430.**

For more details on script parameters and script results, **see** *"Script Parameters,"* **p. 443,** *and "Script Results,"* **p. 451.**

Planning for Trouble

One of the most important ways to avoid software defects (the graceful term for bugs) is to be aware of all the possible failure points in your system and, most importantly, calculate the consequences of failure. Good programmers do this instinctively. They have a clear sense of what will happen if some element of their program fails. The question is never a surprise to them, and they almost always know the answer.

You can combine the proactive techniques in the preceding sections—particularly modularity—in creating code that is as fail-safe as possible. If each module (usually a script or subscript in FileMaker) does one logical thing, and if it reports the result of its processing via a script result, you can call the script with certainty that it is doing only one set of related processes. When it returns a result that you recognize as good, you can then move on. If the result is not good, you have only one set of steps to reverse.

For example, if you are performing an operation on a record that will result in the deletion of that record or the creation of one or more additional records, you can create a script that takes the record's data as a script parameter, processes it, and then returns a value. At that point, the initial record will still exist, and you can delete it if you want to, but you will never have a case in which the record is deleted before the consequent result is good.

Troubleshooting Scripts and Calculations

There are many specific areas of potential trouble in FileMaker, and we get to those in later sections. Here, though, we want to discuss some general principles for dealing with errors in scripts and calculations.

Handling Errors in Scripts

Many FileMaker actions can result in an error. Error in this context can mean any exceptional condition that has to be reported to the user. This can be something as simple as a search that returns no records or a field that fails to pass validation, or it can be a more esoteric error involving something like a missing key field. In general, in the normal operation of FileMaker, these errors are reported to the user via a dialog of some kind, often with some sort of choice as to how to proceed.

This approach is fine, up to a point. But you, the developer, might not want the user to see this default FileMaker dialog. You might want to present a different message or none at all. Well, if your user performs her searches by dropping into Find mode, filling in some search criteria, and clicking the Find button, there's not much you can do. But if your user is performing a find via a script that you've written, you can intervene in such situations.

For more on custom menus, **see** *"Working with Custom Menus,"* **p. 396.**

note

Using the Custom Menus feature of FileMaker Pro Advanced, you can bridge the gap between applications that rely mostly on the native, menu-driven functionality of FileMaker and those that provide much of their functionality through scripts. Using Custom Menus, you can override selected menu items from the regular FileMaker menu set and attach your own scripted functionality to them. You could, for example, replace the generic Find command in FileMaker's View menu with a menu item called Find Customers and tie that menu item to a specific, customized Find script of your own devising.

Using `Set Error Capture` Script Step

There's a very important script step called `Set Error Capture`. It's worth your while to become familiar with it. This step allows you to tell FileMaker whether to suppress error messages while your script is running. If this step is not present in a script, or if it's present and set to "off," FileMaker reports errors to the user directly. If your script performs a search and no records are found, your users see the usual FileMaker dialog box for that situation. However, if you have error capture set to "on," the user sees no visible response of any kind. After you've set the error capture state (on or off), this setting is carried down through all subscripts as well, unless you explicitly disable it by using `Set Error Capture [Off]` somewhere down in a subscript.

In general, you don't just turn error capture on and walk away. In fact, error capture obliges you to do a lot more work than you normally might. With error capture on, FileMaker error dialogs are suppressed, so it's up to you to check for errors and either handle them or inform the user of those that are important.

In addition to checking for specific conditions (such as a found count of zero), it's also possible to check more generically to determine whether the previous script step produced an error. Typically, you use the `Get ( LastError )` function. This function returns whatever error code was produced by the most recent operation. An error code of 0 means "no error." Otherwise, an error of some kind has occurred. You often check for 0, and if that is not the case, you check for one or more specific errors, and then all others are lumped together. You can use custom dialogs or default behaviors to handle the various errors.

Remember that not all "errors" are actually errors. Although finding no records is a FileMaker error, it might not be an error in the context of your database.

`Get ( LastError )` can be tricky. It reports on the most recent action taken regardless of whether the action was launched directly by a user or by a script. Let's say that you have the following script fragment:

```
Set Error Capture [On]
Perform Find[]
Go To Layout ["Search Results"]
If[ Get(LastError)<>0 ]
    Show Message ["An error has occurred"]
End If
```

This script is not going to do quite what you would hope. If the `Perform Find` script step found no records, at that point the "last error" would be 401 (the code for "no records found"). But after the `Go To Layout` step runs, that error code no longer applies. If that step runs successfully (which it might not if, for example, the particular user didn't have privileges to view that layout), the last error code would now be 0. So, if you want to check for errors, check for them at the exact point of possible failure, not a couple of steps down the road. Alternatively, set one or more local variables to `Get ( LastError )` immediately after the call and then test the local variables when it is logical to do so.

Tracking Down Errors

Suppose that, despite your best efforts at defensive programming, some aspect of your system just doesn't work right. When this happens, of course, you'll want to track down the problem and fix it. There are a couple of verbs you'll want to keep in mind: *reproduce* and *isolate*.

Reproducing Errors

The first step to take with any problem is to render it reproducible. Bugs that occur only occasionally are a programmer's worst nightmare. Often the circumstances are clear and entirely reproducible: "If I hit Cancel in the search script, I end up on some goofy-looking utility layout instead of back at the main search screen." At other times, the problem is more slippery: "Sometimes, when I mark an invoice as closed, the system creates a duplicate of that invoice!"

If the bug is not transparently reproducible, you need to gather as much data on the bug as you can. Who experienced it? What type of computer and what operating system? Has it been experienced by one user or several? Does it appear consistently? Look for hidden patterns. Does it occur more at certain times of day? Only from specific computers? Only for a particular account or privilege set? Only during the last week of the fiscal quarter? And so on.

Reproducing the bug should be your first priority because you can't isolate it until it's reproducible, and isolating it is your best means of fixing it. You might find that you, yourself, are unable to make the bug happen. This might be a sign that you are using the software differently from your users. Your usage pattern might never cause the bug to happen. One way to leap this hurdle is just to sit down with a user and watch him work. You might find that he's using a feature of the software differently than you had intended or expected or that he performs functions in a different order. This might give you the clue you need.

Debugging Calculations

As a general rule, we recommend that you debug complex calculations by breaking them down into smaller pieces and testing subunits of the calculation code. This suggestion contains a strong implication for how you should build complex calculations in the first place: Define and test the smaller pieces of functionality first and then add additional pieces to the calculation. Alternatively, if there's anything at all reusable in the smaller pieces, don't just fold them into a larger calculation, but define them as custom functions instead. You can recombine them if you want (the comments section in the Manage Database Design is important here so that you know what is being done).

The key to the idea of isolation is specifically to isolate the broken part. Pull out the pieces that are known to work. As you test each piece, remove it if it tests out correctly. As you do this, the area that contains the problem grows smaller and smaller.

Troubleshooting in Specific Areas: Performance, Context, Connectivity, and Globals

The individual troubleshooting sections in the chapters of this book cover particular isolated "gotchas" that we wanted to highlight. In the following sections, we want to do two things: Talk generally about broad areas of potential FileMaker trouble and how to diagnose them, and talk about a number of specific areas that don't pop up in the other chapters or at least don't get a comprehensive treatment.

Performance

"The system is slow!" Performance is a critical part of the user experience. What can you do if things seem slow? First, of course, it's important to isolate the problem. Is just one area of the system slow or one particular function? Or does the system generally seem sluggish? In general here, we're assuming that your solution is a multiuser solution hosted on FileMaker Server, but most remarks (except those entirely specific to Server) apply equally to Server and non-Server configurations.

General Slowness

If you're not hosting the files on FileMaker Server, but rather have a peer-to-peer configuration, and things seem slow, you should seriously consider moving to Server. If you're working with FileMaker Server, you also need to make sure that your server settings are configured correctly for your situation. The first thing to look at is what else is happening on the server computer: Ideally, the answer is nothing. If someone is editing video on the same computer on which FileMaker Server runs, you will have serious performance problems.

The computer on which FileMaker Server runs might be your file server computer (this is a common source of confusion, so make certain you know what you mean by server). The absolutely most efficient configuration is a dedicated machine for FileMaker Server, but that is certainly not a requirement. If you are publishing databases on the Web, you can offload web publishing to a separate computer to keep the FileMaker Server machine dedicated solely to supporting the database. Even if you are sharing the hardware with other applications, you'll want to look at FileMaker Server settings such as the percentage of cache hits, the frequency with which the cache is flushed, and the amount of RAM dedicated to the file cache size.

If you're using Server and all the hardware seems reasonable, it's time to look at your network. You should know your network's exact topology. What connections are there, at what speeds? Where are the routers, switches, and hubs?

Today's networks and computers are quite fast, but it is easy to wind up with an old router or computer sitting in the middle of the network slowing down everyone. You should also be familiar with your firewall situation. Does FileMaker traffic need to pass through any firewalls or packet filters? This configuration can slow things down as well.

A last point is that FileMaker Server can be set to encrypt traffic between the client and the server. This encryption is somewhat processor intensive and might impose a performance penalty. If you're experiencing slowness in a client/server environment, and you're using client/server encryption, you might want to disable the encryption and see whether that makes a difference. If so, you might consider investing in faster hardware.

note

In general, using triggers as described in Chapter 17, "Working with FileMaker Triggers," can dramatically improve some performance issues. Triggers let you do only what is necessary when it is necessary.

Slowness in Searching and Sorting

Searching and sorting are among the operations that, in a Server configuration, are handled chiefly by the server. Therefore, it's possible the slowness in searching or sorting is symptomatic of some general networking issue of the type discussed in the preceding section.

It's also possible that there's a problem with the search or sort itself. In terms of performance, the cardinal sin is to execute a search or a sort based on one or more unindexed fields. This, of course, means that there's no index for the field, which in turn condemns FileMaker to examining each and every record in the database—somewhat akin to trying to find a word in a dictionary where the order of the words is random.

As you might recall, some fields in FileMaker can be unindexed merely because the designer chose to leave them that way, perhaps to save space. For certain fields, this setting can be changed, and FileMaker can be permitted to index the field. Other fields, though, such as globals, or any calculation that references a global or a field in another table, cannot be indexed under any circumstances. If your search or sort includes such a field, the operation will never go quickly, and in fact its performance degrades linearly as the database grows in size.

note

You should allow an unindexed search only if you're sure that the set of searchable records is always going to remain fairly small, and there's no other way to achieve the result you need. In general, programming a search or sort on unindexed fields should be considered a design error and should be avoided.

Note, too, that "unindexable-ness" has a certain viral character to it, where calculations are concerned. Suppose that you have calculation A, which references calculation B, which references fields 1, 2, and 3. For reasons of saving space, you decide at some point to eliminate the index on field 3. Immediately, calculations A, B, and C all become unindexed as well, for the simple reason that they now all depend on an unindexed field. Any searches or sorts that use these calculations will now potentially run quite slowly. Be aware of the issue cascading dependencies when working with indexes.

For additional discussion of indexes, **see** *"Storage and Indexing,"* **p. 104.**

Slowness in Executing Calculations

If you have a calculation that seems to execute very slowly, there are a few avenues you can explore. In general, the greater the number of fields and other calculations that your calculation references, the slower it'll be. It's possible to build up quite lengthy chains of dependencies or to have

dependencies with a nonobvious performance impact. Consider the previous example, with calculation C referencing calculation B referencing calculation A. Every time A or B changes, C gets reevaluated as well. Therefore, the calculation contains more work than you might expect. It's very easy to create elaborate chains of such dependencies, so watch out for them. If you find such a chain, see whether there are ways to restructure it, perhaps in a way that allows some of the intermediate data to be stored or set by a script.

Likewise, beware if your calculations reference any custom functions with recursive behavior. A recursive function is like a little looping script. How long it loops for any case all depends on the inputs. If you're referencing these in your calculations, be aware of this fact.

For more information on recursion, **see** *"Creating Custom Functions,"* **p. 430.**

Finally, if your calculation references related fields, it will likely be slower than a calculation that looks only at fields in the same table.

Slowness in Performing Lookups

It is advisable to replace the Lookup option with a different auto-entry option that gives exactly the same results and behavior but is much faster with large recordsets. The preferred method is to use the Calculated Value auto-entry option instead. The calculation should simply make a direct reference to the related field you intend to copy.

For more information on the Lookup auto-entry option, **see** *"Working with Field Options,"* **p. 97.**

Slowness in Scripts

You can do almost anything in a script, so in some sense a slow script could be caused by anything that could cause slowness elsewhere in FileMaker. But there's one additional point we want to make here: You can often speed things up by using some of FileMaker's complex built-in functionality from a script, rather than building things up from simpler script steps.

Use `Replace` Rather than `Loop` Use the `Replace Field Contents` script step to update a set of records. This step lets you specify a field and a value to put into the field. The value can be a hard-coded value or the result of a calculation, possibly quite a complex one. (The latter technique is called a *calculated replace*—an essential tool in a FileMaker developer's toolkit.)

A `Replace Field Contents`, calculated or otherwise, has almost exactly the same effect as a `Loop`/`Set Field` combination and is often much faster. In simple tests that we've performed, `Replace` seems to run about twice as fast as `Loop`.

Use Relationships Rather than Searching The `Go To Related Records` script step is one of FileMaker's most powerful tools. Using this step, you can navigate from a starting point in one table to a related set of records in some other table, via the relationships defined in the Relationships Graph (technically, you are navigating from one table occurrence to another via the Graph). This navigational hop can be much quicker than running a search in the desired table.

Creating Records Under certain circumstances, creating records can be a slow process. Specifically, record creation will be slower the more indexes you have on a FileMaker table. Indexes on a table are updated every time a table record changes, and each index on that table might potentially have to be updated. As a general rule, indexes cause searches to run faster but may cause record creation to be slower.

Connectivity and Related Issues

In many scenarios, FileMaker's behavior might be affected by network and connectivity considerations. Unless you are working alone with a FileMaker database that lives on one single computer and is used only on that computer, you're likely going to find yourself in a situation where FileMaker data is being distributed over a network. This situation offers a number of potential problems.

Inability to Contact the Server

What happens if you're running FileMaker Server and your users can't see your files? There could be any number of reasons for this turn of events, but the following list contains a few of the most common reasons:

- **The server is down—**Verify that the server is running via inspection or a network utility such as ping.
- **The server is up but the FileMaker service is not responding—**Verify that both the FileMaker Server and FileMaker Server Helper processes are running. Without the Helper process, clients cannot connect to the server.
- **The server machine is working and the processes are running correctly, but the files have not been correctly set for network hosting—**Make sure you have granted network access to the files for at least some users.
- **The files have been placed on the server but are not opened for sharing on the server—**Even if the files are on the server, with appropriate network hosting, it's still necessary to instruct FileMaker Server to open the files for sharing. If the files are marked Closed, they are not open for hosted sharing.
- **Recent files cannot be opened—**If the server's IP address has changed, a recent or favorite file might not be in the same place it was before the change. Using the Open Remote command to locate the file will allow you thereafter to use the new Open Recent command that is now established for the file.
- **Users might not have appropriate permissions to see the hosted files—**In the FileMaker Network Settings dialog, it's possible to specify that the file will be visible for network sharing only to users with certain privilege sets. Users with insufficient privileges could in theory have no privileges that would allow them to see any of the hosted files.

Making files available via network access is covered fully in Chapter 27, "FileMaker Server and Server Advanced."

- **Firewall problems exist—**If there's a firewall between your server and any of your users, the firewall needs to pass traffic on port 5003. If this port is blocked, users will probably not even be able to see the server, much less access any files on it. See the list of firewall ports to open in Chapter 27.

Crosstalk

If a user comes to you and says that all of last week's sales data has disappeared, there are a number of possible causes for this effect. It's possible, of course, that last week's sales data really is gone (in which case you'll want to price tickets to Nome). But it's also possible you've been bitten by a case of crosstalk.

Crosstalk occurs when FileMaker opens a file other than the one you expect. The Edit Data Source dialog lets you provide multiple paths for any given data source; if the first file cannot be found, FileMaker tries the second and so on down the list. Review your data sources to verify that the sequence is correct. Often it is when FileMaker Pro is trying an alternate file that it opens one that you forgot you had left in the list.

Context Dependencies

The idea of *context* covers a lot of ground. Speaking generally, it refers to the fact that many actions that occur in FileMaker don't happen in a vacuum. The effect of certain script steps, calculations, or references can vary depending on where you are in the system. *Where you are* means specifically what layout you're on, what window you're in, what mode you're in (Browse, Find, Layout, or Preview), and what record you're on in the current table. Each of these dependencies has its own pitfalls, and each one is discussed in the sections that follow.

Layout Dependencies

Be aware, when writing scripts, that a number of script steps might not function as you intend, depending on what layout is currently active. Most of these steps require certain fields to be present on the current layout. These include `Go To Field`, virtually all the editing functions (`Undo`, `Cut`, `Copy`, `Paste`, `Clear`, `Set Selection`, `Select All`, `Perform Find/Replace`), all the `Insert` steps, `Replace Field Contents`, `Relookup Field Contents`, and `Check Selection`. These are all script steps that act on a field on the current layout. You can run each of them without specifying a field, in which case they run on whatever field is current. They can also be run with a particular field specified. If you specify a field, and for some reason the script is invoked on a layout that doesn't contain the field, the desired action doesn't take place. Even if you don't specify a field, the odds are very strong that you have a specific layout on which you intend that script to be run. In general, these script steps are somewhat fragile, and you should use them with care. If you do use them, you should be sure that your logic guarantees that the correct layout will be current when the script step runs.

Table Context

You're certainly familiar with table context if you've read much of the rest of this book. The topic was introduced in Chapter 6, "Working with Multiple Tables," and it plays an important role in most other chapters as well.

For a full discussion of table context, **see** *"Working with Tables,"* **p. 81.**

FileMaker Pro databases can contain multiple tables. For many actions in FileMaker, then, it's necessary to specify which table is the current one. For new records, to what table does the new record get added? When I check the current found count, for which table am I checking it? And so forth.

Table context introduces a new kind of layout dependency, and one that, in our opinion, dwarfs the old layout dependencies of earlier versions of FileMaker. If you're not aware of table context and don't handle it correctly, your FileMaker solutions might appear to be possessed. They will almost certainly not behave as you expect unless your system is extremely simple.

In quite a number of areas in FileMaker Pro, table context comes into play. A brief recap of each of these is provided here.

Layouts A layout's table context is determined by the table occurrence to which it's tied. Table context governs which records the layout displays. Note that the link is to a table occurrence, not to a base table; this is significant if you'll be working with related fields, or navigating to related record sets (via the `Go to Related Record` script step). In that case, the choice of table occurrence can make a difference in the contents of related fields.

caution

As with other kinds of dependencies, it's important to make sure that the context is correct for an operation before trying to perform that operation. This is a special pitfall for scripts, which can easily change context during script operation via a `Go To Layout` step. If your script steps are context sensitive, make sure to establish the proper context first!

For a discussion of table occurrences and their implications for related fields, **see** *Chapter 6, "Working with Multiple Tables," and Chapter 8, "Getting Started with Calculations."*

Importing Records When you import records into FileMaker, the target table is determined by the current table context, which is, of course, determined by the current active layout. Before importing records, manually or via a script, be sure to go to the appropriate layout to set the context correctly.

Exporting Records Exporting records is also context dependent. Furthermore, if you're exporting related fields and you're exporting from a base table with multiple table occurrences, the choice of table occurrence from which to export might also make a difference. As in the case of importing, make sure you establish context before an export operation.

Calculations Calculations can also be context dependent, in very specific circumstances. If a calculation lives in a base table that appears multiple times in the Relationships Graph (that is, there are multiple occurrences of that table in the Graph), and the calculation references related fields, the table context matters. The Calculation dialog in FileMaker Pro has a menu choice at the top, where you can choose the context from which to evaluate the calculation. If the calculation matches the criteria just mentioned, you should make sure you get the context right. In other cases, you can ignore it.

Value Lists Like calculations, value lists can also access and work with related data, via the Also Display Values From Second Field and/or Include Only Related Values options. Here again, if the value list lives in a base table that appears with multiple occurrences, and it works with related data, the table context will be an issue and you should make sure it's set correctly.

Scripts Every script executes in a particular table context, which is determined by the table occurrence of the current layout in the active window. (FileMaker Pro can have several windows open within the same file, and they might even display the same layout.) A large number of script steps in FileMaker Pro are context dependent. If you fail to set the context correctly or change it inadvertently during a script by switching layouts or windows, you could end up deleting records in the wrong table (to take an extreme case). Interestingly, FileMaker Pro currently doesn't offer a `Set Context` script step. You need to establish your context explicitly by using a `Go To Layout` step to reach a layout with the appropriate context.

Mode Dependencies

A variety of actions in FileMaker depend on the current mode. In other words, things taking place in scripts (which is where these dependencies occur) don't happen in a vacuum; they depend on the current state of the application and the user interface. To take an easy example, some script steps don't work if the application is in Preview mode, including especially the editing steps such as `Cut`, `Copy`, and `Paste`, and others such as `Find/Replace` and `Relookup`. If you have a script that's trying to execute a `Relookup` step, and some other script has left the application in Preview mode, your `Relookup` won't happen.

Most of these mode dependencies are really "Browse mode dependencies," because in general it's Browse mode that's required. But a few other mode-based quirks are also important to remember. A few script steps have different meanings in Find mode than in Browse mode. In Find mode, the `Omit` script step causes the current `Find` request to become an `Omit` request, whereas `New Record` and `Delete Record` create and delete `Find` requests, respectively. These three steps work differently in Browse mode, where they respectively omit a record from the current found set and create or delete a record.

note

The `Copy` command does actually have one meaningful and useful behavior in Preview mode. If no target field is selected, a `Copy` command executed in Preview mode copies the graphic image of the current page to the Clipboard.

If you're using such script steps, the answer's the same here as elsewhere: Explicitly set the context if you're using script steps that depend on it. In this case, you should use an explicit `Enter Browse Mode` script step when using steps that depend on this mode.

Finally, there are also mode dependencies that occur outside the context of scripting. A number of FileMaker's presentation features are dependent on Preview mode. These features include the capability to display data in multiple columns, the capability to show the effects of any sliding options you may have set, and the capability to show summary parts and summary fields.

To find full details on these Preview-dependent layout features, **see** *"Working with Objects on a Layout,"* **p. 144** *and "Using Summarized Reports,"* **p. 295**.

The Record Pointer

In addition to all the other elements of context, there's one other important one. Quite a number of scriptable actions depend on what record you're currently on. You might remember that this is a function of two factors: what layout you're on (which in turn translates to a table occurrence, which in turn translates to a base table) and which window you're in. FileMaker Pro, you might remember, supports multiple windows open in the same layout, each with its own found set.

Within each found set, FileMaker keeps track of something called the *record pointer*—in other words, on which record of the set you actually are. This is indicated both by the record number in the Status toolbar, and possibly by the small black bar that appears to the left of each record in list views.

Some script steps are affected by the record pointer, whereas others affect it. Obvious cases of the former are `Delete Record` and `Set Field`. The record that gets deleted and the field that gets set depend on which record you were on to start with. These kinds of cases are clear and trivial.

Less clear are the steps that affect the record pointer—in other words, that move it. Assume that you have a found set of seven found records and you navigate to number 5 and delete it. Which record do you end up on? Old number 6 or old number 4? Old number 6 is the answer: Deletion advances the record pointer (except, of course, when you delete the last record of a set). The omission of one or more records from the found set is treated like deletion as far as the record pointer is concerned.

What about adding or duplicating a record? Is the additional record created immediately after the current record? Just before it? At the end or beginning of all records? Well, the answer depends on whether the current record set is sorted. If the record set is unsorted, new or duplicate records are added at the end of the found set. (More exactly, the set is then sorted by creation order, so, of course, the newest records fall at the end.) But if the record set is sorted, things are different. New records are created right after the current record. A duplicate is created at its correct point in the sort order, which could be immediately after the current record, or possibly several records further along.

The bottom line is that you have to be aware of which script steps move the record pointer. This is a particular pitfall inside looping scripts that perform these kinds of actions, such as a looping script that deletes some records as it goes. If, on a given pass through the loop, you don't delete a record, you need a `Go to Record/Request/Page [Next]` to advance the record pointer. But if you delete a record on one pass, the pointer advances automatically, and unless you skip the "go to next record" step this time around, you'll end up one record ahead of where you want to be.

Globals

Global fields (which in FileMaker Pro are more exactly called "fields with global storage" because "Global" is no longer really a field type) have long been a powerful feature of FileMaker Pro. But there are a few nonobvious facts about globals that can cause problems and confusion.

Unlike data values that are placed in record fields, the values of global fields are specific to each database user (if the databases are being run in a multiuser configuration). That is, if you have an invoicing system with an Invoice Date field, every logged-in user sees exactly the same invoice date

for invoice record number 1300. By contrast, if you have a globally stored field called gFlag, it's possible that every single user could see a different value for that global field. If a global field gets set to a value of 1300 by one user, that value isn't seen by other users. They each have their own copy of the field, unlike a nonglobal data field.

It's helpful to remember that when a file containing globally stored fields is first opened, all global fields are set to the last values they had when the files were last open in single-user mode. This means that users in a multiuser environment can't save the values of global fields. When a user closes a file, all global fields associated with that file's tables are wiped clean. (In effect, they disappear.) If the same user reopens the file, all the globals will have reverted to the server defaults. This is an important troubleshooting point. If you are relying on global fields to store important session information such as user preferences, be aware that if the user closes the file containing those globals, all those session settings disappear, and reopening the file does not, by itself, bring those stored global values back.

caution

From a troubleshooting perspective, it's important to remember that globals are volatile and session specific. Even more important, in a very large number of cases, globals can be replaced by variables—sometimes global variables with a $$ prefix, but even more often by local variables within a single script with a $ prefix.

File Maintenance and Recovery

A corrupted database system is every developer's nightmare, as well as every user's. Database systems are complex and very sensitive to the integrity of their data structures. Errors in the way data is written to a database can damage a system, or in the worst case, render it unusable. Periodic maintenance can help you avoid file structure problems. In the worst case, if one of your files does become corrupted, FileMaker has tools to help you recover from this situation as well.

It might occasionally happen that a FileMaker file becomes so badly damaged it cannot be opened. When this happens, the reason is usually that the file's host (either the FileMaker client or the FileMaker Server) suffered a crash, leaving the database file partially updated and in an inconsistent state. If a file is damaged in this way, it's necessary to use the File, Recover command. When you choose the Recover command, you first select the file, as shown in Figure 19.1. (The file must not be opened at this point.)

If you click the Select button, FileMaker Pro will attempt to recover the file. Before doing so, you can choose to run a consistency check by clicking Check Consistency. This gives you an overview of the file's status, as shown in Figure 19.2. The consistency check does not modify the file. If you want to see the details of what has been found, you can look at the log file.

tip

The best way to prevent corrupted database files is to prevent them from suddenly being corrupted with a power loss. All computers that host critical information should be equipped with an uninterruptible power supply (UPS) that provides battery backup and a connection (often universal serial bus [USB]) that allows the UPS to determine when the power is off and to shut down the computer in an orderly manner. The UPS needs only to protect your computer and, possibly, Internet connection and network hardware. Printers, scanners, and displays can fend for themselves.

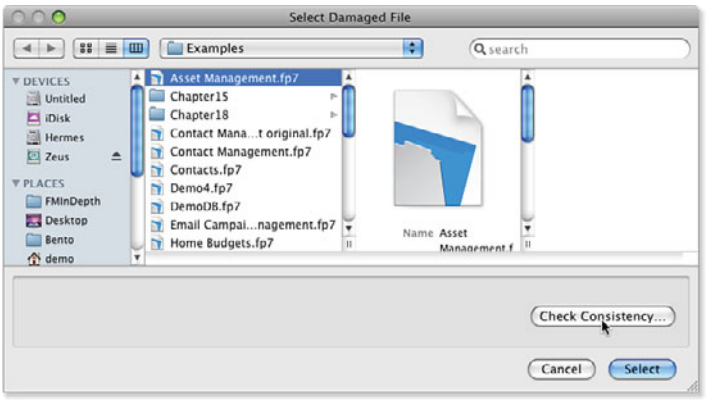


Figure 19.1
Recover a damaged file.

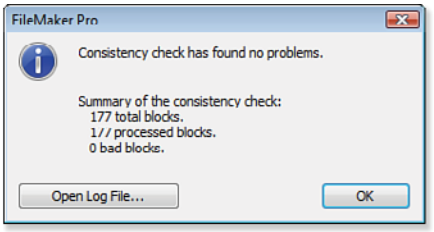


Figure 19.2
Run a consistency check.

If you have chosen to recover the file, you next must specify the name for the recovered file (the original file will not be touched). As you can see in Figure 19.3, in addition to naming the recovered file, you can also choose advanced options.

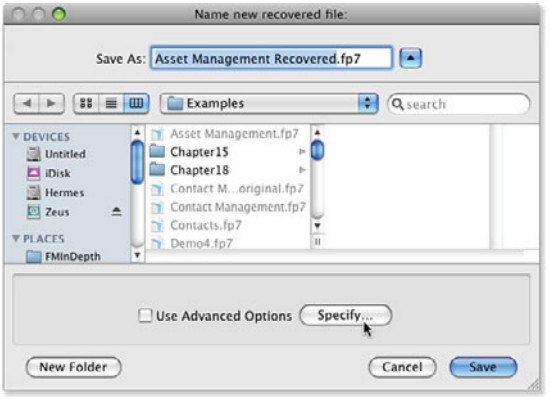
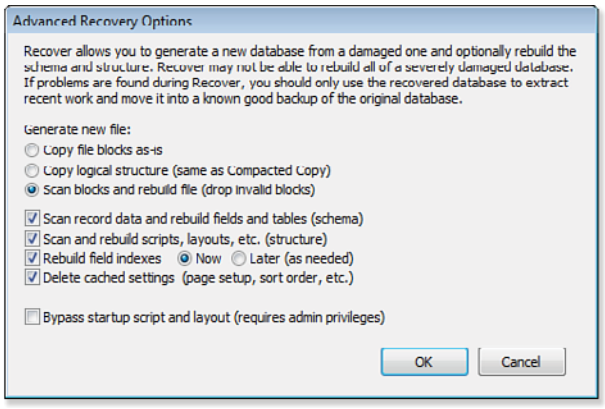


Figure 19.3
Name the recovered file.

If you have chosen Advanced Options, you will see the results shown in Figure 19.4.

Figure 19.4
Use Advanced Options to control the recovery process.



The three radio buttons control the general processing of the recovered file:

- **Copy File Blocks As-Is** simply makes a copy of the file. It is the same as the Save a Copy command using the copy option.
- **Save Logical Structure** is the same as the Save a Copy As command using the compacted copy option.
- **Scan Blocks and Rebuild File** will check each block, copying the good ones and dropping those that are invalid. It is possible that some data may be dropped during this process, but the remaining data should be usable. (This is the default.)

Next, a set of four check boxes lets you choose from a variety of options. All are on by default.

- You can rebuild the schema (the database field and table structure).
- You can rebuild the rest of the database structure (scripts, layouts, and so forth).
- You can rebuild indexes during the recovery process or allow them to be rebuilt later, on an as-needed basis.
- You can drop saved settings for page setup, sort order, and so forth.

The final check box lets you skip a startup script and the startup layout: Recovery ignores those settings and goes ahead to do its best to recover the database. Remember that the goal of recovery is to return the database to a usable state even if some data needs to be dropped. Of course, nothing is better than simply returning to the last good backup copy provided that no new data has been entered. You can use a backup together with recovery. First, restore the backup copy (and make certain to make a copy of the backup if you need to go back to it). Now, run recovery and watch the messages to see if any data has been lost. If no data has been lost, you can rename damaged file, change the name of the recovered file to the name of the damaged file, and go on your way.

Recovery will attempt to warn you if problems were detected, as you can see in Figure 19.5.

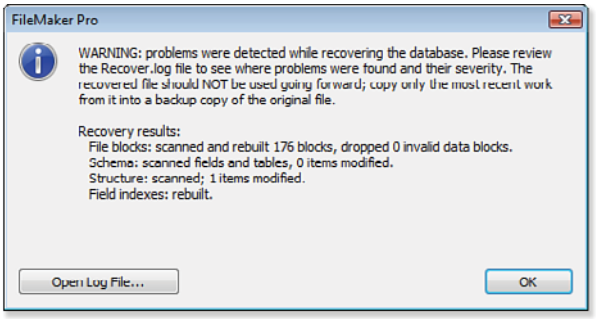


Figure 19.5
Message warning of lost data.

However, if some data has been lost in recovery, you can open the recovered database (without renaming it) and look. It is possible that the data that was entered or modified after the last backup (data that exists only in the recovered file) may be intact. The data that had to be dropped could be in an older portion of the database. In that case, you can import the recovered data to the backup, thereby reconstructing the full database. Even if you have lost some data, you may still be able to recover some of it.

As you proceed, make copies of everything in case you need to revert to the backups or earlier versions. Save copies at every step until you are certain that the crisis has passed.

As you work with a database file, the file can become slowly more fragmented and less efficient over time. Large deletions can leave "holes" in the file's data space. Heavy transaction loads can cause indexes to become fragmented. If your databases are large or heavily used, it's a good idea to perform periodic file maintenance. Use the Recover command with the Copy Logical Structure option to defragment the database and check for problems.

Using the Database Design Report

Beyond documenting your solution within its structure and code, FileMaker Pro Advanced includes the Database Design Report (DDR) feature, which is quite useful and might very well stand as the centerpiece for your system documentation. The report includes an overview of the system, along with detailed information about your database schema, including tables, fields, relationships, layouts, value lists, scripts, accounts, privilege sets, extended privileges, and custom functions. The report can be created as an integrated set of linked HTML documents or as a set of XML files.

Using XSLT, you can transform the XML output of the DDR into a Microsoft Word document that your constituents might find easier to digest and more commonly associated with what they think of as documentation.

Creating a DDR

Creating a Database Design Report is a simple task. But first, you must have FileMaker Pro Advanced, and you must open all the files that you want to include in the report. The files must be opened with an account that has full access privileges. After opening the files, choose Tools, Database Design Report to display the dialog shown in Figure 19.6.

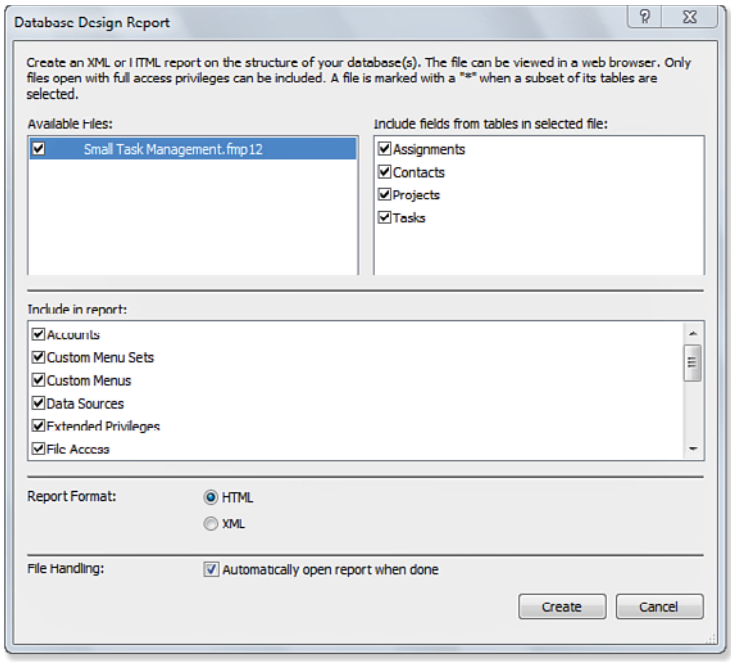


Figure 19.6
FileMaker's Database Design Report can document many aspects of your databases.

By default, all tables in all available files are included in the report. You can uncheck files or tables you do not want to include. You can also specify the types of information to include for each file. Choose either HTML or XML for the report format. Finally, click the Create button and specify the location to which to save the report files.

If you're not sure whether the HTML or XML version of the DDR is more useful to you, think of it this way: The HTML version produces a set of linked web pages that you can open and navigate immediately in a browser. The XML output is more appropriate if you need the data in raw form and plan to manipulate it in some way before viewing or presenting it. One type of manipulation might consist of writing one or more XSLT stylesheets to transform the DDR XML data into a form suitable for importing into a FileMaker database.

The HTML version of the DDR includes a Summary.html document along with various additional HTML documents (<filename>_ReportFrame.html, <filename>_TOCFrame.html, and a Styles.css file). To view the report, open the Summary.html file in any frames-capable web browser.

Each of the solution's files is listed, along with counts of elements within those files (fields, tables, layouts, accounts, and so forth), as shown in Figure 19.7. Click a filename or any of the element counts to view details. All the details for a particular file are included on one (possibly lengthy) page. Use the navigation frame at the left side of the window to quickly move to the section you are interested in. You might also use your browser's Find feature to locate a particular element within the report.

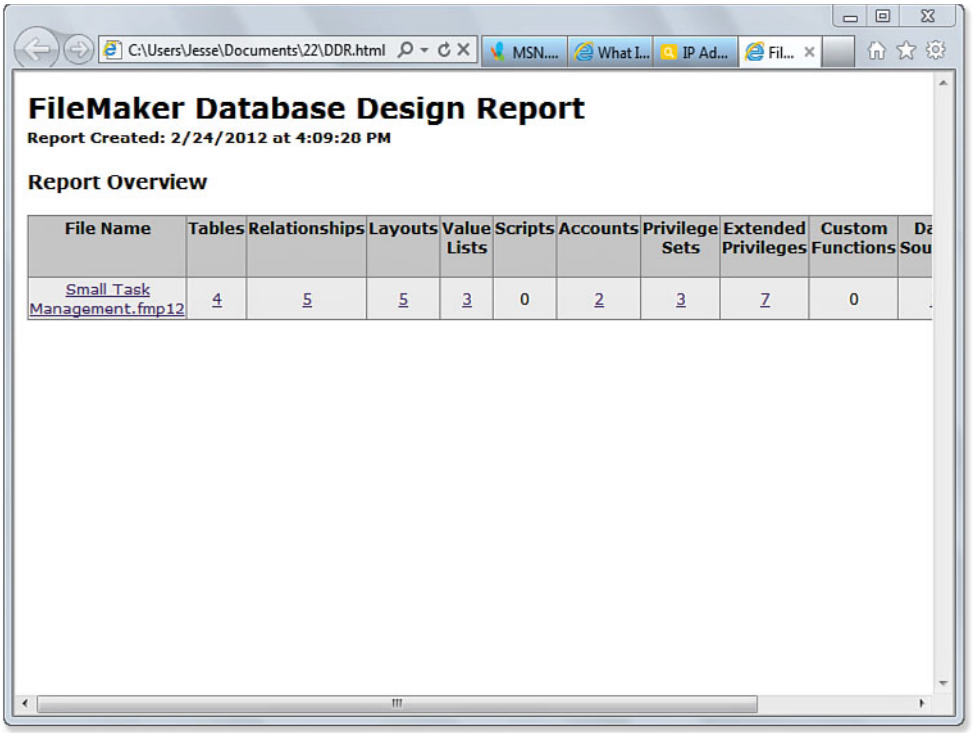


Figure 19.7 The DDR summary shows counts of each database element.

The DDR includes many hyperlinks that make it easy to navigate the report. For instance, the Fields section lists every layout, relationship, script, and value list that uses each field. Each of the listed items is a link that displays the element.

Figure 19.8 shows part of the summary and the beginning of the section of tables in a DDR.

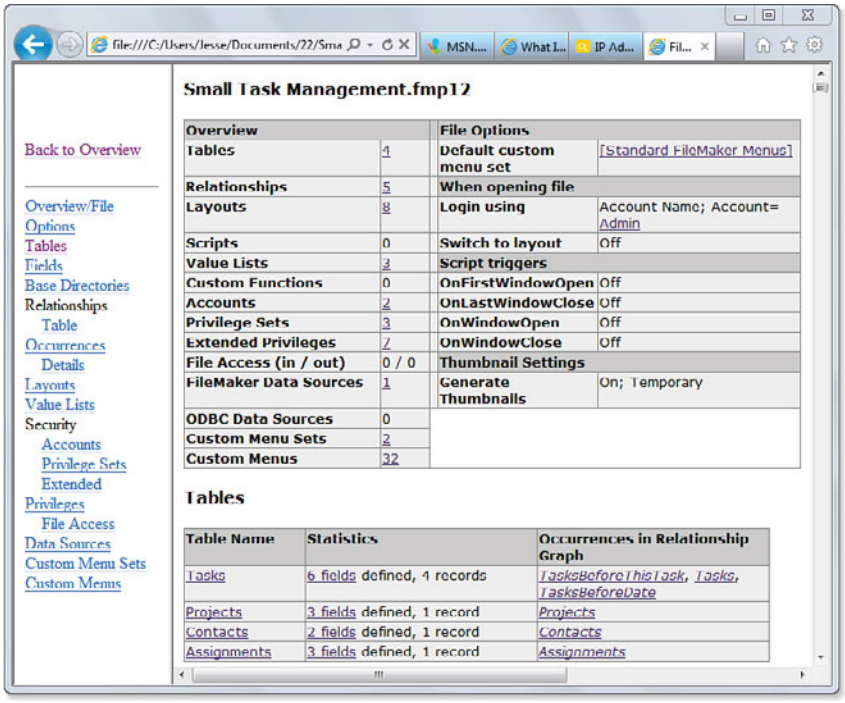


Figure 19.8
Increasing detail is found as you click into the DDR.

Using the Script Debugger

The principle of isolation applies to scripts as well as to calculations. Your problem might lie inside one script, or you might have a complex chain of scripts and subscripts that's exhibiting failure. By far the best tools available for this are the Script Debugger and the Data Viewer, which are part of FileMaker Pro Advanced. The Script Debugger vastly simplifies the process of script debugging, which once upon a time (prior to FileMaker 7) relied chiefly on the insertion of numerous `Pause Script` and `Show Message` script steps! But debugging scripts is still not an automatic process. In the following sections, we walk you through the tools and how to use them.

About the Script Debugger

The Script Debugger and its close companion the Data Viewer are tools that are available only in FileMaker Pro Advanced. This alone is reason enough to invest in Advanced. Trying to troubleshoot a complex script without reasonable debugging tools is a bit like trying to assemble a jigsaw puzzle with your eyes closed. It's not strictly impossible, but it's much harder than it needs to be.

Script debugging can be enabled or disabled from within FileMaker Pro Advanced at any time by choosing Debug Scripts from the Tools menu. This opens the Script Debugger window shown in Figure 19.9.

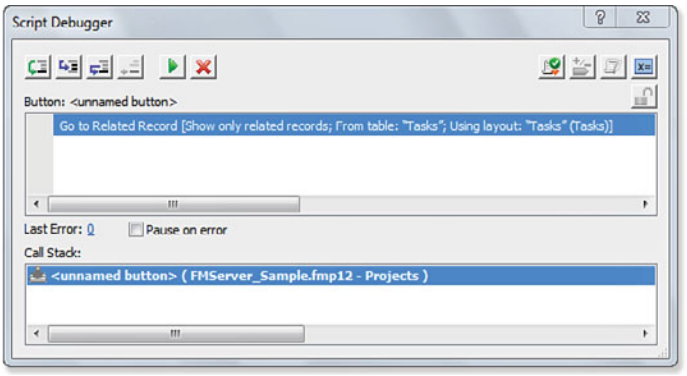


Figure 19.9
The Script Debugger window helps you track down bugs.

Using the Script Debugger, you can step through a script line by line as it executes. You can see when and whether it follows a certain logical path (which branch gets followed when it encounters an `If` statement, for example), when and how it breaks out of a loop, and which subscripts it calls, for example. Using the Data Viewer, you can see how record and calculation data change as the script runs (we'll say more about the Data Viewer later).

In FileMaker Pro, script triggers are supported in the Script Debugger. Not only is the name of the script shown toward the top of the Script Debugger window, but a trigger—if any—is identified at the bottom of the window. This is critically important because with script triggers, stepping through a script line by line does not necessarily make it clear why a script suddenly starts running. Compare Figure 19.9 with Figure 19.10 (a script launched by an `OnLayout Load` trigger), paying particular attention to the bottom of the windows.

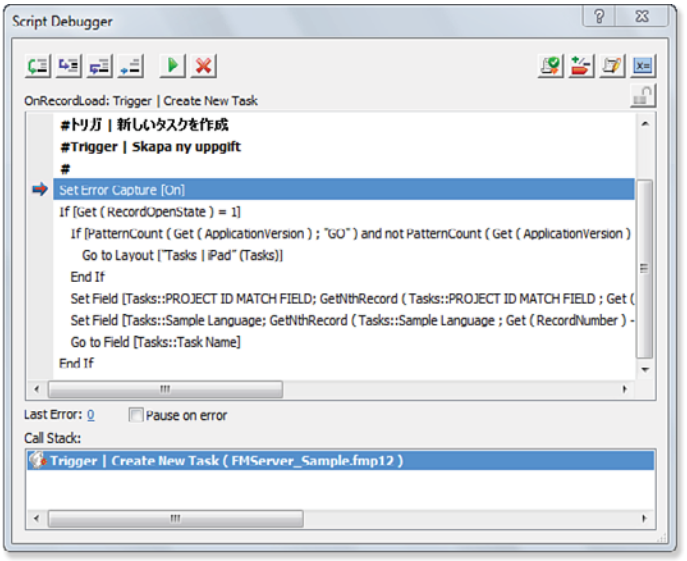


Figure 19.10
The Script Debugger identifies scripts launched by triggers.

The trigger appears in the Call Stack at the bottom of the window. The top window shows the trigger or script that is running now. The second line (if it exists) shows the trigger or script that caused this script to run. On down the call stack you go, tracing backward the scripts and triggers that got you here.

You also see the last error that was encountered, and a checkbox lets you pause the script automatically when an error occurs.

Figures 19.9 and 19.10 show the tools available in the Script Debugger. Most of them have to do with controlling the flow of the script. In general, you'll want to step through the script line by line (using the Step command), but you'll also often want to follow the execution path into subscripts (the Step Into command). Sometimes, when you're inside a subscript, you might want to finish with the subscript and start debugging step-by-step again back in the parent script (the Step Out command).

Finally, the Authenticate/Deauthenticate script allows for immediate overriding of the user's privileges so as to use the Script Debugger. You can also stop the script altogether, open it in ScriptMaker, or use the breakpoint features to allow even more precise control over script execution. We discuss breakpoints in the following section.

At the top right of the window, four buttons let you control debugging. From left to right, they

- Enable/disable triggers
- Set/clear breakpoints (available only if you are stopped at a breakpoint)
- Edit the script
- Show the data viewer

Placing Breakpoints

The Script Debugger enables you to place a breakpoint in a script so that execution stops there and you can see what's happening. In theory, if you have a troublesome script or script chain, you could place a breakpoint at the start and step through the script. But if this is a lengthy script chain, or one that contains a loop that might run many times, this approach may not be very time effective.

Consider a case in which you have a complex set of scripts that call each other; let's say that there are three scripts total. Somewhere in the middle of that script, a date field on the current record is getting wiped out, but you don't know where.

In a case like this, you can use a classic isolation technique called *binary search*. If you have no idea where the problem is happening, place a breakpoint more or less in the middle of everything—say, halfway through script #2. Turn on the Script Debugger, let the script run, and see whether the field has been wiped out by the time you stop at the breakpoint. If the problem has already occurred, move the breakpoint to around the midpoint of the first half of the script chain (that is, 25%) and try again. If it hasn't happened by the 50% mark, move the breakpoint to 75%. Repeat until you narrow the possible range to one or two lines. This may sound

tip

If you have to debug a looping script, it's worthwhile to try to reduce the number of records on which the script runs. In general, if you need to debug the loop itself, one internal breakpoint should suffice at first, either at the beginning or end of the loop.

like it's not much of a timesaver, but using this technique can find the error in a script of more than 1,000 lines using at most ten of these check-and-move operations.

Using the Data Viewer

One of the most important uses of a debugger is to watch certain values and see how they change. These could be database fields, global variables, or aspects of FileMaker state such as the current layout.

The Watch tab lets you add calculations to be evaluated as the script executes. You can use the Add to Watch button at the lower left of the Current tab to add a field or variable to the Watch list. The Current tab, shown in Figure 19.11, shows the values of variables and fields that are accessed in the current script. The Watch tab shows the variables you specifically want to watch. Use the Add to Watch button in the lower left of the Current tab to add the variables and fields you want to watch. You can also add calculations to watch.

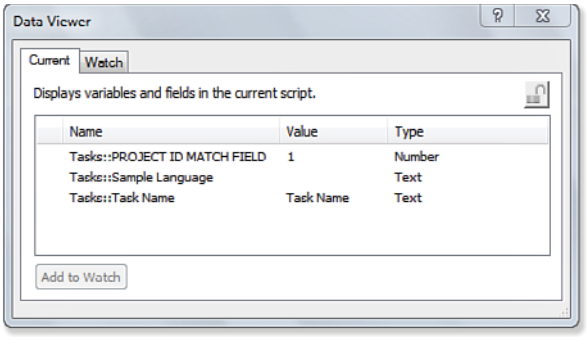


Figure 19.11
FileMaker's Data Viewer, with Current tab selected.

As an example of a typical use of the Data Viewer, consider the example of a script that mysteriously clears out a field. You would like to step through the script line by line and find out when that happens. Your first step is to bring up the Data Viewer and click the Current tab. You can turn on Debug Scripts (if it's not on already) and run your script. Using the various stepping operations, you can move slowly through the script, watching the fields and seeing how they change. In this case, you can pin down the exact step where the field gets cleared.

The Data Viewer is a critical tool in FileMaker troubleshooting, and we heartily recommend you become familiar with it.

20

CONVERTING SYSTEMS FROM PREVIOUS VERSIONS OF FILEMAKER PRO

Updating and Upgrading FileMaker Software

FileMaker—like almost every software developer in the world—issues updates and upgrades to its software periodically. Typically, updates are downloadable, and, with rare exceptions, they're free. On the other hand, upgrades generally are priced.

Updates and upgrades are sometimes issued to correct bugs and tighten security lapses that have been discovered. More often, they implement new features and new ways of doing things. Most of the time this is due to the hard work and imagination of the developers at FileMaker, Inc., but it also is due to requests from users.

note

You can set up FileMaker products to check automatically for updates.

Use the forums at www.filemaker.com/support/contact.html to ask questions and make suggestions. FileMaker monitors them to see where people are having problems and whether something can be done to make life easier for the users. Find FileMaker conversion documentation at www.filemaker.com/r/conver.

Because updates contain fewer changes than upgrades, they generally cause few, if any, problems. However, whenever you change anything at all in your computing environment, you should always back up the relevant files so that if anything does go wrong, you can revert to the previous version. Many people rely on full system backups for this purpose: It can be much faster to just back up everything than to determine what might be affected.

tip

If you do regular backups of your system (and you should), save your pre-update backup in a safe place. In the unlikely event that something does go amiss with one of your databases, it might not be discovered for some time and you may need to revert to it in the future. As part of your backup, make sure you back up the current FileMaker software. Don't rely on very old backups for permanent data storage. Remember that, over time, you may change operating systems and even your computer hardware. For many people, backups that rely on software that is many years old are unusable.

Migrating to New FileMaker File Formats

As you have seen in this book, your FileMaker environment frequently consists of a variety of FileMaker files and, often, a variety of FileMaker software products. You might use FileMaker Pro for your daily work, but you might have a copy of FileMaker Pro Advanced that you use when designing new databases. You might use FileMaker Go on your iPhone or iPad; you might access databases that you have downloaded or sent to the iOS device, or you might access them from shared files that you have hosted on FileMaker Pro.

You also might use FileMaker Pro or FileMaker Go to access files that are shared through FileMaker Server. Furthermore, you might have deployed Custom Web Publishing (CWP) and Instant Web Publishing (IWP) solutions with FileMaker Pro or FileMaker Server. You (or others) may access CWP and IWP solutions via a browser.

note

Unless otherwise noted, in this chapter FileMaker Pro refers to both FileMaker Pro and FileMaker Pro Advanced. Similarly, unless otherwise noted, FileMaker Server refers to both FileMaker Server and FileMaker Server Advanced.

As a general rule, FileMaker maintains interoperability among its products so that you don't have to worry about upgrading or updating all of your FileMaker products at the same time. However, this is not always the case. Occasionally, it is necessary for FileMaker to change the formats of database files in order to implement new functionality or accommodate new features of hardware and environmental software. At such times, you must migrate your FileMaker files to the new format. The change in format is indicated by a change in the extension for FileMaker files.

This doesn't happen very often. Here are the major extensions that have been used:

- **.fmp12**—The FileMaker 12 products (FileMaker Pro 12, FileMaker Server 12, and FileMaker Go 12) introduced this file format in 2012. Files are automatically converted from the prior .fmp format when you open them for the first time.
- **.fp7**—This format was introduced with FileMaker Pro 7 in 2004. Files were automatically converted from previous file formats using the .fp3, .fp5, and .fpj extensions, which date back to FileMaker 3 in 1995.

Because the .fp7 extension has been in use for nearly a decade (at the time this book was written), the focus in this chapter is on migration from that format to the new FileMaker 12 format. Fortunately, the issues involved in converting from .fp7 to .fmp12 are relatively few. This chapter explores the major ones.

Themes in Conversion

The conversion to .fmp12 converts all layouts to the Classic theme. That doesn't matter to you because before FileMaker Pro 12, it was impossible to create themes. Therefore, automatically, all pre–FileMaker 12 layouts effectively use the Classic theme.

Any customization of the layout with colors, fonts, and so forth, are retained as customizations to the converted layout and its Classic theme.

Planning the Conversion

Unless you are working with a single-file solution that is used for noncritical work, you should plan for the conversion. The first step in planning a conversion is to decide what you will do. Your choices range from totally rewriting the solution to doing the smallest amount of work necessary to get everything up and running in the new environment. For an old solution with years and years of edits (which often do not use scripting features such as parameters that make your life easier), this can be an opportunity to start over or at least to clean up the code.

Beyond code cleanups, think about whether it is time to add functionality to your solution. One major area to think about is making your solution accessible to mobile devices and FileMaker Go if it is not already ready for them. You do not have to go all the way to mobile implementation; you can begin to pave the way during your conversion. With even just a few of these steps, you will be making the eventual mobile implementation easier.

Here are some steps you might consider with regard to layouts and layout names:

- Look at the strategy used in the Starter Solutions where you have parallel layouts for iPhone, iPad, and desktop. You can begin to implement that structure in your converted solution by naming layouts appropriately.
- You can plan for an incremental evolution if you have a solution that does not run on mobile devices. In that case, every layout is designed for the desktop.
- Use the structure of layouts in the Starter Solutions by creating a Desktop layout folder and putting the existing layouts in it.
- Rename all the existing layouts with a prefix such as Desktop.
- Any `Go To Layout` script steps will still work because they use internal layout ID numbers rather than the layout names, unless you explicitly calculate the layout name. This will position you to add mobile devices easily in the future.

Whether you can afford the time and effort to start over or to clean up the code depends on the project and the resources available. For critical multiuser systems, you must budget not only your own development time, but also user time for testing.

For a system that has been running for years and years, users might have gotten out of the habit of testing code. In fact, they might actually have never tested solutions that have always worked and have just grown and grown without the benefit of testing.

Building a New Solution

If you think it might be time to rebuild your solution from scratch, take a few moments to consider the details of such a project. Perhaps most important is to think about your existing data. If you can use FileMaker's powerful import and export functionalities to move your data, starting over may not be a daunting task. The one thing to remember is that using the import and export tools doesn't work between FileMaker formats with the FileMaker format, but you can move your data in tab- or comma-delimited format or even XML.

In considering a new solution, browse through this book's detailed table of contents to remind yourself of the features available in FileMaker. You may find new ways of doing things that can improve your solution and reduce the effort needed to rebuild it. Features that are often not used in very old solutions and that can be very useful in rewritten ones include script parameters, script results, triggers, and themes.

One thing that can help the conversion project is to plan right from the start to do the project at least twice. The first conversion will give you an idea of major issues that need to be resolved. In other words, if you can't do a basic task, you have to resolve that issue.

A second conversion may incorporate changes and enhancements subject to the resources available. Testing can continue with this version. Having successfully tested the second conversion, you can then do a real-life conversion, applying any changes you made during testing. At that point you are ready to go live.

note

Many users of FileMaker solutions are relatively sophisticated when it comes to the data and databases they are using. In the world of FileMaker, it is not uncommon to find solutions that are built and maintained and not tested to the extent that major enterprise systems are tested. And, in many cases, this works. The more you know about FileMaker, the solution involved, and—most of all—about your organization, the more you will be able to decide on a strategy.

Preconversion Tasks

You can and should do a number of things before converting older solutions to a new version of FileMaker. Your preconversion tasks vary somewhat from solution to solution, but some categories of tasks are still common to most solutions.

Our comments here are aimed at people who are converting older relational (multifile) systems of some complexity. The purpose of doing any preconversion work at all is to make the post-conversion work go more smoothly; for single-file and simple relational solutions, you might not need to have rigorous conversion plans like this in place.

Document Your Solution

The more familiar you are with a solution, the better your conversion will go. Even if you're the sole creator of a system, having up-to-date documentation comes in handy during the conversion process. We recommend having at least the following items:

- **An ER diagram**—If you've never taken the time to formally create an ER diagram of your system, now's the time. For a refresher on creating ER diagrams, see Chapter 5, "Relational Database Design." If you are converting from a pre–FileMaker 7 database, your ER diagram can be done roughly on a piece of paper. When you are in the modern FileMaker world, you can use the Relationships Graph with its documentation and design tools to produce a more complete ER diagram.
- **Printouts of field definitions, scripts, and layouts**—You might balk at the thought of actually printing out and organizing all these documents, and some people might indeed find that creating PDFs rather than printing works well for them. Many subtle changes take place during conversion, and it's very helpful when looking at a script or calculation formula to be able to compare it with the original. One nice thing about hard copies, of course, is that you can annotate them as you go. You might, for instance, check off buttons on screenshots of layouts as you test them, noting whether everything worked as planned or needs post-conversion attention. The printouts become both your testing plan and your post-conversion audit trail.
- **An access privilege matrix**—This is simply an overview of the privilege settings in your current files. Create it in a database, spreadsheet, or text document; the location really doesn't matter.

If you use FileMaker Pro Advanced, you might want to create a Database Design Report (DDR) of your old solution as part of your documentation process. When you are finished, save the converted solution and its own DDR for reference.

Do Some Housekeeping

In addition to file references, you can avoid other potential post-conversion problems if you do a bit of preconversion work. You can actually identify much preconversion work by examining the alpha conversion files. You might, for instance, discover that you have objects with illegal names, which are placed in between curly brackets during conversion. These can be changed in the original system so that by the time you're ready to do your beta conversion, they're no longer an issue. By doing as much work as possible in the pre-beta conversion stage, you reduce the amount of time and work required to get your converted files ready for production.

If there are scripts, layouts, relationships, external data sources, passwords, value lists, or fields that you know are no longer used or needed, try to eliminate them before conversion. If there has been case inconsistency in the entry of passwords in your current system, take the time to standardize them. These efforts will be rewarded by shorter conversion time and having less to test after conversion. Any other housekeeping in the original files can only be beneficial, including organizing scripts, editing object names, and archiving old data.

Converting Files

The actual conversion of files from previous versions of FileMaker is a simple task. In many cases, it consists simply of opening a file in a FileMaker Pro 12 (or later) version of FileMaker, and responding to the dialog shown in Figure 20.1. Everything just works.

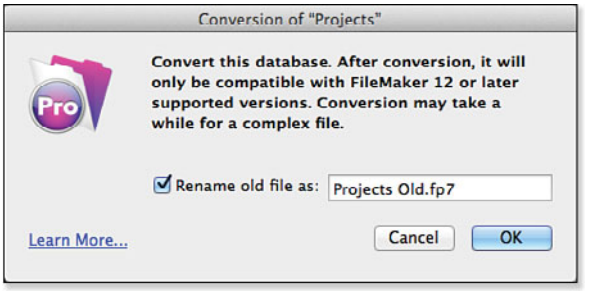


Figure 20.1
Many files can be converted with a single mouse click.

This one-click conversion works with a single-file solution subject to some of the issues described later in this chapter in "Potential Conversion Issues." If you have a multifile solution, you need to convert all of the files to .fmp12.

Figure 20.2 shows the relationships diagram for a two-file solution. Note that the People table title is shown in italics, which indicates that the table is located in an external data source rather than in the same file.

At this point, if you try to use the external table, FileMaker Pro will be searching for the file with an .fmp12 prefix. Until you convert it, FileMaker Pro will be unable to find the file. This means that you should start by converting all of the files in your solution, possibly by dragging them all onto FileMaker Pro 12 and converting them in a single operation.

tip
You might want to save the unconverted (old) files as well as the newly converted files in case you need to go back to either step.

Post-Conversion Tasks

As discussed in the previous sections, you can avoid many potential post-conversion problems by doing some preconversion work on your old system. However, a number of tasks can be done only post-conversion.note

You should begin a list of post-conversion tasks during your exploration of the initial conversion files. You'll spot problems and potential areas of improvement. Anything that can't easily be fixed through preconversion work should go on your post-conversion task list. Keep in mind that you'll end up destroying the alpha files, so don't spend too much time or effort fixing problems. Some fixes are necessary just so that you can continue your exploration; you might opt to do other fixes just so that you can test the results.

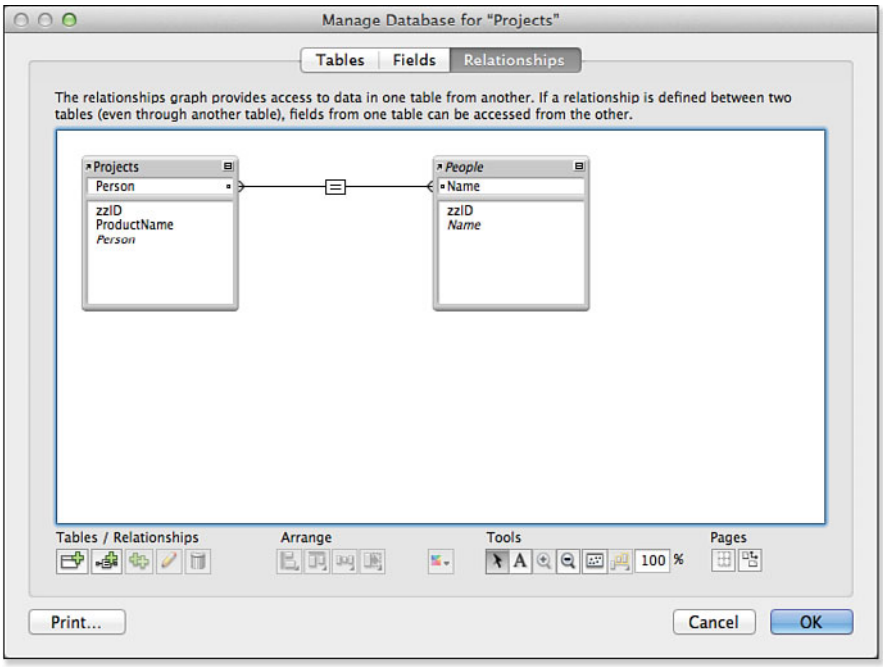


Figure 20.2
Tables in external files have italicized titles.

This page intentionally left blank

21

CONNECTING TO EXTERNAL SQL DATA SOURCES

ODBC Basics

Initially developed by Microsoft in 1991 as Open Database Connectivity interface, ODBC provides a way of accessing ODBC-compliant databases without your knowing anything about the internal workings of the database. ODBC is an API you can call from a variety of programming languages on a variety of operating systems. JDBC is a set of Java classes that allows access to the ODBC database. (Sometimes the two concepts are referred to collectively as *x*DBC.)

SQL

FileMaker's ODBC implementations convert internal FileMaker concepts to SQL both when sending and receiving data. You never see the SQL, but it is there, making the connections work. SQL is sometimes considered an acronym for Structured Query Language, but it is not. Its original name was Structured English Query Language (SEQUEL), but it turned out that SEQUEL was a trademark already in use.

SQL is what is called a *declarative* language: It describes what the data is and what its relationships are. Specific databases process the SQL declarations in their own ways. The other style of programming, *imperative* programming, specifies how a process is to be carried out, not what its final state should be, although that is contained in the imperative instructions.

In FileMaker terms, the Manage Database dialog in which you specify tables, fields, and relationships is declarative, as are layouts and value lists. Scripts are imperative.

FileMaker Architecture

FileMaker files contain a variety of items: scripts, layouts, value lists, accounts, privileges...and data. One of the main benefits of FileMaker is that everything is in one place and works seamlessly together, even if you are working with multiple copies of FileMaker Pro or a shared copy of FileMaker Server.

Database systems such as Oracle and SQL Server typically separate the data from the interface elements. In fact, although these products can have interface elements, programming, and scripting features, other products that access the data in the database directly can replace them (usually by using SQL).

FileMaker can interact with databases such as SQL Server, Oracle, and MySQL; it brings its own nondatabase elements (scripts, layouts, and so forth), and interacts with the external database using ODBC, an industry standard. Many databases use SQL internally, but FileMaker does not.

ODBC Architecture

The ODBC architecture is very simple; understanding it will make it easier for you to use FileMaker's various ODBC features. There are four basic concepts in ODBC:

- *Applications* are programs that need to access ODBC data.
- *Databases* are repositories of data. They can be traditional databases such as Oracle, DB2, FileMaker, and Access; but they can also be other repositories of data, such as Excel spreadsheets.
- *Drivers* interact with databases and driver managers.
- *Driver managers* mediate between drivers and applications.

This structure means that applications and databases can talk to one another without either one knowing the inner workings of the other. Drivers must know about their specific databases, and driver managers must know about their specific applications. The ODBC API forges the critical link between drivers and driver managers, and that link requires nothing specific to either the application or the database. It frequently is a link between computers. The database and its driver run on one computer, while the driver manager and the application run on another.

Driver managers can be distributed or even built into applications. The necessary components to implement driver management functionality are installed as part of standard FileMaker installations. This allows FileMaker to connect to ODBC drivers and through them to other databases. Applications that want to use ODBC to connect to FileMaker as a database provide their own driver managers.

The specific driver required depends on the operating system on which the database is running—that is, the database in ODBC

note

This section describes the standard architecture. In most cases, there are a few notes and exceptions. Sometimes an application supports a subset or superset of ODBC commands; likewise, a database can support a subset or superset of ODBC commands. Drivers can come from the database vendor or from a third party. A company such as FileMaker often tests drivers and recommends specific drivers that it knows will work. This applies both to the application side and the database side.

terms. It could be FileMaker, or it might be Oracle, SQL Server, or MySQL. They, too, can be built into the database code. In the case of FileMaker functioning as a database (an ODBC data source), the necessary code is built into FileMaker Server Advanced for up to 50 connections and into FileMaker Pro for up to five connections on the same computer on which FileMaker Pro is running.

Setting Up FileMaker Databases for ODBC

ODBC can work with FileMaker as a data source or as a consumer. If you are going to be using FileMaker as a data source, there is one essential step to setting up your FileMaker databases to be shared, and there are several optional steps. The essential step is to enable the ODBC/JDBC extended privilege for the database, as shown in Figure 21.1.

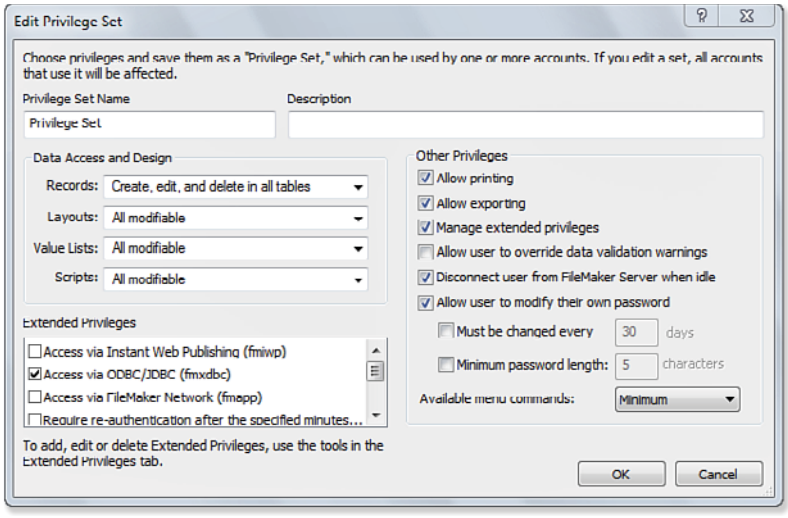


Figure 21.1
Enable the ODBC/JDBC extended privilege.

Included with the electronic documentation for FileMaker Pro and FileMaker Server (along with their Advanced versions) you will find the ODBC and JDBC Guide. It outlines the optional steps.

> *If you have more questions, searching the FileMaker Knowledge Base for "external SQL" will provide the latest updates. In addition, for issues such as this, the TechNet area on Filemaker.com is an excellent resource to search for information and to pose questions.*

note

Because there are some differences in field types and the way in which ODBC functions, as opposed to FileMaker, you might have to make some adjustments in these areas or simply be aware of them. In most cases, you can put this information in the back of your mind and deal with it only if your testing reveals problems. For the vast majority of cases, enabling the extended privilege is sufficient.

Setting Up and Administering ODBC

This section helps you understand what you have to do to set up ODBC without regard to FileMaker—that is, the steps that must be taken before you can start to use the FileMaker ODBC and SQL features described in the rest of this chapter.

ODBC enables you to access a database; it handles the technical matters. You have to handle the practical matters: You need the permission of the database administrator (DBA), and you might need a whole host of sign-offs from various owners of the data involved. Gaining access to data is sometimes difficult, particularly if you are doing something that an organization has never done before or if the data is particularly sensitive.

tip

Neither FileMaker nor ODBC can help you out here except for the general suggestion that by using standards, you are not bypassing security but enhancing it. That is an argument that has worked on occasion. You can phrase it as, "Wouldn't you rather I logged in under your supervision using ODBC than access the data in some other way that you won't know about?"

Installing Drivers

You must set up a driver for each database management system (DBMS) you will access. Often, the driver is set up or installed when the DBMS is installed, and you have nothing further to do. Even if you do have to install it yourself, as you will see here, it is usually a matter of running an installer or dragging a file into a specified location. Because drivers are specific to databases, you normally have to do little configuration: It has all been done for you, which is the point of the driver.

You must set up a data source for each database you will access. Each data source will have its own data source name—a DSN. Often, the DSN will be set up for you by the database administrator and, again, you have nothing further to do.

All this is done outside FileMaker for anyone who wants to access the relevant DBMS and the specific database using ODBC. Because FileMaker itself can be used as an ODBC data source, you will see instructions for setting up drivers and DSN for FileMaker in this section, but you will not be using FileMaker to do so. In this regard, FileMaker is just another DBMS.

Here are the ways in which you can integrate FileMaker with SQL:

note

This section of the chapter might be optional for you. It concerns setting up drivers for databases and setting up ODBC data source names (DSNs). These steps are basically done once, so if you are taking over a project that has already been configured and set up, you can skip these steps and jump right into the database.

note

ODBC client functionality is built in to FileMaker Pro and FileMaker Server.

- **Use another application to access FileMaker data**—In this case, FileMaker is the *data source* and the other application is the *ODBC client*.
- **Use FileMaker to access other SQL data**—In this case, FileMaker is the ODBC client and the other database is the data source.

You need a driver that is compatible with the data source. The ODBC functionality for the ODBC client might be built in, or it might require another driver.

The driver used to access FileMaker data is distributed with FileMaker applications. It may be an optional install. You can also search Filemaker.com for downloadable drivers. Once the driver is installed and configured, you usually do not have to worry about it again.

When you use FileMaker as an ODBC client to access other data sources, the driver(s) are usually provided by the other database vendor.

➥ *Complete instructions are available in the Documentation folder of your installation disc. Look for "FileMaker X ODBC and JDBC Guide," where X is your version number. The filename for the English version is fmX_odbc_jdbc_guide_en.pdf; others are similarly named. You might want to search the FileMaker Knowledge Base just before doing the installation to check whether there are updated versions.*

Check for the latest version numbers on the FileMaker website. These are the certified drivers at the time the book is being written.

➥ *For the latest information on drivers for external SQL data sources,* **see** *the FileMaker website at www.filemaker.com/support/technologies/sql.html.*

➥ *Drivers for OS X are provided by Actual Technologies. You can reach them at www.actualtechnologies.com/filemaker.php*

caution

Before installing new ODBC drivers, check to see whether you have older versions on the computer. If so, uninstall them. On Windows, use Add or Remove Programs from the Start menu; on OS X, you might have to physically remove them. Their most likely locations are the locations into which you will install the new drivers.

tip

If you are using FileMaker Server Advanced, you can install the drivers on that computer so that it has access to the databases. Then clients connect as usual to FileMaker Server Advanced using their own copies of FileMaker. Because the server has ODBC access through its drivers, client users do not need drivers on their computers. This can be a more efficient installation than having each FileMaker user install drivers and connect individually to the SQL databases.

Oracle

- Oracle 9i Release 2 (9.2.0.1) on Windows use Oracle Database Client version 9.2.0.6.5
- Oracle 10g Release 2 (10.2.0.4) on Windows use Oracle Database Client version 10.2.0.3.0
- Oracle 11g Release 1 (11.1.0) on Windows use Oracle Database Client version 11.1.0.6.0
- For all versions of Oracle supported above on Mac OS X, use Actual Technologies, Oracle version 3.0

MS SQL Server

- MS SQL Server 2000 SP4 (8.0.2039) on Windows use Microsoft SQL Server version 2000.85.1132.00
- MS SQL Server 2005 SP3 (9.0.4035) on Windows use Microsoft SQL Native Client 2005.90.4035.00
- MS SQL Server 2008 SP1 (10.00.2531.00) on Windows use Microsoft SQL Native Client 2007.100.2531.00
- For all versions of MS SQL Server supported above on Mac OS X, use Actual Technologies, SQL Server version 3.0

MySQL

- MySQL 5.0 Community Edition (5.0.51b-community-net) and MySQL 5.1 Community Edition (5.1.24-community-nt) on Windows use MySQL Connector/ODBC version 3.51.14
- For all versions of MySQL Community Edition supported above on Mac OS X use Actual Technologies, Open Source Databases version 3.0

Administering ODBC

If you have used ODBC in the past (either with FileMaker or other applications), there might be changes for you to consider. Several years ago, ODBC shipped as a standard part of the Windows operating system, but on the Macintosh it was a third-party product. This section describes the administration process; in the next section you will see how to actually perform the tasks you need to do.

Today, ODBC management is built into both operating systems. On Windows, you set up ODBC using ODBC Data Source Administrator, which is inside Administrative Tools in Control Panel. On recent versions of Windows, Administrative Tools is inside System and Security in Control Panel. The item you are looking for inside Administrative Tools is Data Sources (ODBC), which opens the window shown in Figure 21.2.

On OS X, you use ODBC Administrator in Applications, Utilities, as shown in Figure 21.3. Note that this name is used first in OS X 10.4. If you are using an earlier version of the operating system, it is a good idea to upgrade to use ODBC.

These tools typically run on the computer where the database resides. In fact, you might not have to run them; they might be administered by the DBA as part of the database management operation.

As you can see, both windows have similar tabs.

note

The distinction between a host and client computer is easy to grasp in a large computing environment. With small applications such as FileMaker, it is quite possible for a single computer to be both host and client. If you are using FileMaker Server, that computer is the host for the databases, so you run these applications on the FileMaker Server computer.

Figure 21.2
Use ODBC Data Source Administrator on Windows.

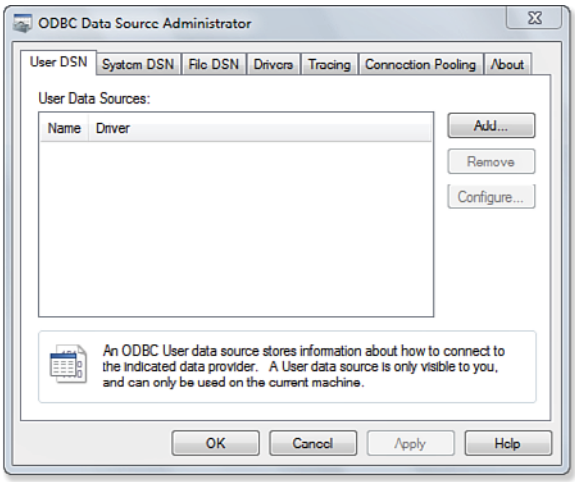
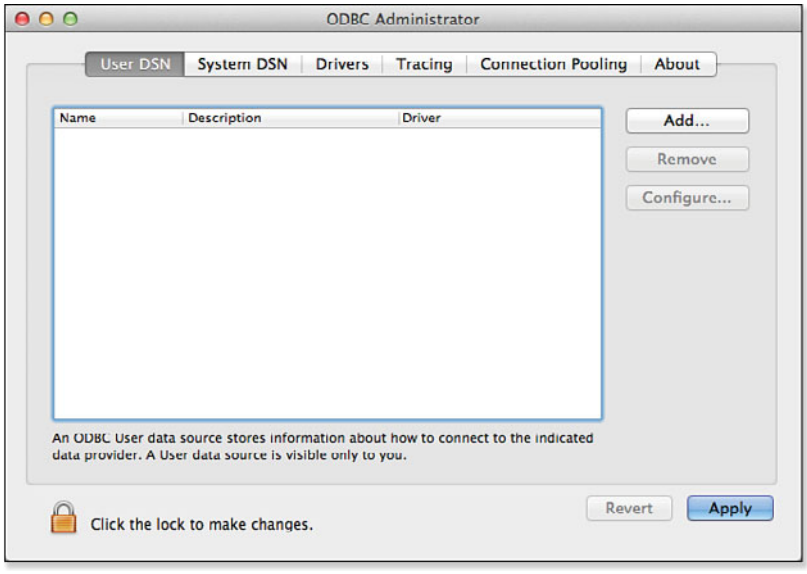


Figure 21.3
Use ODBC Administrator on OS X.



Data Source Names

The first tabs let you manage data source names. These are the objects that link to a driver, which in turn links to a database. Each DSN has the following information that you specify:

- Name
- Description

- Driver to use to connect to the database
- Database
- Login information (user ID and password)
- Other information required by the driver as needed

In short, a DSN has everything you need to connect to a database. There are three types of DSNs:

- A *user DSN* is local to a given user.
- A *system DSN* can be used by all users. This is the only type of DSN supported by FileMaker.
- A *file DSN* (Windows only) stores the information for a system DSN in a file rather than internally in the Registry.

Drivers

This tab lists the available ODBC drivers on the computer. On Windows, a large number of ODBC drivers come installed with the operating system, as you can see in Figure 21.4. Not one comes pre-installed on OS X.

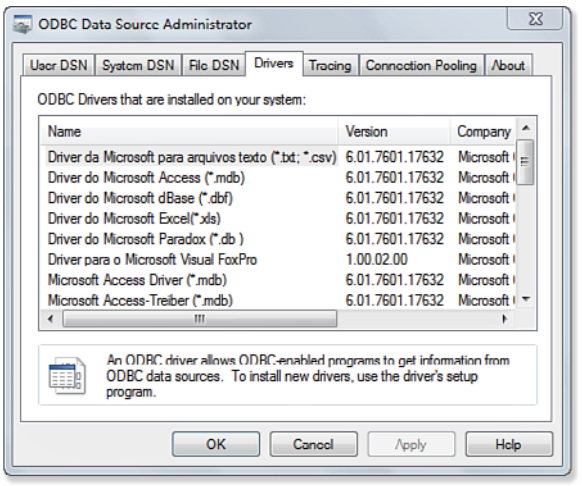


Figure 21.4
ODBC drivers on Windows.

The SQL Server drivers are normally among the installed drivers on Windows. If you want to connect to Oracle or to MySQL on Windows, you have to install the appropriate drivers. On OS X, you have to install any driver you need. The next section shows you how to do this.

Tracing, Connection Pooling, and About

The Tracing, Connection Pooling, and About tabs are used to monitor performance, adjust performance, and view miscellaneous information, respectively. They are normally the province of the DBA rather than the user.

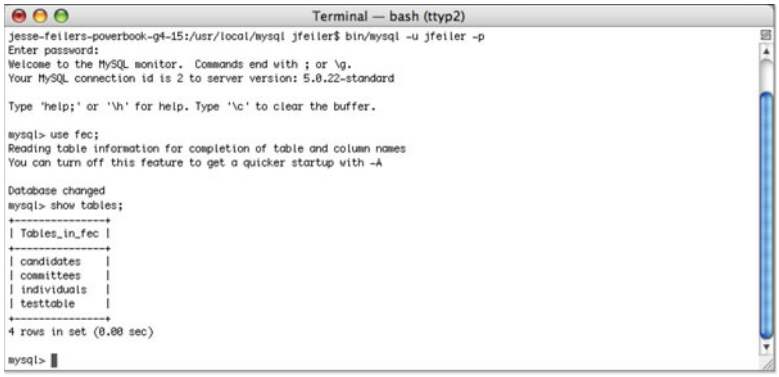
Example: Setting Up a DSN on OS X to Connect to MySQL

This example starts with the assumption that you have downloaded and installed the Actual ODBC Driver for Open Source Databases, as described previously. You must have it to access MySQL via ODBC. After you have done that, you can set up the DSN to allow FileMaker (and others) to connect to a MySQL database.

This example uses a database from the Federal Election Commission (www.fec.gov). For more information on how to download that data and how to create and load the MySQL database, as well as to learn the basics of using MySQL, ***see*** *Jesse Feiler's book* How to Do Everything with Web 2.0 Mashups.

Figure 21.5 shows the MySQL database and its tables.

Figure 21.5
The MySQL database for which the DSN will be built.



You launch ODBC Administrator as described previously. Select the System DSN tab and click Add (you might have to provide your administrator password). You receive a prompt to choose a driver, as shown in Figure 21.6. You want the Actual Open Source Databases driver to connect to MySQL.

When you select the Actual Open Source Databases driver and click OK, the Open Source Database DSN Configuration window, shown in Figure 21.7, opens.

note

If you do not see the Actual Open Source Databases driver, it is not properly installed. Review the previous section, the FileMaker documentation, and the Knowledge Base on the FileMaker website (search for "MySQL" or "ODBC").

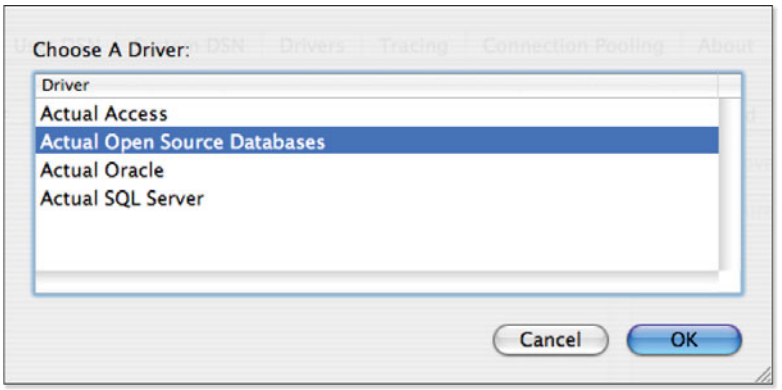


Figure 21.6
Select a driver.

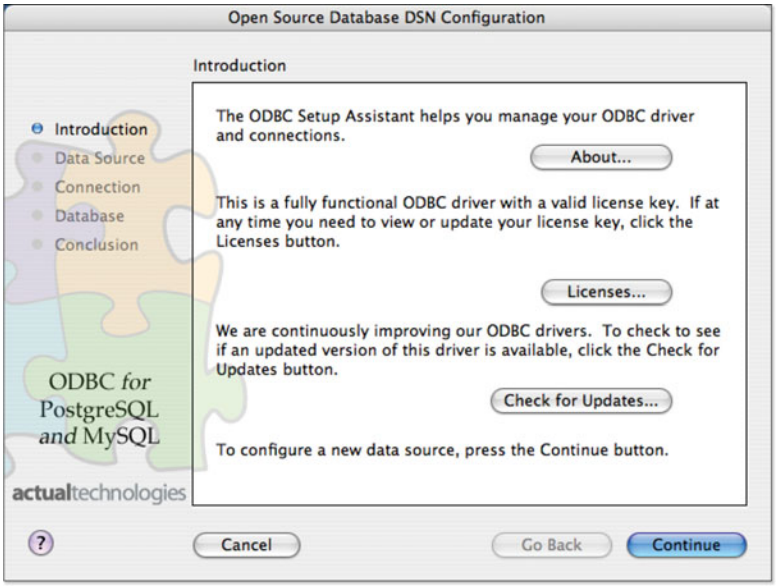


Figure 21.7
Begin configuring the DSN.

note

You can tell from the text and graphics in the lower left of this window that it is the specific driver you want to use. ODBC Administrator takes care of launching the correct interface.

Next, you specify the database you want to use, as shown in Figure 21.8.

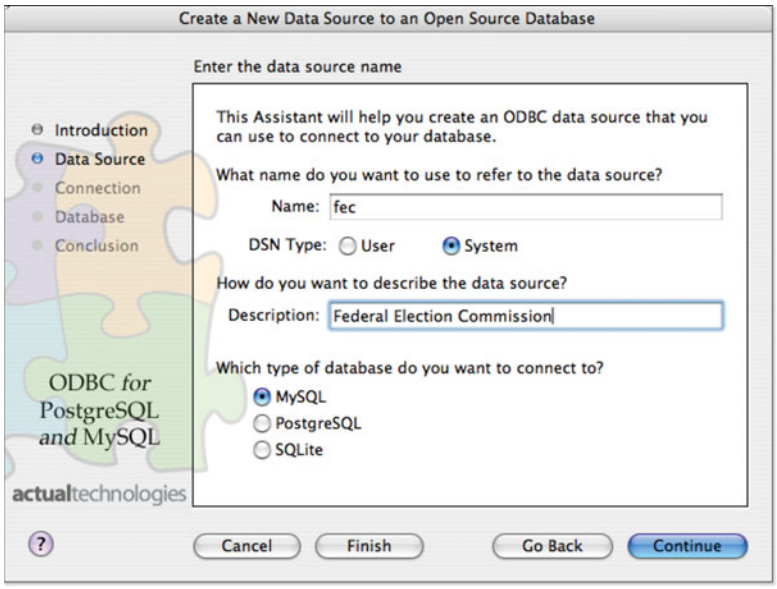


Figure 21.8
Identify the database.

You continue, as shown in Figure 21.9, by specifying the connection and the login information.

Next, for a MySQL database, you get a prompt to specify the socket. As you can see in Figure 21.10, there are two recommended defaults. Try the first one and then the next to see which works for you.

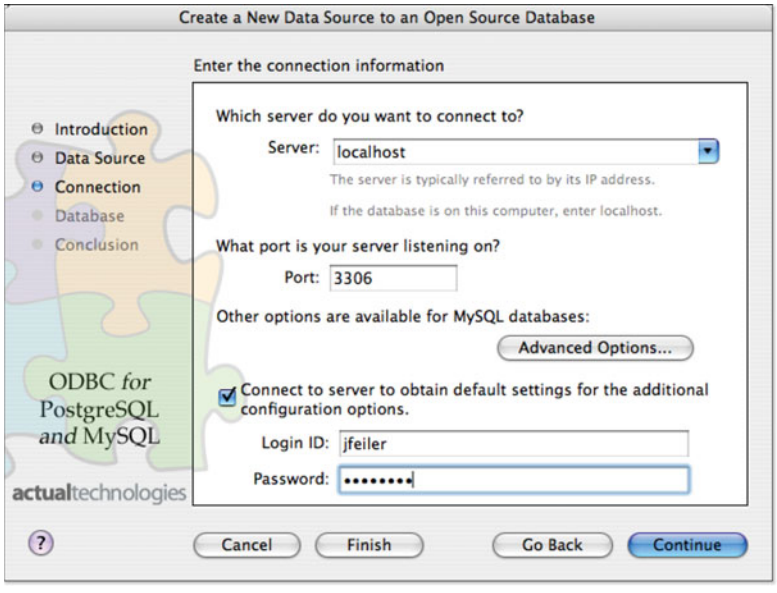


Figure 21.9
Provide connection information.

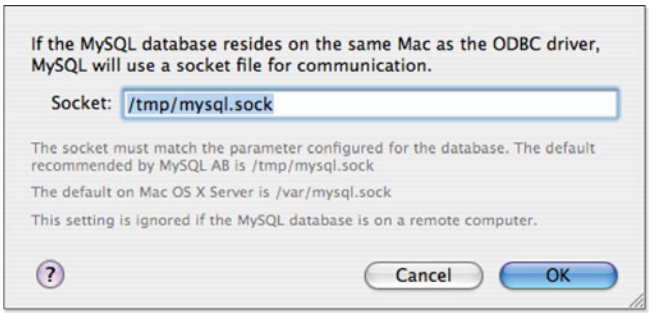


Figure 21.10
Specify the socket.

At this point, the driver attempts to connect to MySQL. If it is successful, you will be able to select the database you want to use from the pop-up menu, as shown in Figure 21.11.

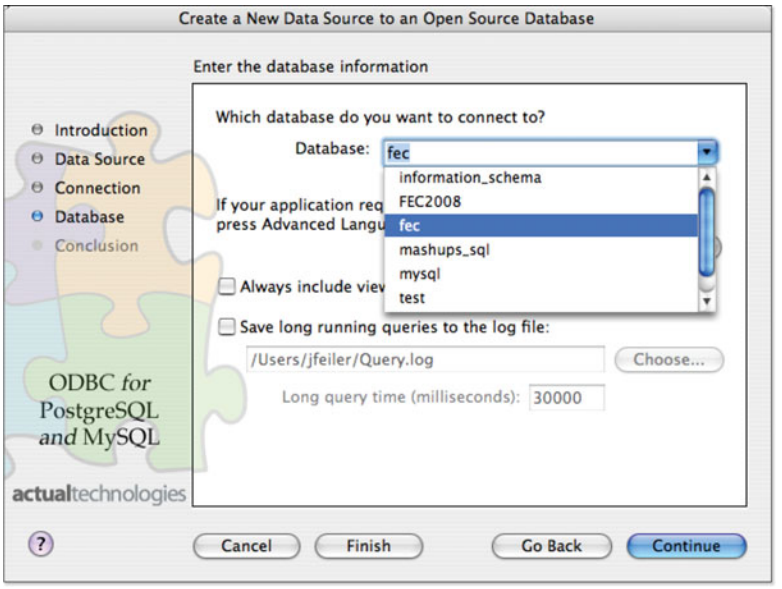


Figure 21.11
Select the database.

Finally, you see a summary of the connection, as shown in Figure 21.12. There is only one step left.

Before exiting the DSN configuration, test the connection with the Test button at the bottom of the window. You should see the results of the test, as shown in Figure 21.13.

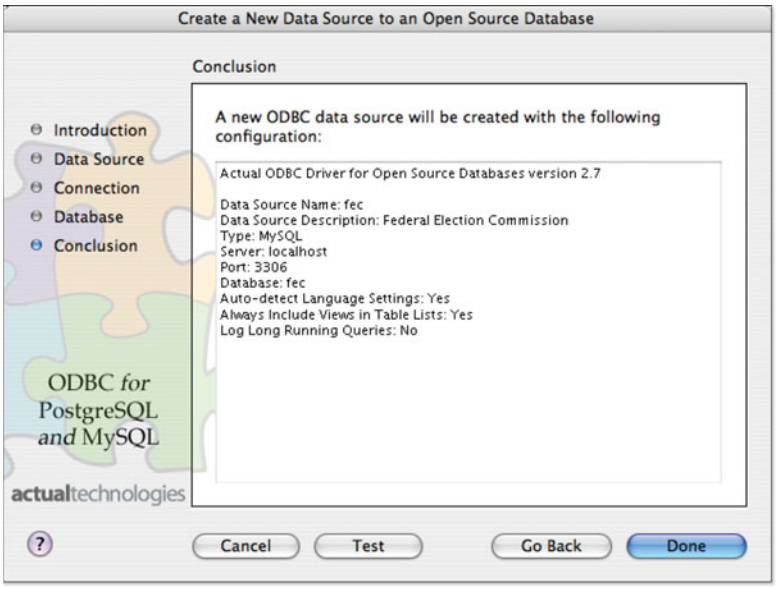


Figure 21.12
The DSN is set up.

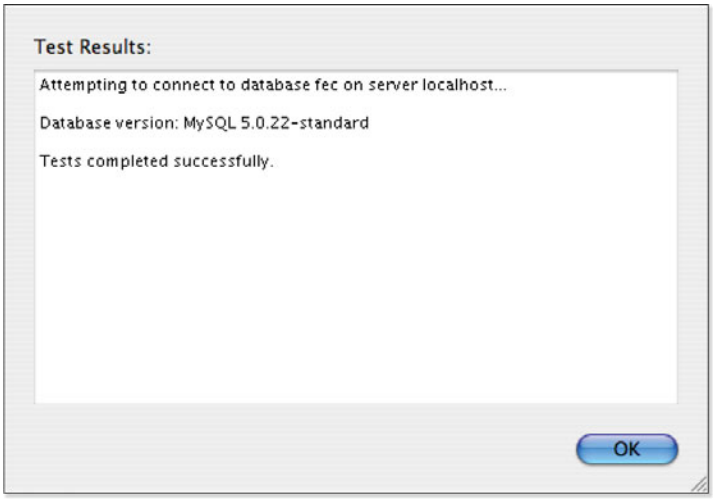


Figure 21.13
Test the connection.

Example: Setting Up a DSN on Windows to Connect to FileMaker

If you want to use FileMaker as a data source, you must configure a DSN for it. This section walks you through the process of doing this on Windows, but the process is the same on OS X.

Open the ODBC Data Source Administrator by choosing Start, Control Panel, Administrative Told, Data Sources (ODBC). It was shown previously in Figure 21.2. Select the System DSN tab and click Add. Select the FileMaker ODBC driver, as shown in Figure 21.14, and click Finish.

The setup dialog shown in Figure 21.14 opens.

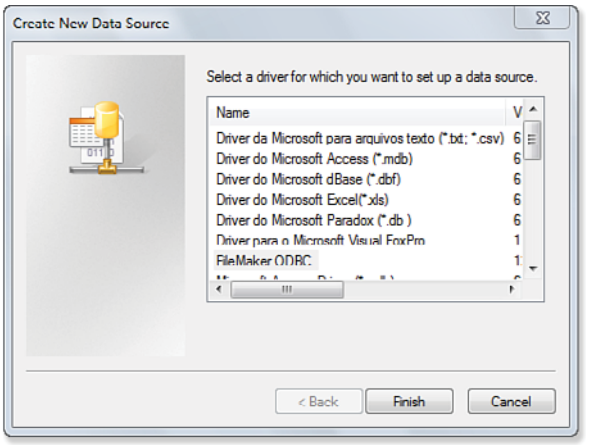


Figure 21.14
Create a new DSN.

tip

You might want to compare this section, including its screenshots, with the previous section. The underlying process is the same, but the details differ because you are using FileMaker here and MySQL in the previous section.

As soon as you click Finish, you enter the same sequence you saw for setting up a MySQL data source. The difference is that this time it is a FileMaker data source. The overview screen shown in Figure 21.15 outlines the process for you.

Name the data source and provide a description, as shown in Figure 21.16.

Figure 21.15
Begin the process of creating the ODBC driver and connections.

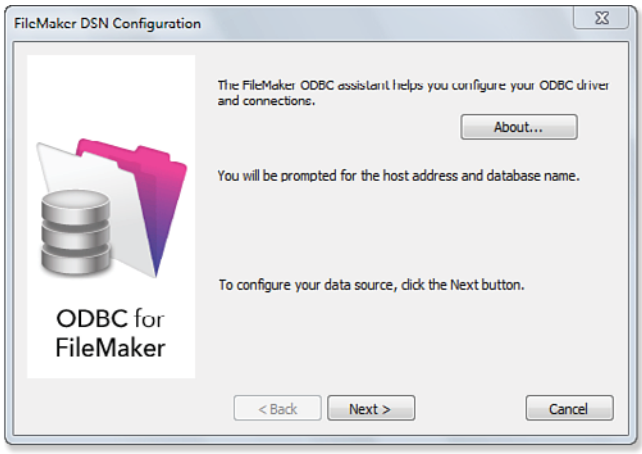
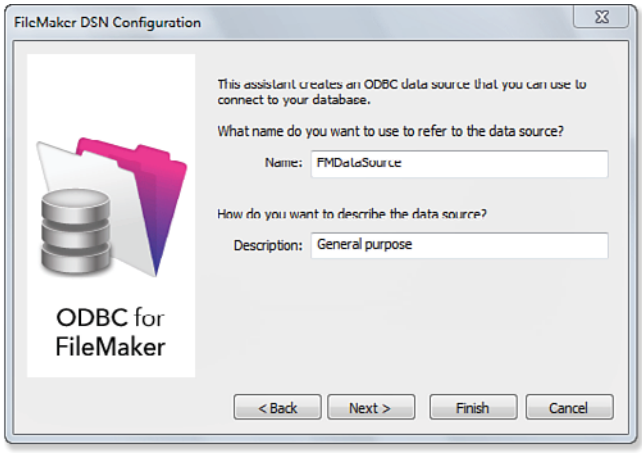


Figure 21.16
Name the data source.



Specify the host address as shown in Figure 21.17. Use `localhost` to connect to your own computer. If you are connecting to FileMaker Pro or FileMaker Server on another computer, use its IP address. It can be a good idea to use the check box to get a list of the sharable databases on the other computer. This tests out the connection. If you don't see the databases you're interested in, check the privileges, as shown previously in Figure 21.1. ODBC sharing is not on by default in new databases.

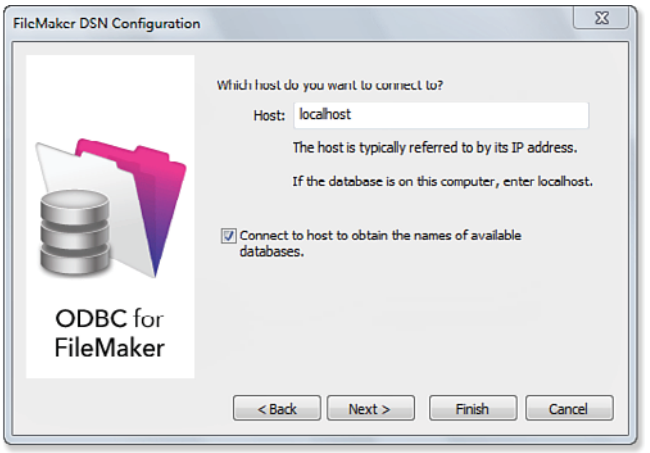


Figure 21.17
Specify the host location.

Choose your database, as shown in Figure 21.18. Note that you can change FileMaker text fields to `long varchar` fields so that when ODBC clients connect to your FileMaker table, they see a SQL-compliant field type. Don't worry: In the FileMaker database, it is still a text field. The driver will take care of the conversion.

tip
You might have to provide a password and ID to actually make the connection—it depends on how the database is configured.

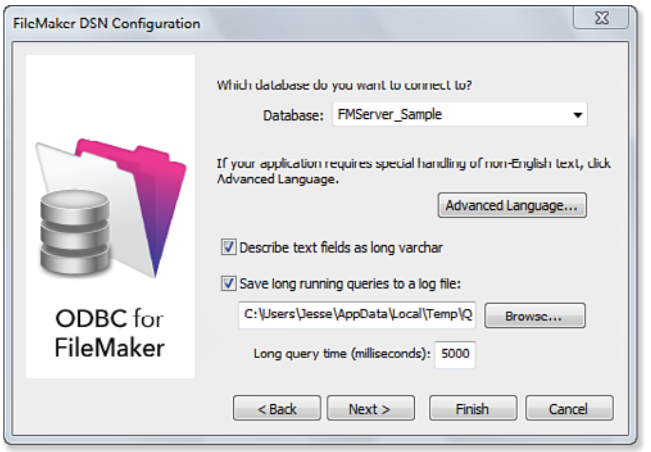
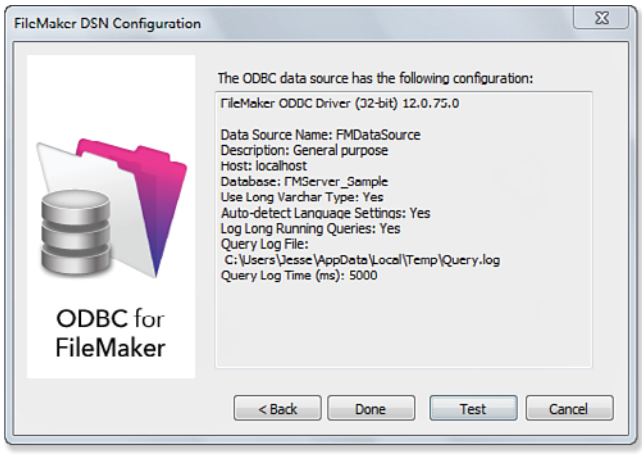


Figure 21.18
Choose the database.

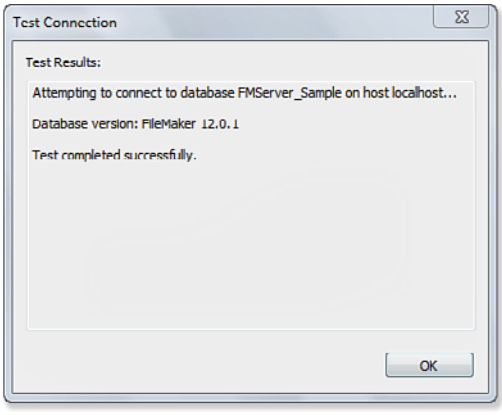
Figure 21.19 shows the next step—a summary of your settings. Check it out and go back if necessary to make changes. It is a good idea to click Test to make certain that it actually works.

Figure 21.19
Review your settings.



If all goes well, you will see the screen shown in Figure 21.20. Your FileMaker database is now ready to be accessed via ODBC from other applications.

Figure 21.20
Test the connection.



Importing ODBC Data into FileMaker

After you have an ODBC DSN set up, you can use it to import data into FileMaker. You use exactly the same process as any other import. Choose File, Import, ODBC Data Source. This opens the window shown in Figure 21.21.

You might be prompted to enter a username and password. Next, you must enter a SQL query to generate the data to be imported. The simplest query retrieves all data from all rows in a table, as shown in Figure 21.22. If you are used to SQL, you can refine your query to retrieve only the needed rows and columns.

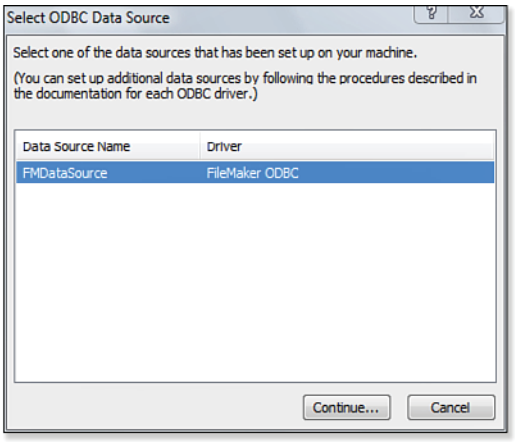


Figure 21.21
Select a data source.

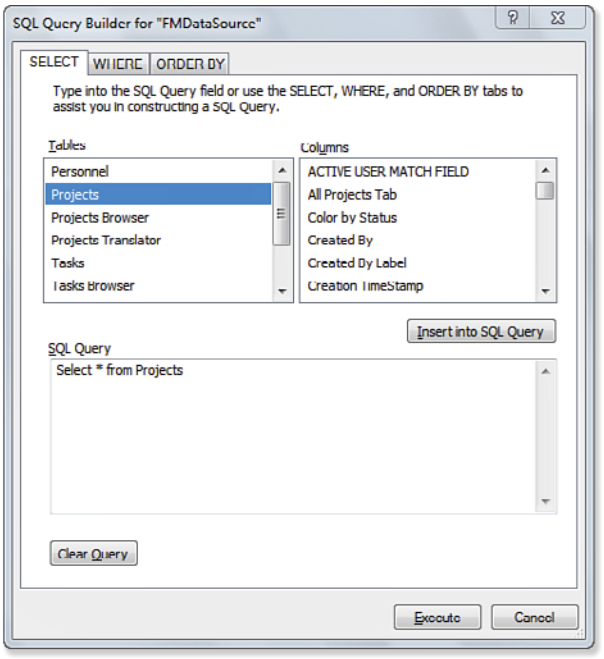


Figure 21.22
Enter a SQL query.

The query runs, and it generates a set of rows and columns. You will see the same Import Field Mapping window that you see with other data imports; simply match the imported data fields to the FileMaker fields you want to fill, and the import proceeds.

Using External ODBC Data Sources with the Relationships Graph

After you have set up the relevant DSNs on the computer where the external data source is located, incorporating the data is remarkably easy.

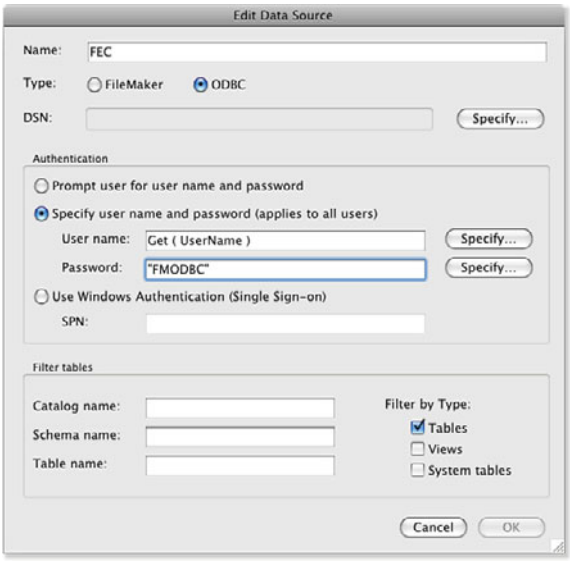
Specifying the Data Source

From the File menu, choose Manage, External Data Sources, just as you would to add any other data source. When you click New in the Manage External Data Sources dialog, you will see the window shown in Figure 21.23 (the window changes its contents depending on whether ODBC or FileMaker is selected). Click ODBC for the type of the data source.

note

This section continues from the DSN that was set up previously for MySQL. It was set up on OS X, but the process here is the same whether it was set up on OS X or on Windows. If you want to review where you are, refer to Figure 21.12, which summarizes the DSN.

Figure 21.23
Select an ODBC external data source.



You can use Windows Authentication (Single Sign-On).

Name the data source. You have to specify the DSN to use. Click Specify next to the DSN field to open the window shown in Figure 21.24. You will see the available DSNs on the host.

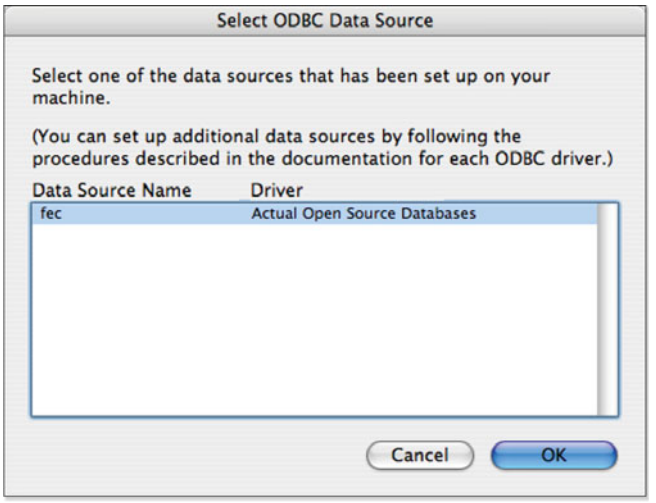


Figure 21.24
Select the DSN.

After you have selected the DSN, provide the authentication information if you want. You can choose to have a prompt every time a user logs in, or you can specify the username and password here.

That's all there is to this process. You will now see your ODBC in the list of external data sources, and you can use it just as you would any other data source.

Adding the External Data Source to the Relationships Graph

To add an external data source to the Relationships Graph, add a table just as you normally would do. In the Specify Table dialog shown in Figure 21.25, you can select the data source and the table within it that you want to use as a base table. Note that there is absolutely no difference in this dialog between using an external FileMaker data source and an ODBC data source. FileMaker has done all the work for you, provided that you have set up the DSN and driver properly. There is a slight difference in the pop-up menu from which you select data sources: They are now organized into local data sources and external data sources grouped together by the name you specified in the Edit Data Source dialog shown previously in Figure 21.23.

The next dialog is specific to ODBC data sources. You will need a unique key in the external table. Many SQL tables do have unique keys (all FileMaker tables have an internal unique key). If your table does not have a unique key, you can construct one by selecting two or more fields that together uniquely identify each record, as shown in Figure 21.26.

tip

This area is one where performance might be affected. If there is no unique key in the external table, see whether you can find a database administrator to add one. If not, try using the method of selecting several fields that, together, form a unique key. It is sometimes the case that you cannot find such a combination. You will know this because when you click OK, FileMaker takes some time to read the external database to see that you have in fact specified a unique key or combination.

Figure 21.25
Select the table to use.

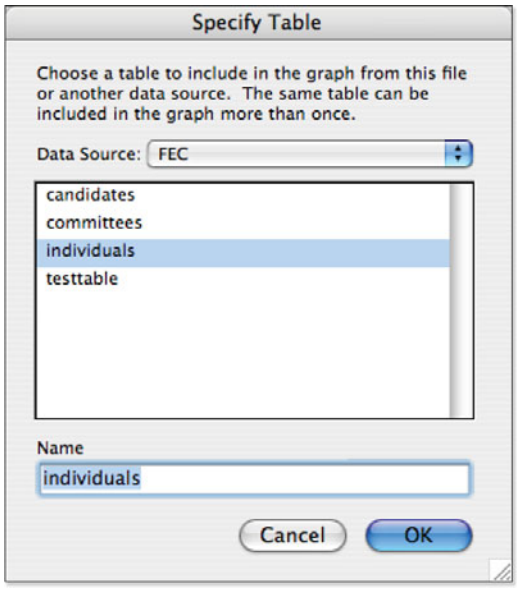
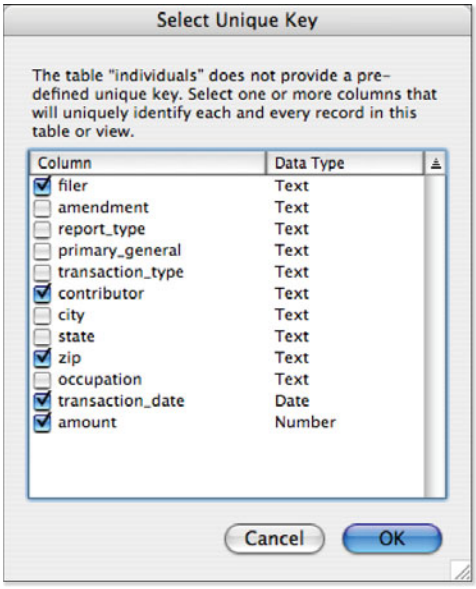


Figure 21.26
Select a unique key.



The external data source is now added to the Relationships Graph, just as any other table would be. Figure 21.27 shows three external data source tables in the Relationships Graph. You can create relations between them just as you normally do. The only distinction is that the external data source tables have their titles in italics. Thus, in Figure 21.27, the demo table is part of the current FileMaker database file; committees and candidates are external ODBC data sources.

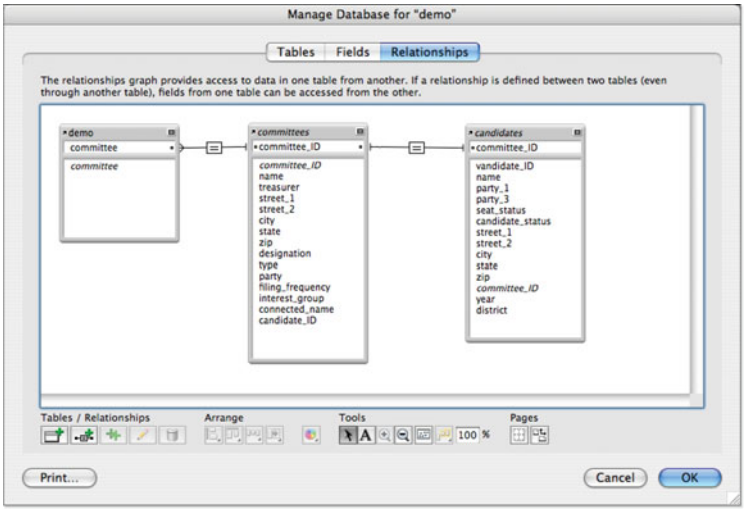


Figure 21.27
The external SQL data sources are now part of the Relationships Graph.

If you go into Layout mode, you can add fields from the external data sources to layouts. Figure 21.28 shows a simple layout in Browse mode that displays data from the external data sources.

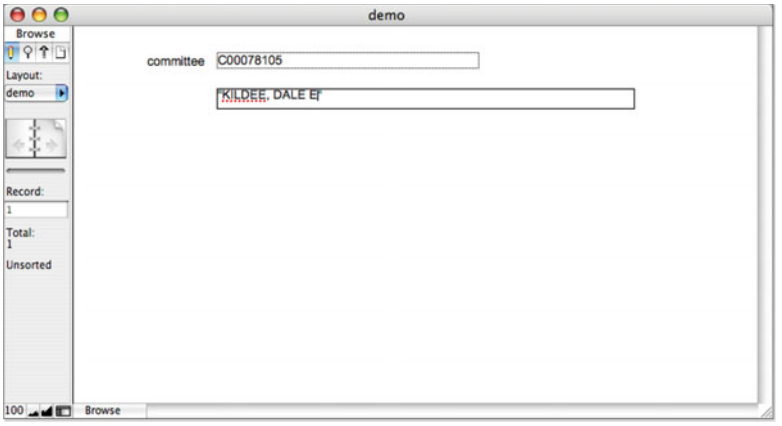
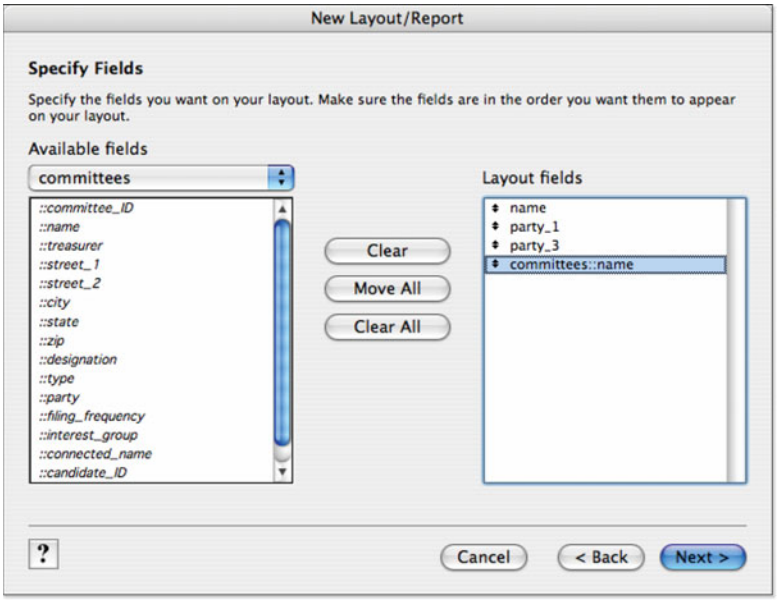


Figure 21.28
You can use fields from external data sources just as you would local FileMaker tables.

In fact, everything you do with FileMaker tables you can do with external data sources. There is very little distinction, although, as you see in Figure 21.29, field names from external data sources are italicized in the New Layout Report dialog.

Figure 21.29
You can use fields from external data sources in new layouts.



Beginning with FileMaker Pro 10, fields from external data sources are like standard FileMaker fields in yet another way: Their values can automatically populate a value list.

Using Supplemental Fields

Just as exciting as adding external data sources to the Relationships Graph is the ability to add *shadow fields* to those tables. Shadow fields appear in the Relationships Graph as if they were part of the external data source, but they are stored in the FileMaker database and merged as necessary. Figure 21.30 shows a calculation field added to the candidates tables.

Two points are relevant here. First, shadow fields frequently are calculation fields that modify the external data in ways that make it easier for FileMaker to use. Here, the cCommitteeCount field is the count of the number of committees for a candidate.

You also might notice the misspelled name of the first field, vandidate_ID. When you are using external data sources, you cannot control field names; they might be in a foreign language, or they might be misspelled. In this case, if you download the data, you will see that from the original data source on, this field name is misspelled. Because it is visible only (!) to programmers, no one has bothered to change it. With the calculation field that has been added, you can create a subsummary part for a layout, as shown in Figure 21.31.

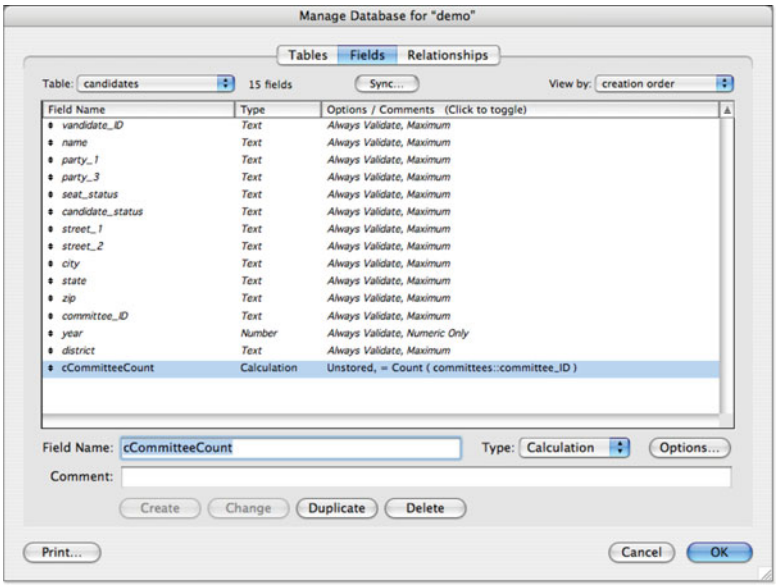


Figure 21.30
Add a shadow field.

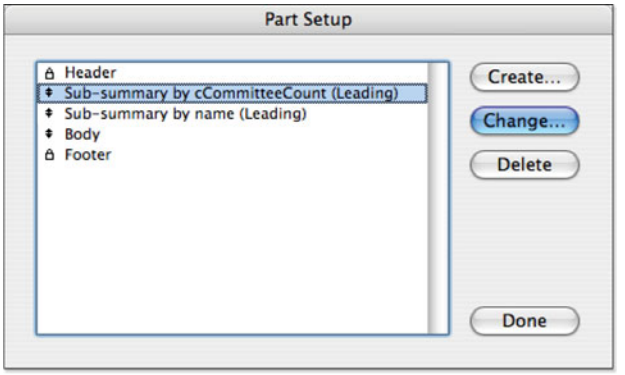


Figure 21.31
Use the shadow calculation field as a sub-summary break.

You can use the field in a sort as well, as shown in Figure 21.32. This sorts the records for candidates first by descending order of the number of committees and then in ascending alphabetical order by name. The first sort field is the shadow field; the second sort field is part of the external data source.

Although external data sources are treated almost exactly as local tables, you will notice that you have the option to sync them with the external data source from the Tables tab of the Manage Databases dialog, as shown in Figure 21.33. For example, if a field in the database is renamed, syncing with the database would update the table. Likewise, if you remove a field from the shadow table of the external data source, it remains in the actual SQL table. You can restore it to the shadow table in Manage Databases by clicking the Sync button.

Figure 21.32
Use both shadow fields and external data source fields in sorts.

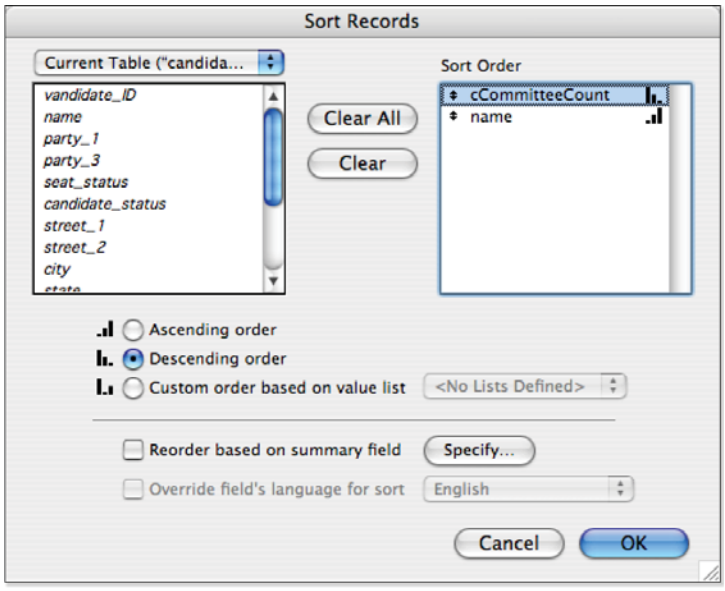
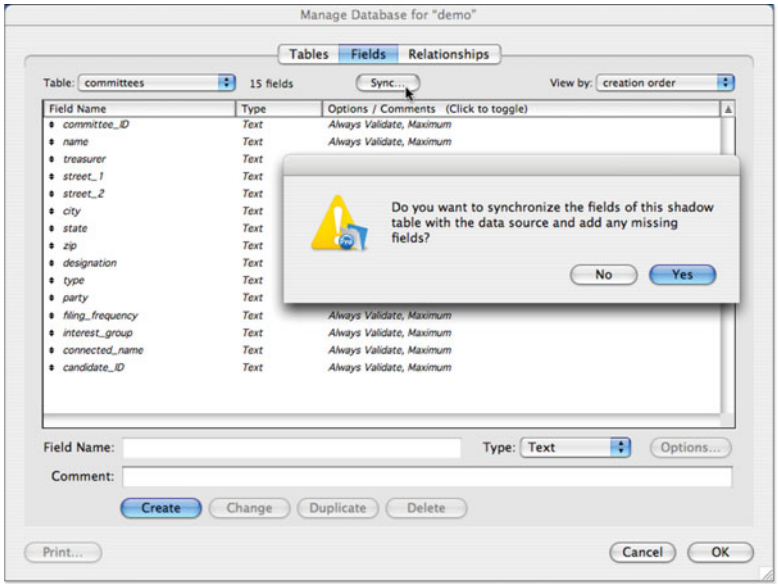


Figure 21.33
You can always sync external data sources with your tables in the Relationships Graph.



Troubleshooting

Translating SQL to FileMaker

The DBA of the SQL database to which I want to connect is asking me a lot of questions that I can't answer and don't understand. What should I do?

Go to the FileMaker website and look at the section for IT managers and technology professionals (http://filemaker.com/articles/guide/it_resources.html). Or, better yet, give the DBA that link. The articles and papers in this area are written in IT-ese.

Apparently FileMaker is executing many individual queries on the database; they show up in the log for the SQL database. Is this normal?

It might be. FileMaker fetches data as it is needed, typically in batches of 25–100 records. FileMaker retrieves external SQL data using primary keys, so the queries are very efficient, but they are more numerous than some other methods of accessing the database in which all the needed records are retrieved in one batch.

How can I track down performance issues on the SQL side?

Some of my queries seem to take a long time to execute in the external data source. More troubling is the fact that some do not, and I can't seem to find the distinction.

Most databases have a query log that keeps track of the queries executed against the database. The DBA should have access to the log. In it, you will be able to see the exact queries that FileMaker is generating. Table 21.1 shows some common FileMaker find requests and the SQL fragments that they produce. SQL queries that use the `%` wildcard character at the beginning and end of the search string cannot use an index in MySQL; thus, they must read the entire database. If you remove the initial wildcard (using the whole word search shown in the second line of Table21.1), you can use the index, but it is still not as efficient as the exact match query in the third row, which generates no wildcard characters in the SQL query.

Table 21.1 FileMaker Find Requests and SQL Implementations

FileMaker	SQL
`=abc`	`LIKE '%abc%'`
`="abc"`	`LIKE 'abc%'`
`=="abc"`	`LIKE 'abc'`

22

IMPORTING DATA INTO FILEMAKER PRO

Working with External Data

FileMaker Pro can work with data from a variety of other sources. It's possible to bring data directly into FileMaker from a number of different flat-file formats, as well as from remote databases and XML-based data sources. In many cases, you can open data files from other applications simply by dropping them onto the FileMaker Pro application as though they were native FileMaker files. FileMaker can also import data that resides on other computers (such as data from a remote database or a web-based XML data source).

Importing Data

FileMaker Go does not enable you to import data using the hands-on process as described in this chapter. You can use the hands-on process described here as the basis for a script step that can run in FileMaker Go. After data has been imported into a FileMaker Pro database, you can open that database in FileMaker Go.

Additional information on the topic of FileMaker data exchange can be found in Chapter 21, "Connecting to External SQL Data Sources," and Chapter 23, "Exporting Data from FileMaker Pro."

caution

Before you make the wholesale changes to your FileMaker database that importing creates, make certain you have backed up your database.

Flat-File Data Sources

Flat file is a generic term that refers to a file containing data in row-and-column format. If you think of a spreadsheet that holds data about personal contacts, the spreadsheet will have some number of columns, for attributes such as first name, last name, address, and so forth, and some number of rows, each one representing a single contact.

The formats of flat files vary. Some might separate one column from the next by tabs and one row from the next by carriage returns (a tab-delimited file). Another might use commas to separate column values. Some might include a first row that gives a name for each column. Some might be in a plain-text format that you could read with any text editor, whereas others might be in a specialized file format (such as FileMaker Pro or Microsoft Excel). In general, though, all flat-file data sources represent some variation of the idea of row-and-column data. The following sections show you the various ways of importing from flat files using a variety of options. There is tremendous flexibility, including the option to import flat-file data into a new table that FileMaker Pro creates automatically.

Choosing the Target Table

As you can tell from the previous description, a flat data file maps well onto the concept of a database table. And indeed, in FileMaker Pro, you do import data into only one table at a time. FileMaker chooses this target table for you automatically, based on the prevailing table context.

Current Table Context

The active layout determines the current table context. You can examine the table context for a given layout by choosing View, Layout Mode, and then choosing Layouts, Layout Setup and inspecting the Show Records From menu. Be sure to switch back to Browse mode before trying to import records, though.

For a full discussion of table context, **see** *Chapter 18, "Advanced FileMaker Solution Architecture."*

Initiating the Import

The example of importing tab-separated data is a good place to start because it's a typical text-based flat-file format. Many of the other text-based formats vary from tab-separated text only in small details. (Those differences are noted as necessary in this chapter.) Here, we walk through the process of importing from a tab-separated text file. As with other formats, you can import a tab-separated data file in one of three ways:

- Choose File, Import Records and then navigate to the file and select it.
- Choose File, Open and then navigate to the file and select it.
- Drag and drop the file directly onto the FileMaker Pro application.

The actions of importing and opening non-FileMaker files are similar in FileMaker Pro. The main difference is that the "open" action creates a new FileMaker file (complete with data from the originating document), whereas the "import" action is used to bring data into an existing file. You can also use the importing technique to bring in data from multiple files in a folder.

Automatic Recurring Imports

The File, Import Records command begins by letting you choose the file to import. A check box in the lower left of the dialog lets you choose to make this an automatic recurring import. If you choose that option, FileMaker creates a script and a layout for the import. See the "Creating Automatic Recurring Imports" section later in this chapter.

The Import Field Mapping Dialog

When you're importing data, after you choose your source file, FileMaker Pro presents you with the Import Field Mapping dialog, as shown in Figure 22.1. This dialog lets you choose how the records in your source file will be imported and in what order. (This example uses the Inventory Starter Solution to receive imported data.)

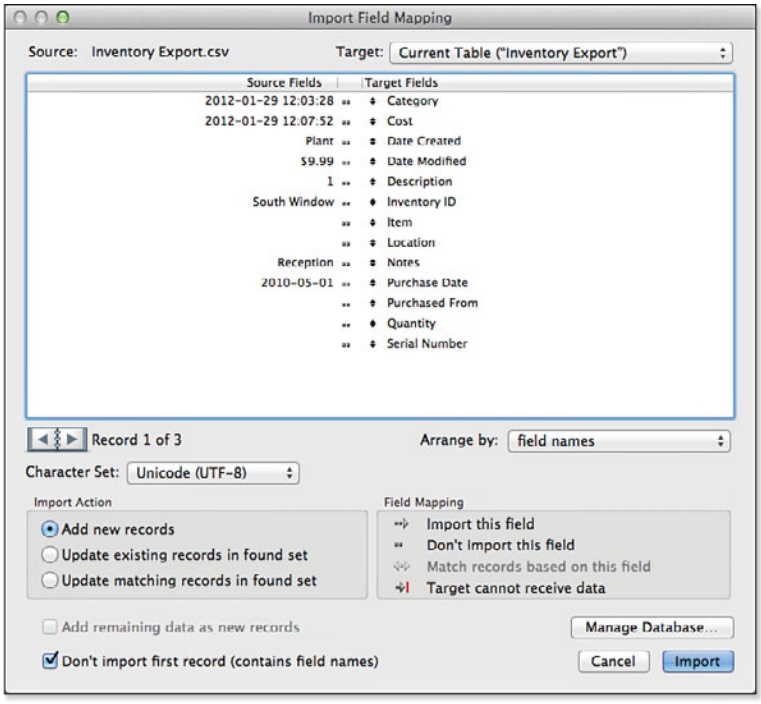


Figure 22.1
FileMaker's Import Field Mapping dialog. All importing processes pass through this dialog at some point.

The top of the Import Field Mapping dialog lists two filenames: Source and Target. Source is the file from which you're importing, and Target is the current table in the current file—in other words, the one that's receiving the imported data.

Choosing an Import Action

One of the things you have to choose in the Import Field Mapping dialog is the Import Action. It's visible in the lower left of the dialog. This choice tells FileMaker whether to try to add new records in the target table (one record per row of source data) or to try to update the existing FileMaker records with the source data. Updating on import is a topic in its own right, one that we deal with later in this chapter. For now, we cover what happens when we want to create new FileMaker records based on the source data.

Aligning Source and Target Fields

You also have to decide which fields in the target are to receive data and from which source columns they'll receive it.

If you look at the way the source fields line up with the target fields in Figure 22.1, you'll notice something isn't right. At the left, you see the value in the source field (the field to import); at the right, you see the field into which the data will be imported. The value "Plant" will be imported into Date Created; "South Window" will be imported into InventoryID; and "Reception" will be imported into Notes. Along with other mismatches, the import obviously is not correct as it first stands.

It's not possible to manipulate the ordering of the fields on the left (the source fields in the input file), but you can use the black up/down arrows next to each target field to change the target ordering manually. In this case, you just drag the Item field up until it is opposite Plant, and make the other adjustments in the same way.

caution

When you change the target field ordering by dragging a field manually, the field you drag changes places with the field you drop it on. Often you might want to drop the field you're moving between two others in the import ordering so that it pushes all the fields underneath it down a step, but this is not how the manual ordering works.

Deciding Where the Data Goes

After all the target fields correctly align with the source fields, you need to make sure they're all set to receive data. Between the columns of source and target fields is a column of field mapping indicators. The possible indicators are shown in the Import Field Mapping dialog, in the section at the lower right called Field Mapping.

The meaning of the different indicators is as follows:

- **Arrow—**Data from the source field will be imported into the target field.
- **Straight line—**Data from the source field will not be imported into the target field.

- **Equal sign**—The source and target fields are being used as part of a match criterion. This choice is available only if you've chosen one of the update import actions. We discuss the update options fully in the section "Updating Records with Imported Data."
- **Red X**—This indicates that the target field cannot receive data. Typical causes are that the target field is a calculation or summary field.

To sum up, make sure that all your target fields align with the correct source fields and that the mapping indicators are set to allow data to flow into the fields you intend to receive it.

Ways of Auto-aligning Source and Target Fields

In the Import Field Mapping dialog, you might have noticed a menu at the middle right called Arrange By. This menu simply governs the ordering of the target fields in the column on the right. It might be that you can line up the target fields with the source fields by putting the target fields in creation order, for example, or in alphabetical order by name. If you choose one of these options, FileMaker rearranges the target fields in the order you chose and then does its best to set the mapping indicators accordingly. Most likely you'll have to do some manual adjustment of the result, but these choices can often eliminate a lot of tedious hand labor.

One very useful choice in this menu is the first one: Matching Field Names. This choice is available only when the source file has some kind of data in it that attaches names to each of the source fields. Examples of such files are actual FileMaker files (of course) or flat data files with field names in the first row. If your source file contains field names that correspond to the names of target fields, you can choose this arrangement option, and all the fields with identical names will simply line up, no matter what position they have in their respective files.

note

Choosing Matching Field Names doesn't guarantee that the target fields will be able to accept data. If a source field has the same name as a field in the target table but the target field is defined as a calculation, the two will line up, but importing any data into the target field will still be impossible because you cannot import into calculation fields.

Scanning the Data Before Importing

When the Import Field Mapping dialog first opens, the Source column shows data from the first record in the source file. You might find that the first record's data is not enough to remind you of the appropriate field mapping, or you might want to scan through the source data for other reasons.

Directly under the Source column, you'll notice forward-arrow and back-arrow buttons, and a display that shows the total number of inbound records as well as the record you're currently viewing. You can use the forward- and back-arrow buttons to scan through the inbound data, either to verify that you have the correct mapping of source to target or to examine it for other reasons. Note, for example, in Figure 22.1 that a blank value is going to be imported into Cost.

tip

Browsing through other records in the file will help you understand whether a blank field is correct or whether it actually contains other data, such as a serial number. Documentation can help!

Performing the Import

After you verify all your field mappings and make your choice of import action (so far we've looked only at adding records), clicking the Import button starts the import proper. When the import completes, FileMaker displays a dialog telling you how many records were imported and whether there were any errors in the import process.

Assuming that there were no serious errors and at least some records were imported, the newly imported records are isolated in their own found set after the import is complete. This is an important point because if something is seriously amiss with the imported data, you have an opportunity to delete the whole set and start over. Or, more optimistically, the records are all there in one set if you need to perform any other operations on them as a group, such as a batch Replace operation.

caution

Depending on how you set up your field validation, the inbound data might or might not be acceptable. Under certain circumstances, FileMaker might reject imported records for this reason. See "Imports and Validation" in the "Troubleshooting" section at the end of this chapter for more information.

Updating Records with Imported Data

When you import data into a table, you have a choice as to whether FileMaker Pro should use the source data to create new records or add it into records that already exist. You can also choose to import the data into an entirely new table.

To match records, choose Update Matching Records in Found Set as the Import Action. This tells FileMaker that you're going to specify at least one pair of fields as matching fields. This pair of fields acts a lot like a match field in a FileMaker relationship: Each row (or record) in the source is matched with any corresponding records in the target.

FileMaker's Update Matching Records feature can be tricky. For an overview of some of the potential pitfalls, see "Matching Imports" in the "Troubleshooting" section at the end of the chapter.

In the Field Mapping section, choose at least one field on which to match records. That field is used only for matching; no data is imported to it (because the data value should be the same in the database and in the import file if the fields match). In addition to selecting the match field, you then select the fields into which you do want data to be imported. These fields may be empty, or your import may override existing data.

As a final note on update importing, you should be aware that the update affects only records in the current found set on the target side. If a record on the target matches a record in the source, but the target record is outside the current found set, the import does not affect it.

Updating Records Without Using Match Fields

You've probably noticed that another update option is available in the Import Action section. It's called Update Existing Records in Found Set, and it's simpler than the Update Matching Records choice. When this action is selected, rather than matching records based on a match field or fields, FileMaker matches records based purely on their position: The first record in the source updates the

first record in the current found set on the target side, the second source record updates the second found target record, and so on.

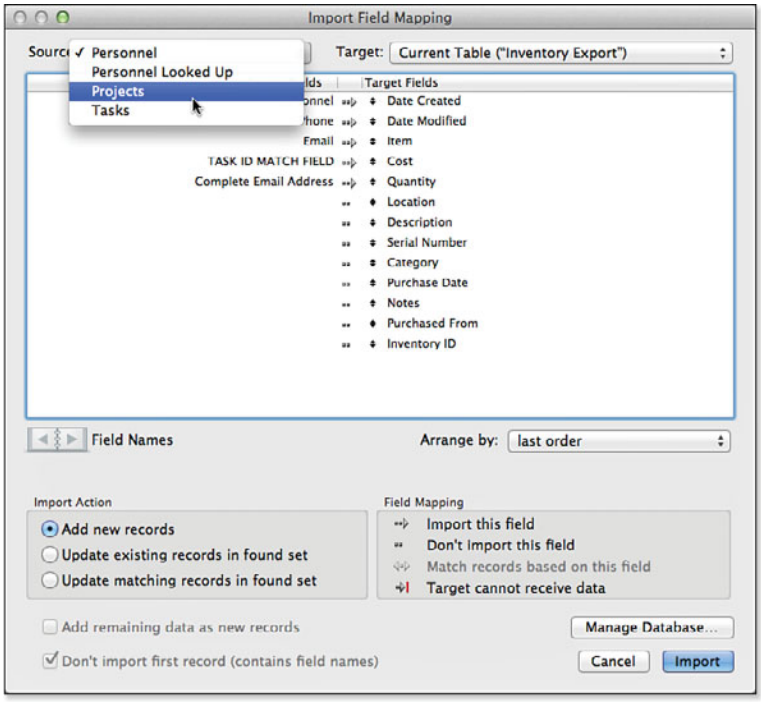
If the number of records in the source doesn't exactly equal the number of records in the target found set, FileMaker takes account of this. If there are more source records than target records, FileMaker skips the extra source records. If there are more target records than source records, the extra target records are left untouched. In either case, FileMaker provides an extra message to tell you what happened.

The only exception occurs if you check the box labeled Add Remaining Data as New Records. In that case, if extra records are available on the source side, FileMaker imports them into the target as brand-new records.

Importing from Another FileMaker Pro File

As you might expect, you are able to import from other FileMaker Pro files. If you choose FileMaker Pro as your source format, you also have to specify a table in the source file from which you want to draw data. This choice is available in the Import Field Mapping dialog, as shown in Figure 22.2.

Figure 22.2
When importing from a FileMaker database with multiple tables, you need to pick the source table from which you want to draw data.



Importing from a FileMaker file can be particularly convenient in that it allows you to use the Matching Field Names option for lining up the source and target fields. Developers will often choose to open a source file within FileMaker; create a new FileMaker file based on the originating document; and then use that new FileMaker file for importing, data cleanup, and so on.

For some other uses of the FileMaker-to-FileMaker import feature, **see** *the "FileMaker Extra: Exploiting the FileMaker-to-FileMaker Import" section at the end of this chapter,* **p. 564.**

Using an Import to Create a New Table

You can choose to create an entirely new table at the time of import and have the imported data flow into the new table. Figure 22.3 illustrates this feature.

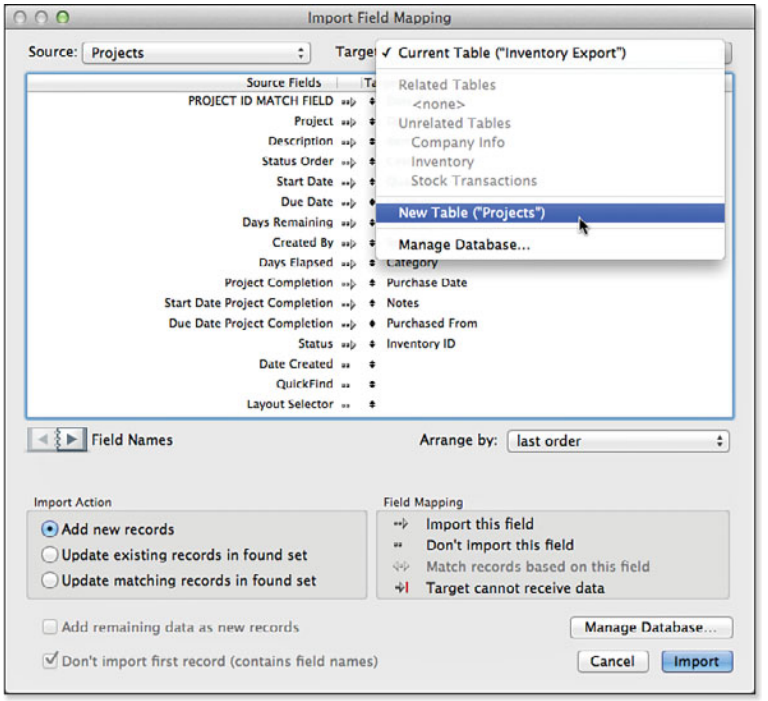


Figure 22.3
FileMaker Pro enables you to create a new table from imported data.

The new table will behave in many ways like a table created by choosing File, Open and opening the data source directly; see the discussion of this behavior earlier in this chapter. This feature is particularly useful, though, when you are importing from another FileMaker table. In this case, the entire schema of the table (including calculation and summary fields, for example) is re-created.

tip

Note that information such as value lists, custom functions, relationships, and security privileges will not be imported because they are attached at the file level rather than at the table level.

The newly created table will be an exact copy of the old one, including things such as field IDs, which is important if you're re-creating this table as a way of consolidating two formerly separate FileMaker files.

Importing from a Microsoft Excel File

FileMaker Pro has some special capabilities for importing data from Microsoft Excel documents. FileMaker is aware of multiple worksheets within an Excel document and of any named ranges (groups of cells that have been given specific names).

After you choose the specific part of the Excel document you want to import, the rest of the import proceeds. If you're bringing Excel data into FileMaker by choosing File, Open, and selecting an Excel file to open, FileMaker creates a new FileMaker file, as it does when opening other importable file types. In this situation, FileMaker can apply a little extra intelligence to creating the new FileMaker file. If a column in the Excel file contains only one type of data (numbers, text, dates), FileMaker assigns a suitable field type to the resulting FileMaker field. If the data in the column is somehow mixed—that is, the column contains some data that looks like numbers, and other data that looks like dates, for example—then the resulting FileMaker field will be a Text field.

FileMaker Pro supports Excel workbooks import (.xslx) as well as older versions (.xsl).

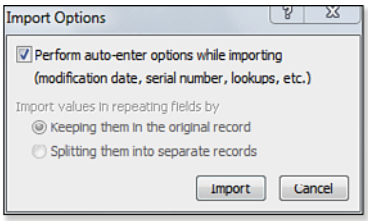
note

When importing from an Excel file, FileMaker brings in only the raw data it finds in the file. FileMaker does not import Excel formulas, only their results. FileMaker imports neither graphics nor charts, nor does it import notes. Programming logic, such as Visual Basic macros, is also not imported by FileMaker.

Setting Import Options and Reviewing Status

No matter which import technique you use, just before the import is actually executed, you will see the dialog shown in Figure 22.4.

Figure 22.4
Set import options.



The most important setting in this dialog is the initial check box that controls auto-enter options. If it is checked, the auto-enter options will be performed; records will have new serial numbers and modification timestamps. This may be what you want if you want them to be marked as new to your FileMaker database. However, if you are merging data from another data source (particularly a FileMaker database with its own timestamps), you may want to preserve those, so you need to make certain that the check box is not checked.

In the case of serial numbers, you need to think through your actions. First of all, if serial numbers are used as keys for related records, changing them (by using the auto-enter option) will destroy all the relationships based on old serial numbers. However, keeping the old serial numbers may result in duplicate serial numbers, and that can be just as bad. The simplest way to handle this situation is to browse the data in the new and old databases. Find the minimum and maximum serial number in both databases: If you are lucky, there will be no overlap. If there is an overlap, consider how many records are involved and then plan to resolve duplicates either manually or with a script. When the process is complete, you may want to use the Options dialog for the affected field in Manage Databases to set the next value for the serial number to a higher number than is found in either file so that future serial numbers will be safe from duplication.

When the import is finished, a dialog will show you the results, as shown in Figure 22.5. Check to see whether errors were encountered and resolve them before proceeding. They will not go away on their own.

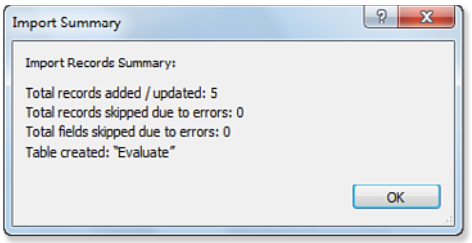


Figure 22.5
Review import results.

Importing Multiple Files from a Folder

FileMaker can import data from several files at once. In this batch mode, FileMaker takes the data from a file and imports it into one or more fields in a FileMaker table. FileMaker can also bring in information about each file's name and directory path.

FileMaker can work with two types of data when performing a folder import: image files and text files. In the case of image files, FileMaker can bring the image data from each file into a container field so that each image can be viewed inside FileMaker. In the case of text files, FileMaker brings the entire contents of the file into a specified text field.

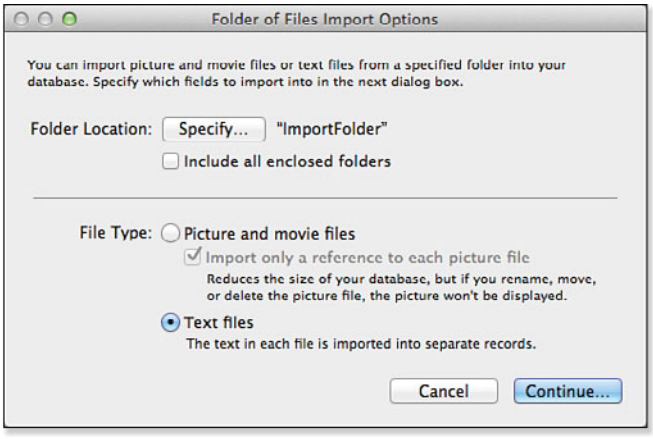
caution

FileMaker can store a maximum of 2GB of data in a single field. This amount might seem like a lot, and it is a lot compared to the limit of 64KB that was in force in previous versions of FileMaker! But it follows from this that you shouldn't import text or image files into FileMaker if any single imported file will be larger than 2GB.

Importing Text Files

Assume that you have a folder with a number of plain-text files in it. Assume also that you have a FileMaker database that has a table in it with fields called TextContent, FileName, and FilePath. If you select File, Import Records, Folder, you'll see FileMaker's Folder of Files Import Options dialog, as shown in Figure 22.6.

Figure 22.6
FileMaker kicks off the Import from Folder process with a special initial dialog.



In the upper area, you can choose the folder from which to import data. You can also choose whether to confine the import to files at the first level inside the folder or whether to drill into all the subfolders that might be below the top level.

After you choose a folder from which to import, choose the file type. To import from text files, choose the Text Files option and click Continue. You'll then see a folder import dialog that is similar but not identical to the regular Import Field Mapping dialog, as shown in Figure 22.7.

When you're doing a folder import, the names and contents of the source fields on the left are fixed: They depend on the type of file from which you're importing. When you're importing from text files, the source fields are called Text Content, File Name, and File Path. These fields contain, respectively, the actual text content of the field, the name of the individual file from which the data is coming, and the full name of the path to the file. As with any other data source, you can choose to import some or all of these fields, and you can choose how to map them to fields in the FileMaker table that's the target of the import. You can also choose to create a new table as the example in Figure 22.7 demonstrates.

Unlike imports from other kinds of flat-file data sources, FileMaker's batch text import brings the entire contents of each text file into a single FileMaker field.

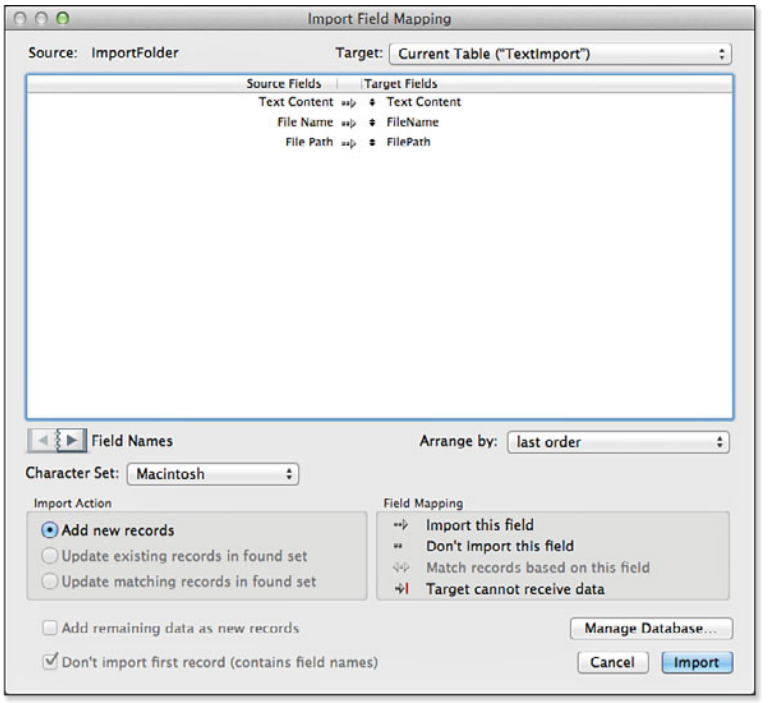


Figure 22.7
When you are importing from a folder of files, the source fields have a special name and meaning.

Determining File Type

When you choose to import files from a folder, FileMaker scans the files in the directory to determine which ones are of the right type to import. So, for each file in the folder, FileMaker decides whether it's an image file if you're importing images or a text file if you're importing text. But how does it make this determination?

If you're familiar with the way Mac OS X and Windows handle file types, you know that the file's extension (.html, .jpeg, and so on) often has a lot to do with it. Applications often use the file extension to determine whether an application owns that file type and can try to open it.

FileMaker's batch import determines file type differently, depending on platform. On Mac OS X, FileMaker looks first at the file's type and creator—special information (also called metadata) that Mac OS X stores with each file. If a file has no type or creator (for example, if it was created on a non-Macintosh platform), FileMaker falls back on the file extension. Windows, by contrast, has no file type metadata, so FileMaker simply relies on the file extension to determine whether a file is eligible for a batch import.

All this means that FileMaker has no other innate intelligence about file types. If you take an image or PDF file in Windows and give it a .txt file extension, FileMaker considers it eligible for

a text import and tries to bring its content into a text field. Likewise, if you strip out the file type and creator on the Mac and manipulate the extension, it's possible to confuse FileMaker about the file type.

To see a file's type and creator in Mac OS X, if you have the Apple Developer Tools installed, you can use the command-line tool /Developer/Tools/GetFileInfo to see file metadata and /Developer/Tools/SetFile to change the metadata. Alternatively, you can use a shareware tool such as Xray (www.brockerhoff.net/xray/).

Importing Image Files

Importing image files from a folder is quite similar to importing text files. As with text files, you need to choose a source folder and decide whether to drill down into any subfolders as well. The fields that can be imported are the File Name and File Path, as for text files. There is no Text Content field to import, but there are Image and Image Thumbnail fields.

Images or References?

Image data can take up a great deal of storage space, and it might not make sense to try to store thousands of high-resolution images inside a FileMaker file. Accordingly, FileMaker offers you the option (when importing images from a folder) to import only a reference to each file, rather than the entire contents of the image. If you choose to import a reference, FileMaker remembers where the image is stored on disk and refers to it when necessary in a fashion similar to the way in which Mac OS and Windows work with shortcuts and aliases.

➥ *There are some additional considerations when using container fields with FileMaker's Instant Web Publishing:* **See** *"Container Fields,"* **p. 593.** *Also, see the discussion of container fields in Chapter 3, "Defining and Working with Fields and Tables,"* **p. 89.**

➥ *Beginning in FileMaker 12, you can let FileMaker manage files that are placed in a container. As described in the "Container" section of Chapter 3, "Defining and Working with Fields and Tables," you can set an option for a container field so that when you import a file to the container field, it is stored outside the database file in a location you specify and that FileMaker can manage. This can combine the benefits of importing files and using only references.*

The benefit of storing references is, of course, that they take up much less space in the database. The disadvantage is that if you move or rename the original files in any way, FileMaker will no longer be able to find them, and the images will not display in FileMaker or be otherwise usable.

This situation is especially problematic if the file containing the images is hosted for multiuser access. Each user of the system has to see the image directory via the same network path. Because Windows and Macintosh handle server paths differently, creating a unified server structure to work in both environments could be challenging.

In the end, the decision as to whether to import whole image files or just references is up to you, keeping in mind the trade-off between the flexibility of having all images stored directly in the database versus the increased capacity that comes from working with the file references alone.

Images Versus Thumbnails

When you import data from text files, you can bring in up to three pieces of data: the filename, the full path to the file, and the text contents of the file. With image files, it is possible to bring in four pieces of data. As with text files, you can bring in the filename and file path. You can bring in the full contents of the image file (into a container field, presumably), and if you choose, you can bring in a smaller version of the image, called a thumbnail.

Naturally, a full-sized image can take a lot of space, so FileMaker gives you the option of bringing in only a smaller thumbnail instead. You can bring in the thumbnail in addition to the larger image or instead of it. (Of course, you could choose to import just the filename and path if that suits your purpose.)

In FileMaker 12, you can specify that thumbnails are created as needed or are stored when you access large images in a remote database. This is described in Chapter 3.

tip

FileMaker doesn't give you any control over how it creates thumbnails during the image import process. You might find that although you do want to store only a smaller copy of the image in the database, FileMaker's thumbnail process doesn't give you what you want. You might want the thumbnails a little smaller or larger or with some kind of color adjustment. If so, you will want to experiment with creating your own thumbnails first and import them instead.

Manipulating Images

With a tool such as Adobe Photoshop, you are able to create batch-processing scripts (called *actions* in Photoshop) that can apply a series of transformations to every image in a folder. You might want to create an action to shrink every image to 120 pixels wide, 72 dots per inch, and save it as a high-quality JPEG with a two-pixel black border. You could then batch-import the resulting custom thumbnails. In doing so, make sure to import the image data rather than the thumbnail data: If you asked FileMaker for the thumbnail data, your classy custom thumbnails would be further scrunched down into thumbnails of thumbnails—probably not the desired effect.

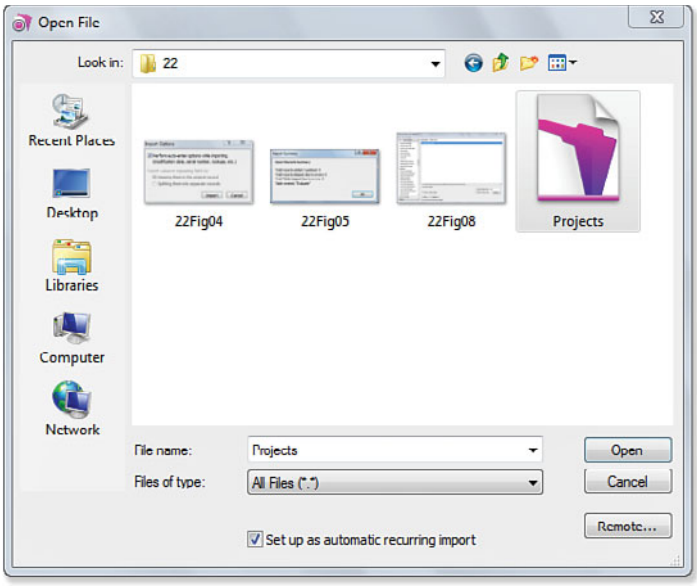
Scripting Imports with FileMaker

The following sections explain the two ways you can script imports easily with FileMaker. The first has FileMaker create an import script for you. The second enables you to create your own import script. You can also choose a combination by letting FileMaker create the script and then modifying it yourself.

Creating Automatic Recurring Imports

If you want FileMaker to create the import script for you, begin as you normally would with File, File Import. In the Open File dialog where you select the file to import, choose the file to import, but before clicking Open click the Set Up As Automatic Recurring Import box at the bottom, as shown in Figure 22.8.

Figure 22.8
Set up an automatic recurring import.



After you have chosen the import file, you are then asked to provide the recurring import information, as shown in Figure 22.9.

This information is going to be placed in the script that is generated, so you will be able to change it if you want. However, in normal circumstances, it is best to leave it alone. Note that the recurring import will expect the file to have the same name each time it is imported. This works well with scripted exports from other systems that need to be imported into FileMaker. However, some scripted exports may change the name of the file (perhaps including the date and time in the file name). If that happens, modify the script before running it.

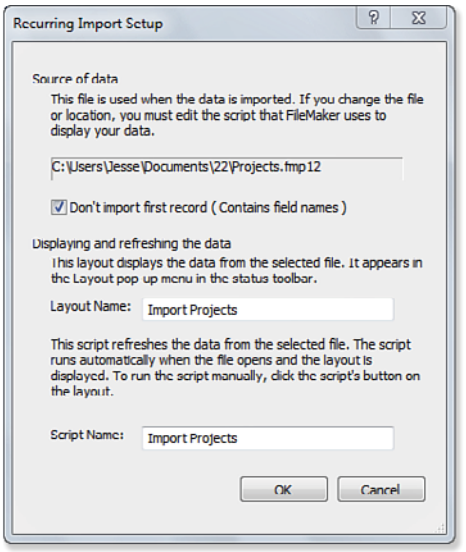


Figure 22.9
Provide import information.

Know Which Side You're On

Be careful about your naming conventions. Remember that every file you import usually has been exported from another system. The export comes first in time, so that file may be called an export file there. If you simply choose that file to import, the script and the layout that is automatically created for import will have that name, which can be confusing. Many people agree that the moment at which to change the name is the moment of import: In other words, importing a file called Export Projects into a FileMaker layout called Import Projects may, in the long run, be the best naming convention, particularly if the import and export processes are managed by different people or groups. Alternatively, you may opt for more verbose but unambiguous names such as SQLExportProjects and FileMakerImportProjects.

Using a Script to Import Data

Like most other actions in FileMaker Pro, a script can trigger a data-import operation. It's possible to save your import settings in a script for later reuse as well. Creating an import script is important if you want to import data to FileMaker Go: The manual steps described previously in this chapter are not available in FileMaker Go, but a scripted import will run.

A scripted import has a few steps and options that are slightly different from the regular File, Import Records method. To import records from within a script, choose the Import Records script step and add it to your script. ScriptMaker gives you several choices, as shown in Figure 22.10. The data source is the same set of choices as in the submenu of File, Import Records. The Specify Import Order choice brings up the Import Field Mapping dialog for you to use to control the import. Unless you choose to perform the import without a dialog, users can modify these choices when the script runs.

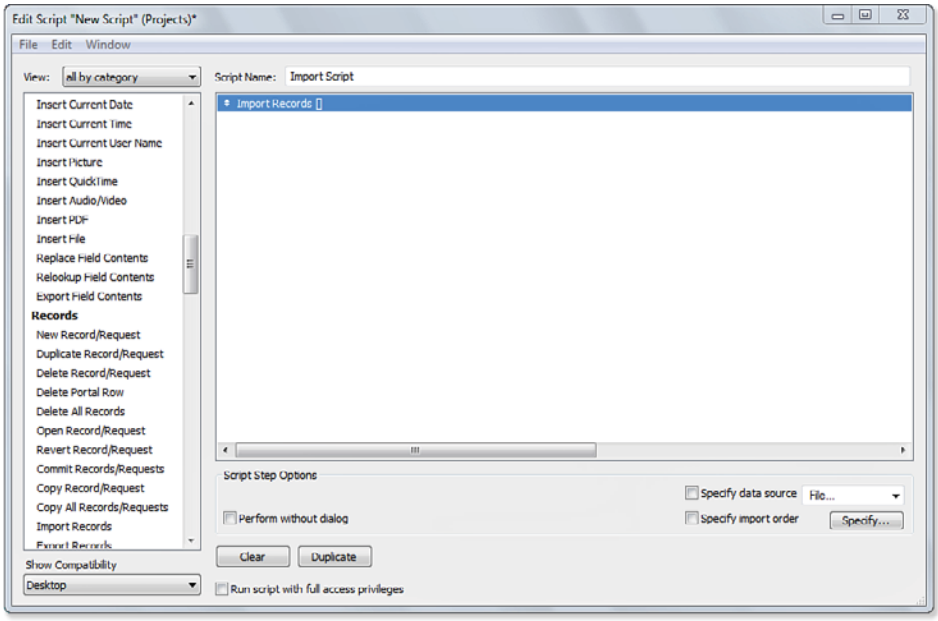


Figure 22.10 FileMaker enables you to save a number of options when you import records from within a script.

When the `Import Records` script step is selected, a Specify Data Source menu at the lower right gives you access to a set of options identical to those you see when you choose File, Import Records. Using this selection, you can save all the important information about your data source. For files, this means mainly the filename. For folders, it includes the file type and the choice of whether to save references. For ODBC data sources, it includes the DSN information, password, and other data such as a SQL query.

When an `Import Records` script step is selected, a Specify Import Order button at the lower right gives you access to the Import Field Mapping dialog, where you can set any or all of the relevant import-mapping features. Finally, as with other script steps in FileMaker, you have the choice of performing the import with or without dialogs. If you choose to run the import with dialogs, the user can re-specify any aspect of the data source or import order on the fly. If you choose to run it without dialogs, the import is a canned process that uses all the saved options you specified.

tip

When performing a complex import, you might want to save drafts of the import into a script as you go. That way, if you make a mistake or need to change things, you don't run the risk of FileMaker forgetting the import specification you worked so hard on.

caution

A scripted import can go awry if your database structure changes after you configure the script. Adding fields should be no problem, but deleting any fields, especially those involved in the import, disrupts your field mappings, and data will no longer flow into the correct fields.

Using Bento Data Sources

Bento is a personal database from FileMaker; it runs only on Mac OS X Leopard and later versions. It is not a stripped-down version of FileMaker; rather, it is something quite different. Its tight integration with Mac OS X means that iCal tasks and events, Address Book, and Mail messages can all be accessed. Combined with MobileMe, you can use Bento and your iPhone to create an environment in which your calendar, addresses, and email messages are always where you are.

For more information, **see** The Bento Book: Beautify and Simplicity in Digital Organization*, by Jesse Feiler.*

You can import data from Bento libraries directly into FileMaker. To do so, choose File, Import Records, Bento Data Source. If there are collections for that data source, you will be prompted to choose the one you want, as shown in Figure 22.11.

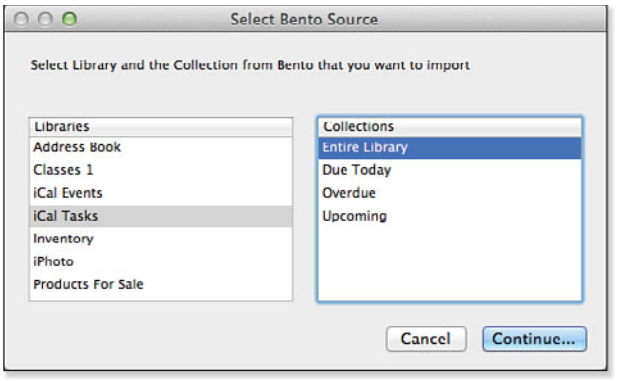


Figure 22.11
Import from a Bento data source.

If there are no collections, the entire library is imported. Just as with any FileMaker import, a single table is imported; this means that related Bento records are not imported unless you import those tables from Bento.

Because Bento interacts with your Address Book (as well as iCal for Tasks and Events), you can use Bento's collections that it automatically creates from Address Book groups and iCal Calendars to quickly import data to FileMaker. By choosing to import a Bento collection that is created from a group or calendar, you quickly import relevant information into FileMaker.

Troubleshooting

Matching Imports

I can't get an import to work using the Update Matching Records option. The outcome is never what I expect.

When you choose the Update Matching Records option when importing data into a FileMaker table, FileMaker tries to match records in the source to records in the target, based on the specified match field or match fields. We've been assuming that there will be at most one source record and one target record that share the same match criteria. But what happens if there are multiple matches on either or both sides?

Assume that you're doing a matching import based on a stock control number. If several records in the source data have the same stock control number, FileMaker uses the data from the last of these records to update matching records in the target (assuming that there are any).

On the other hand, if there are multiple records with the same stock control number on the target side, FileMaker updates them all with whatever turns out to be the matching data from the source side. So, FileMaker updates all target records with the same value in their match field(s) with the same data from the source, whether that means updating two target records or two thousand.

If you put both scenarios together, and multiple records in both the source and target share the same stock control number, the outcome is as follows: Data from the last such record in the source is used to update all the matching records in the target. If four matching records were in the source, and 19 in the target, data from the fourth matching source record would be used to update all 19 matching target records.

Imports and Validation

I imported data, but some of it turned out to be invalid. I have field validation rules set up, but it seems as though FileMaker is ignoring them.

Everything depends on your field validation settings. When you apply validation to a field in FileMaker, you can choose to validate the data only during data entry, or always. If you select the Only During Data Entry option, the behavior is similar to previous versions of FileMaker: Imported data is not checked for validity, and it's up to you to handle the consequences. On the other hand, if you choose Always for the data validation on a field, imported records are checked. If this is the case, any record that does not pass a validation check is rejected, and the dialog that appears at the end of the import tells you how many records were rejected (although not which ones, unfortunately).

FileMaker Extra: Exploiting the FileMaker-to-FileMaker Import

You saw earlier that it's possible to import data into one FileMaker table from another. Those tables can be in the same FileMaker file or different ones. This capability has a number of useful and interesting applications.

Duplicating a Found Set

Occasionally, you'll encounter situations in which you want to duplicate a found set of records. Of course, as with most things in FileMaker, there are several approaches. You could write a script that starts at the beginning of the found set and loops through it, duplicating as it goes. But you'd quickly find you had some tricky record-position issues to deal with. (Duplicating records can change which record is the current one, so it can be hard to keep your place when looping through a found set.)

One general rule for speeding up FileMaker operations goes something like this: Where possible, replace scripts, especially looping scripts, with built-in FileMaker operations. FileMaker's Replace command is much quicker than a script that loops over a group of records and performs a `Set Field` step on each record. FileMaker's Delete Found Records command is quicker than a script that loops over a set of records and deletes each one. And so on.

Another choice is to export these records to a separate table and then import them into the original table again. A single script can control both the export and the import, and the logic is much easier to read and understand.

Duplicating Between Tables

Suppose that you have a simple order-tracking database. The database has tables for customers, orders, order lines, and products. Each order, of course, has one order line per product on the order.

Suppose also that users have said that they want to create new orders by checking off a number of products from a list and then having a new order be created with one line for each selected product. So, a user would check off Screwdrivers, Milk, and Roofing Tar in a product list, click a button that says Make Order, and see a new order with lines for the three selected products.

Again, you can do a number of things with scripts, but one elegant solution is to gather the selected products into a found set and then simply import that found set (well, the relevant fields from it, anyway) into the Order Lines table, thus creating one new order line per selected product.

23

EXPORTING DATA FROM FILEMAKER

Getting Out What You Put In

Much of this book concentrates on tools for data entry—for getting data into a database system. But that information often needs to be extracted again. Sometimes the extraction takes the form of a report of some kind. At other times, the best choice is simply to export the data into some specific format so that another program can import that data and work with it using different tools than might be available in FileMaker. Reasons for exporting might include the following:

- Perhaps you know someone who is compiling a quarterly report in Excel and needs some numbers from your FileMaker system.
- Perhaps the payroll system needs a list of employee names that you have in FileMaker.
- Perhaps you've been storing images for an upcoming ad campaign in FileMaker, but you would like to make all the images available so that they can be used on a CD without FileMaker.

This chapter covers various means for getting data out of FileMaker. There are two ways of doing so: exporting and real-time sharing. Exporting is a batch or offline process. Real-time sharing of data is discussed in Chapter 21, "Connecting to External SQL Data Sources."

*For more on saving and sending records as Excel or PDF files, **see "Delivering Reports," p. 309**.*

Information on real-time sharing of data can be found in Chapter 21, "Connecting to External SQL Data Sources."

Additional information on the topic of FileMaker web publishing can be found in Chapter 24, "Instant Web Publishing," and Chapter 25, "Custom Web Publishing with PHP and XML."

The Basic Mechanics of Exporting

The basic principles of exporting data from FileMaker are straightforward. You pick a single table or layout from which to export. (Although you start from a table or layout, you can reference related fields in the export process so that you can actually export from additional tables.) You then choose an output file format and file location and pick specific fields from your chosen table for export. Before you export, there are a few extra options you can choose that govern grouping and formatting of the exported data. That's all there is to it. Let's look at each step a bit more closely.

Choosing a Source Table

As with much else in FileMaker Pro, the starting point for a data export is determined by the user's *context* in the current system—specifically by the currently active layout, which in turn is tied (via its table occurrence) to an underlying data table (a.k.a. source table). So, the currently active layout controls implicitly which table is the source table for the export.

Choosing an Output File Format

After your context is established, you have to choose File, Export to begin the export process. The next step in that process is to choose an output file format and file destination. FileMaker offers you a choice of export formats, as shown in Figure 23.1. Some of these are plain-text formats, such as tab-delimited text, that could be read in any text editor; others are binary file formats that require more specific software to open, such as FileMaker Pro. This chapter goes into greater detail on available file formats in a later section.

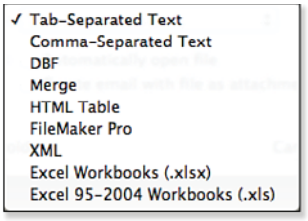


Figure 23.1
FileMaker Pro can export data to various formats.

When you chose File, Export, in addition to selecting your export format, you have the choice to automatically open the file after saving it and the choice to automatically create an email with the file attached. These choices parallel those available with the Save as Excel and Save as PDF features; the goal in all cases is to make the final delivery of the data faster and easier.

Selecting Fields to Export

After you've selected an output file type and destination, you're prompted to choose some fields to export via the dialog shown in Figure 23.2.

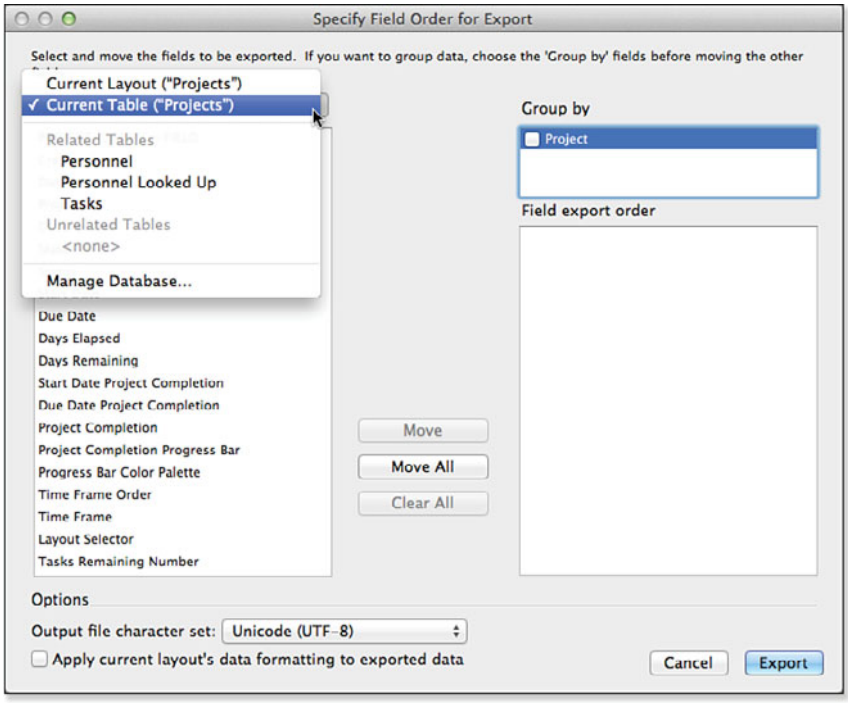


Figure 23.2
Most fields can be exported from a FileMaker database, but container fields cannot.

By default, the fields displayed in the list at the left are limited to those found on the current layout, as indicated by the Current Layout menu setting. That is always the first item in the list. It's also possible to switch the view to show all fields in the current or related tables, as well as to select fields from any related table for export.

*For more information on exporting related fields, **see** "Exporting Related Fields," **p. 570**.*

In addition to selecting fields for export, you also can select grouping options for the fields and choose whether to format the exported data according to the current layout formats. These options, too, are discussed in more detail later in this chapter.

*For more information, **see** "Exporting Grouped Data," **p. 571**, and "Formatting Exported Data," **p. 570**.*

At any point in the field-selection process, you can use the small up/down arrows beside each selected field name to change the order in which the fields are exported: Click the arrow and drag

to move the field, or use (Command-up/down) [Ctrl+up/down] if you prefer the keyboard. After you have a satisfactory field list, click Export, and the data will be exported to the file format and location of your choice.

Exporting Issues to Consider

Although the basic mechanics of exporting are simple, a few key points bear remembering:

- To export data from a file, a user must have sufficient privileges to do so. This is governed by selections within the user's privilege set. The user must have the Allow Exporting check box checked within the settings for her privilege set. Additionally, the user will not be able to export any records for which she does not have at least the capability to view the record.

  *For more information on privileges and security in FileMaker, **see** Chapter 12, "Implementing Security."*

- Data will be exported only from records in the user's current found set. To export data from all records in a table, the user first needs to choose Show All Records to ensure that no records are omitted from the found set. Regardless of the found set, any records that the user's privileges prevent her from viewing cannot be exported.

- There are certain practical limits on which fields you can export. Container fields can be exported only if the target file format is a FileMaker Pro file (we discuss other strategies for exporting files in container fields later in the chapter). And there are certain fields, such as summary or global fields, for which exporting might not always make sense, even though doing so is technically possible.

  *For more information, **see** "Working with Large Fields and Container Fields," **p. 572**, and "Formatting Exported Data," **p. 570**.*

Export File Formats

FileMaker's Export Records feature can create export files in various formats. Many of these are text based, and a few are binary. The following sections provide an overview of the available file types, with some specific notes on each. Each format has its own quirks and limitations. As is always the case with data import and export, you'll need to experiment to see just how a specific data set translates to a chosen file format—not just theoretically but practically. The sooner you can test the entire route from export to import with the actual software involved, the safer you will be.

Character Transformations

When exporting data, FileMaker often performs substitutions on certain characters that tend to cause confusion when they appear embedded in field contents. For example, FileMaker permits you to embed a tab character in field data, but because the tab character is frequently used as a field separator in text-based data, FileMaker transforms these internal tabs to spaces when exporting in

many cases. In the same vein, carriage returns within fields sometimes are transformed to the vertical tab character (ASCII code 11).

One other common transformation occurs when repeating fields are exported (for those formats that support it). Multiple repetitions of a field are often exported with the individual repetition data separated by the group separator character (ASCII code 29). Common transformations are listed in Table 23.1.

note

The specific transformations that occur depend on the output file format; see the notes on each format outlined in the following sections for further details.

Table 23.1 Character Transformation Information for Exporting FileMaker Data

Character	Transformation
Tab-separated text	One of the most common data interchange formats, the tab-separated text format, exports each record as a single line of text, terminated by a carriage return. The contents of individual fields are separated by the tab character. Return characters are transformed to vertical tabs. The repetitions of repeating fields are run together into a single string, with repetitions separated by the group separator character (ASCII code 29).
Comma-separated text	Comma-separated text (or values, commonly referred to as *CSV*) is another very common text interchange format. As with tab-separated text, records are separated by carriage returns; but individual records are separated by commas, and field contents are enclosed in quotation marks. Quotation marks already present in the data are turned into pairs of quotation marks, so `"data"` becomes `""data""`. The repetitions of repeating fields are run together into a single string, with repetitions separated by the group separator character. Returns within quoted field contents are not transformed.
Merge	The Merge format is intended for use with word processors and other applications that support mail-merge or similar functionality. Field names are fully preserved, as are internal tab characters. Internal returns are exported as vertical tabs. All repetitions of a repeating field are exported.
HTML Table	As the name suggests, this export format writes data from the selected records into a basic HTML table. Field names are output as column headers. Internal tabs are preserved, as are internal carriage returns. Field repetitions are exported into a nested table. Returns are transformed to ` `. Horizontal tabs are not transformed, but they are ignored by browsers.
FileMaker Pro	This export format will create a new FileMaker Pro file with a field structure that matches the fields being exported. This is the only file format into which it's possible to export data from container fields. Not all FileMaker field types are preserved; summary fields become number fields, and calculation fields become data fields of the appropriate type (whatever the output type of the calculation is defined to be).
XML	FileMaker can export its data with an XML format, or *grammar*, called FMPXMLRESULT. You can choose whether to export raw XML or to apply a style sheet as the XML is exported. Returns and tabs are exported without transformation.
Excel	FileMaker Pro can export data to a file in the native Excel formats (.xls and .xlsx). When doing so, you can specify certain parameters of the result file, such as the name of the target worksheet, and whether to use the field names as column headers, as shown in Figure 23.3. Internal tabs and carriage returns are converted to spaces. Only the first repetition of a repeating field is exported. FileMaker fields will be assigned the appropriate Excel data type in their resultant columns, where possible. For an example of where this is not possible, consider FileMaker data that falls outside the range of dates supported by Excel.

*For more information on FileMaker's XML grammars, **see** Chapter 25, "Custom Web Publishing with PHP and XML."*

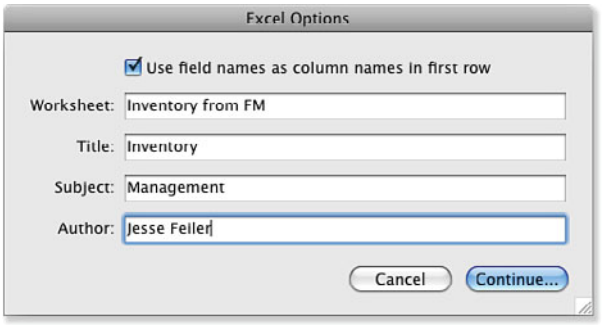


Figure 23.3
You have additional options when using the capability to export to Excel.

Formatting Exported Data

FileMaker maintains a distinction between the way data is stored in a field and the way it is displayed. For example, although all dates are stored internally as simple integers, they might be displayed in many different date formats, such as "1-3-2019." Or a number, stored internally with 17 digits of precision, might be displayed with just three or four digits. Not one of these display options has any effect on the data stored in the field; these options simply affect the way the data is shown to the user.

On FileMaker layouts, these formatting options are governed by choices made via the Inspector, in Layout mode. Some of these formatting options can be made to carry through to data when it's exported. To do so, when specifying fields for export, check the box labeled Apply Current Layout's Formatting to Exported Data. When this choice is selected, any formatting options applied via the Number, Date, or Time formatting dialogs are preserved. Text formatting, even character-based formatting such as uppercase, is not preserved. Date and time formatting may both be applied to a timestamp field and will be carried through on export.

Exporting Related Fields

All exporting in FileMaker takes place from the context of a single table. In general, then, it's not possible to export data from several tables independently in one stroke. It *is* possible, though, to export data from tables related to the current one, whether immediately or more distantly.

Doing so is a simple matter of choosing fields from related tables when specifying fields for export. If the related fields are in the layout you are using, they will be shown in the Specify Field Order dialog, as shown previously in Figure 23.2. When there are related records in a one-to-many relationship, you will get all the existing child records. The main record's value will appear in the first record.

Exporting Grouped Data

A typical export outputs some data from each record in the current found set.

Sometimes you might get multiple sets of information per current record if you export related fields. ***See*** *"Exporting Related Fields,"* ***p. 570,*** *for more information.*

To output summary data, you need to have one or more summary fields defined.

But what if you don't want data for each and every record? What if you want to export only data that summarizes information from the current record set, such as you might see in a subsummary report? FileMaker makes this possible as well.

For more information on summary fields and summary reporting, ***see*** *"Working with Field Types,"* ***p. 87,*** *and "Using Summarized Reports,"* ***p. 295.***

It now remains to use this summary field in an export. The process is similar to that required for preparing a subsummary report for display. First, isolate the transactions to be summarized (for example, to summarize across all transactions, you would perform Show All Records). Next, sort by the field that would be the break field if you were displaying the data in a subsummary report to group the data. You can sort by more than that field. As shown in Figure 23.2, the various sort fields in use at the time you export data are shown; you select the check boxes on any for which you want summarized data.

When you use more complex sorts and summary field choices, more complex summarized exports are possible.

Exporting to Fixed-Width Formats

Many computer systems exchange data in some form of *fixed-width* format. This term refers to formats in which an individual field always contains a certain number of characters of data. Data that's too wide for the field width is sometimes truncated to fit. Data that takes up less space than the field width allows is *padded* with a padding character, such as a zero or a space, to bring it up to the specified width. For example, the number 797 in a ten-character fixed-width format might be rendered as "0000000797" (left-padded with zeros). The name Tomczak displayed in a 15-character fixed-width format might be displayed as "Tomczak " (right-padded with spaces). Fixed-width formats also sometimes simply run all the columns together into a single big fixed-width string. There's no need for internal field separators; because the exact width of each field is known, you can easily determine where each field's data starts and stops.

If you need to export FileMaker data to a fixed-width format, you'll have to do a bit of work by hand; FileMaker has no built-in support for exporting to a fixed-width format. At a minimum, you'll need to define some calculations to perform padding and concatenation. If you want to build a more permanent framework for working with fixed-width data, you can consider developing a small library of custom functions to do some of the work.

Padding data is a straightforward activity using FileMaker calculations. Say you have a number field called OrderTotal. To left-pad this number with zeros and enforce a fixed width of ten characters, you would use the following calculation:

```
Right( "0000000000" & OrderTotal; 10)
```

If you think about that for a moment, how it works should be clear. The calculation tacks ten zeros onto the *left* of the numeric value and then takes the *rightmost* ten characters of the result. Likewise, to right-pad a text field called FirstName with spaces to a width of ten characters, you would use a calculation that looks like this:

```
Left( FirstName & "          "; 10)
```

Finally, if you needed to run a set of these fields together into a single fixed-width row, a calculation that concatenates all the individual padding calculations together using the & operator would suffice. You could also create a single row-level calculation without bothering with individual calculations for each field:

```
Right( "0000000000" & OrderTotal; 10) & Left( FirstName & "          "; 10)
```

note

Calculations such as these are fine for simple or occasional fixed-width exports. FileMaker also ships with an XSL style sheet, called fixed_width.xsl, which can be applied to a FileMaker data set on export to produce a fixed-width export. The style sheet supports only a single fixed width for all output columns. For more complex needs, you can build a tool of some sort to streamline the process.

Working with Large Fields and Container Fields

Most of the formats discussed so far are predominantly text oriented; that is, either they treat exported data as text or, at the very least, they describe its attributes using text-based formats. But FileMaker has extensive capabilities for handling binary data as well, via the container field type. FileMaker can import files in batches, as discussed in the preceding chapter. FileMaker also has tools that allow you to create a batch export of binary files.

The key to most such exporting operations is the Export Field Contents command, found in the Edit menu. You can manually enter a single field on a FileMaker layout and choose Edit, Export Field Contents, and the contents of that one field are exported to a file of the appropriate type: a text file for most field types or the actual file contents of a container field. (For example, exporting from a container field containing a file called hurricanes.dbf produces exactly the hurricanes.dbf file.) The Export Field Contents option is not available for a container field unless the field contains something—either an embedded file or a reference to a file.

*For more information on the batch import of images, **see** "Importing Multiple Files from a Folder," **p. 554**.*

When used via the menu, Export Field Contents exports the contents of one selected field from one record. It uses UTF-16 Little Endian, so all characters are preserved, including returns and horizontal tabs. To create something like a batch export of images, you need to write a script that uses the

`Export Field Contents` script step. Using scripted exports is a powerful technique that's covered in the next section.

Scripted Exports

All the techniques covered so far involve manual export operations, in which the user drives the process by hand, including the selection of output file type and filename as well as the selection of fields for export. Exporting, though, is often an operation you want to be able to perform repeatedly, on demand. You might have to export a membership list to a text file periodically or create a file containing information on this month's invoices to send to an accounting system that doesn't interact with FileMaker. In such circumstances, it's typical that you'll want to export the same set of fields, for different data sets at different times. In these cases, it makes sense to consider using a script to perform the export.

All aspects of exporting can be scripted, from the selection of the records to be exported; to the determination of output file type, filename, and location; to the choices of specific fields and export options.

You can use the `Get ( TemporaryPath )` *function to export data using schedules in FileMaker Server. For more information,* ***see*** *Chapter 27, "FileMaker Server and Server Advanced."*

The `Export Records` script step is similar to the `Import Records` script step: You can specify the output file and the export order, just as you could specify the input file and the import order.

This page intentionally left blank

24

INSTANT WEB PUBLISHING

Overview of Instant Web Publishing

Instant Web Publishing (IWP) lets you publish your FileMaker databases to a web server with remarkably little effort.

Broadly speaking, Instant Web Publishing is one of several options for sharing data from a FileMaker database to the Web. The other options include exporting static Hypertext Markup Language (HTML), exporting XML and transforming it into HTML with a style sheet, and Custom Web Publishing (CWP), which involves doing Hypertext Transfer Protocol (HTTP) queries against the Web Publishing Engine and using PHP with the Web Publishing Engine. Although less common, you can also use Web Publishing with ODBC or JDBC with FileMaker Server Advanced.

*For more details on Custom Web Publishing with PHP, **see** Chapter 25, "Custom Web Publishing with PHP and XML."*

The goal of IWP is to translate to a web browser as much of the appearance and functionality of a FileMaker Pro database as possible, without requiring that a developer do any additional programming. FileMaker layouts are rendered in the user's browser almost exactly as they appear to users of the FileMaker Pro desktop application. To give you an idea of what this looks like from the user's perspective, Figures 24.1 and 24.2 show an example of a layout rendered both in FileMaker Pro and through IWP in a web browser. (The database is the FMServer_Sample that is installed automatically with FileMaker Server.)

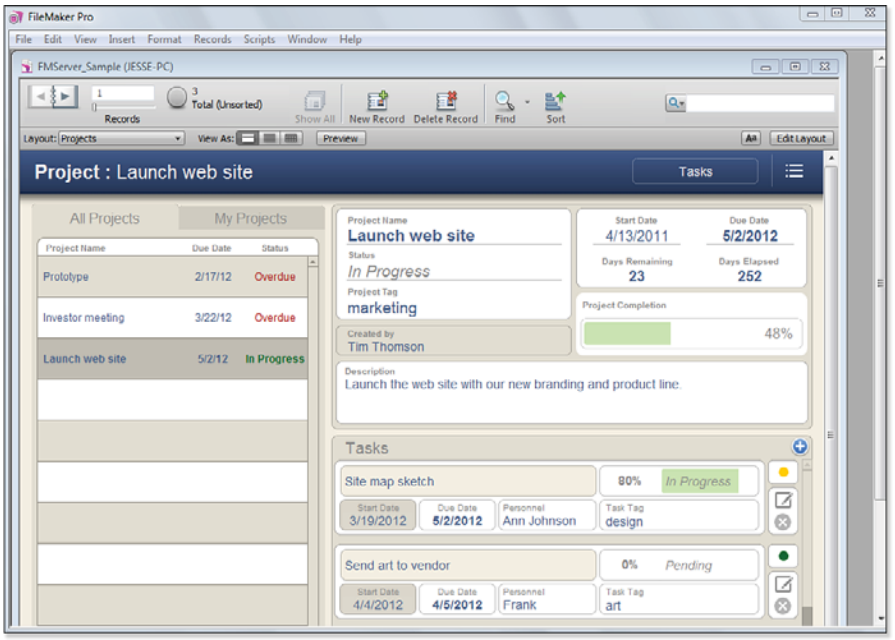


Figure 24.1
View FMServer_Sample with FileMaker Pro.

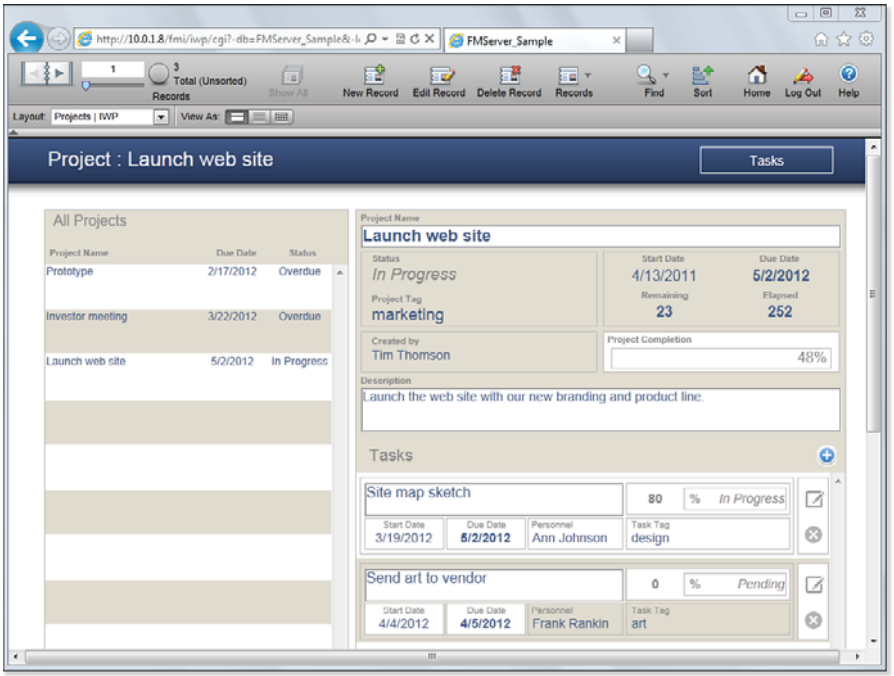


Figure 24.2
View FMServer_Sample with Instant Web Publishing.

IWP is more, though, than simply rendering your layouts as web pages. IWP users have much, if not all, of the same application functionality as do FileMaker Pro users. They can run scripts and view, create, edit, and delete data just like traditional FileMaker Pro users.

Almost all the differences between IWP and FileMaker access to databases have to do with the fact that in IWP the database displays in a web browser, and in FileMaker it displays in a FileMaker window. The consequence of this is that the menus at the top of the window or display belong to the browser, not to FileMaker. Therefore, all FileMaker menu commands available in IWP have to be provided as buttons, triggers, or other controls in the IWP controls area at the left of the window.

To edit data in IWP, click Edit Record in the toolbar, as shown in Figure 24.2. The toolbar changes to the editing toolbar shown in Figure 24.3. You can edit data, submit changes, or cancel them.

caution

As of FileMaker Pro 12, Instant Web Publishing works only with the Classic theme.

tip

The fact that the IWP menu bar has the browser's menu bar should be a familiar thought. You must implement functionality without resorting to menu commands, just as is the case on the iOS devices where there is no menu bar.

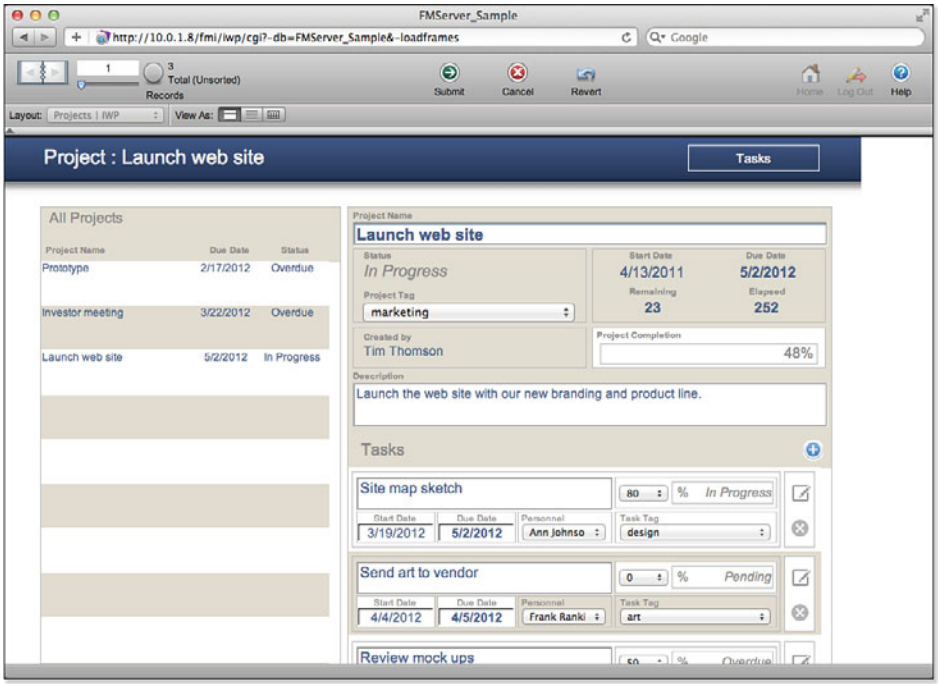


Figure 24.3 Edit data in IWP.

Most of the desktop commands are available in the IWP toolbar although, as you see in Figure 24.4, they may be in different places. Here, some of the most frequently used commands from the desktop Records menu are moved to a submenu in the toolbar.

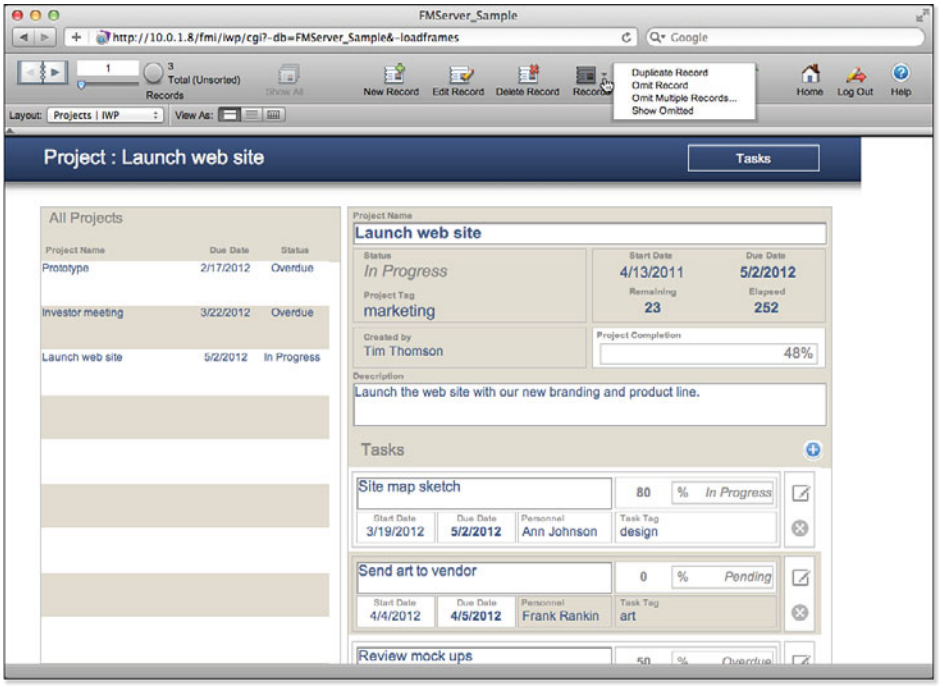


Figure 24.4 Commands may be in different places on IWP.

Getting Started with IWP

After you decide that IWP is something you want to try, there isn't too much you'll need to do to get started. There are two ways to deploy IWP. You can use the regular FileMaker Pro desktop application, in which case you're limited to publishing a maximum of ten database files to at most five concurrent users. Alternatively, you can use FileMaker Server Advanced, which allows for significantly more files and users. The configurations for these options are covered in detail in the next section. (FileMaker Server itself does not support IWP.)

The host machine—whether running FileMaker Pro or FileMaker Server Advanced—of course needs to have an Internet (or intranet) connection. The host machine also must have a static IP address. If you don't have a static IP address on the host machine, remote users can have a difficult time accessing your solution. Finally, any databases you want users to access via IWP need to be open on the host machine.

tip

A static IP address can be provided by your ISP; it will allow people outside your organization to connect to the IWP databases. You may have to do a bit of research with your ISP. Some vendors try to make their offerings more consumer-friendly by avoiding technical terms such as "static IP address." You might have success if you inquire about a business account and look to see if a static IP address is one of the features of a business account. If you have a local area network that communicates with the Internet through a single connection, each machine on the network has a separate IP address. They can be created dynamically, or they can be assigned within the network. (They typically start with 192.168 or with 10.0, depending on the size of the network.) Your IWP host machine can have a static IP address within the network that you assign even if the entire network's IP address (visible from the outside) changes. This static IP address will allow everyone inside the network to consistently connect to the IWP host, but outsiders will not be able to do so.

To access your IWP-enabled files, remote users need to have an Internet connection and a compatible browser. Because IWP makes heavy use of Cascading Style Sheets (CSS), the browser restrictions are important and are something you need to consider carefully if you intend to use IWP as part of a publicly accessible website.

On Windows, the supported web browsers are Internet Explorer 8 or 9, Firefox, or Safari 5. Later versions of these browsers are supported as well. Obviously, these options might change with new releases of browsers and new versions of operating systems, so consult the Filemaker.com website for the latest configuration guidelines. Whichever approved browser is used, JavaScript needs to be enabled, and the cache settings should be set to always update pages. The web browsers on iPhone, iPad, and iPod touch do not support IWP. Use FileMaker Go instead.

Enabling and Configuring IWP

To publish databases to the Web via IWP, you must enable and configure IWP on the host machine, and you have to set up one or more database files to allow IWP access. Each of these topics is covered in detail in the sections that follow.

Configuring FileMaker Pro for IWP

Using FileMaker Pro, you can share up to ten databases with up to five users. To share more files or share with up to 100 users, you need to use FileMaker Server Advanced as your IWP host. FileMaker Pro can serve only files that it opens as a host. That is, it's not possible for FileMaker Pro to open a file as a guest of FileMaker Server Advanced and to further share it to IWP users.

Figure 24.5 shows the Instant Web Publishing setup screen in FileMaker Pro. In Windows, you get to this screen by choosing File, Sharing, Instant Web Publishing. On a Mac, choose FileMaker Pro, Sharing, Instant Web Publishing. The top half of the Instant Web Publishing dialog relates to the status of IWP at the application level; the bottom half details the sharing status of any currently open database files. The two halves function independently of one another and are discussed separately here. For now, we're just concerned with getting IWP working at the application level and therefore limit our discussion to the options on the top half of the Instant Web Publishing dialog.

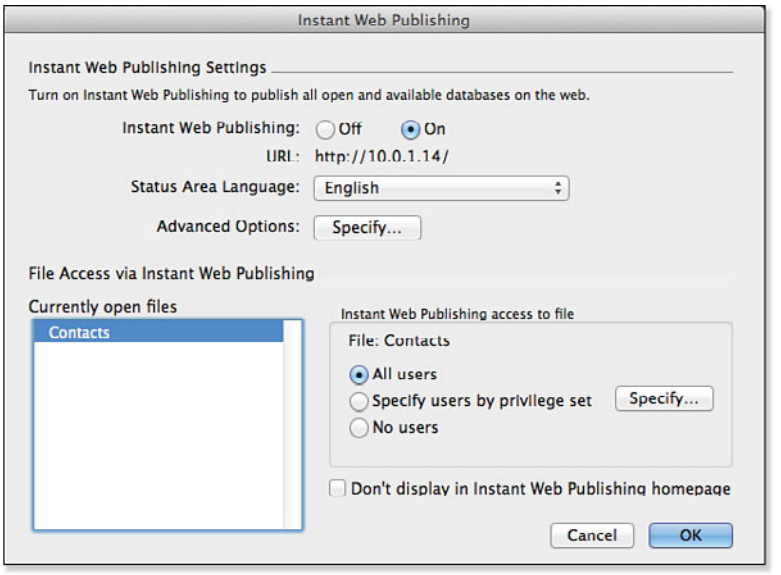


Figure 24.5
To enable Instant Web Publishing in FileMaker Pro, simply select the On option on the IWP configuration screen.

Turning Instant Web Publishing on and off is as simple as toggling the Off/On selection. Selecting On enables this particular copy of FileMaker Pro to act as an IWP host. You can choose the language that will be used on the IWP Database Homepage and in the IWP controls at the left of the window. You can also configure a handful of advanced options, as shown in Figure 24.6.

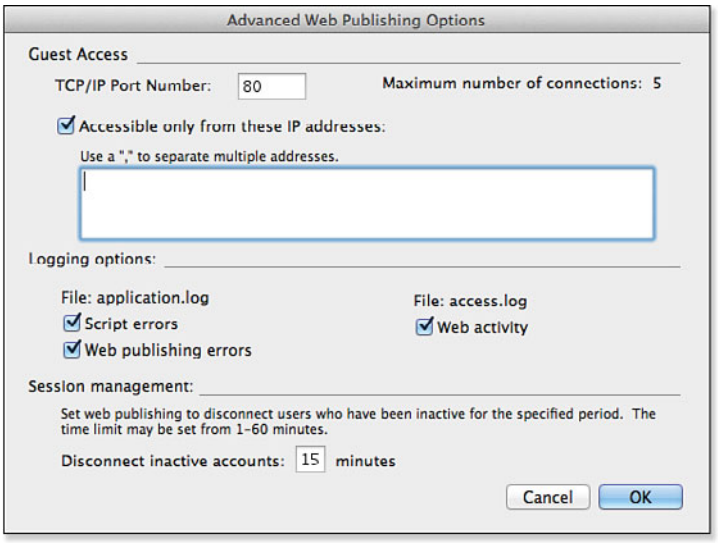


Figure 24.6
On the Advanced Web Publishing Options dialog, you can configure the port number, logging options, IP restrictions, and session disconnect time.

Port Number

By default, IWP is configured to use port 80 on the host machine. If another application, such as a web server, is already using that port, you see an error message and are asked to specify a different port to use. FileMaker, Inc., has registered port 591 with the Internet Assigned Numbers Authority (IANA), so that's the recommended alternative port number. The only downside of using a port other than 80 is that users need to explicitly specify the port as part of the URL to access IWP. For instance, instead of typing **127.0.0.1**, your users would have to type **127.0.0.1:591** (or whatever port number you specified).

note

If you are using OS X, you might be asked to type your computer's passphrase if you attempt to change the port number when configuring IWP within the FileMaker client.

Security

If you know the IP addresses of the machines your IWP users will use when accessing your solution, you can greatly increase your solution's security by restricting access to only those addresses. Multiple IP addresses can be entered as a comma-separated list. You can use an asterisk (*) as a wildcard in place of any part of the IP address (except for the first part). That is, entering **192.168.101.*** causes any IP address from 192.168.101.0 to 192.168.101.255 to be accepted. Entering **192.*** allows access to any user whose IP address begins with 192.

If you don't set IP restrictions, anyone in the world who knows the IP address of your host machine and has network access to it can see at least the IWP Database Homepage (which lists IWP-enabled files). And if you've enabled the Instant Web Publishing extended privilege on the Guest privilege set, remote users could open the files as well. This is, of course, exactly the behavior you would want when IWP is used as part of a publicly accessible website.

Logging

You can enable two activity logs for tracking and monitoring your IWP solution: the application log and the access log. The application log tracks script errors and web publishing errors:

- **Script errors**—These errors occur when a web user runs a script that contains non–web-compatible script steps. See the section "Scripting for IWP," later in this chapter, for more information about what particular steps are not web compatible. A script error can also occur if a user attempts to do something (via a script) that's not permitted by that user's privilege set. Logging script errors—especially as you're testing an existing solution for IWP friendliness—is a great way to troubleshoot potential problems.
- **Web publishing errors**—These errors include more generic errors, such as "page not found" errors. The log entry generated by one of these generic errors is very sparse and might not be terribly helpful for troubleshooting purposes.

The access log records all IWP activity at a granular level: Every hit is recorded, just as you'd find with any web server. As a result, the access log can grow quite large very quickly, and there are no mechanisms that allow for automatic purging of the logs. Be sure to check the size of the logs periodically and to prune them as necessary to keep them from eating up disk space. (A knowledge-

able system administrator can configure both Windows and OS X to periodically trim or rotate logs to prevent uncontrolled log growth.)

Ending a Session

The final option on the Advanced Web Publishing Options dialog is the setting for the session disconnect time. As mentioned previously, IWP establishes a unique database session for each web user. This means that as a user interacts with the system, things such as global values, the current layout, and the active found set are remembered. Because only five sessions can be active at any given time when using FileMaker Pro as an IWP host, it's important that sessions be ended at some point. A session can be ended in several ways:

- A user can click the Log Out button in the IWP controls.
- The `Exit Application` script step ends an IWP session and returns the user to the Database Homepage.
- You can automatically terminate a session after a certain amount of inactivity. The default is 15 minutes, but you can set it to anything from 1 to 60 minutes.

note

Each of the two logs can be read with any text editor, but you might find it helpful to build a FileMaker database into which you can import log data. It will be much easier to read and search that way.

tip

Clicking the home icon in the IWP controls to return to the Database Homepage does not end a session. If a user reenters the file from the Database Homepage without ending his session, he returns to exactly the same place he left, even if a startup script or default layout is specified for the file.

*Are your IWP sessions not ending when you think they should? **See** "Problems Ending IWP Sessions" in the "Troubleshooting" section at the end of this chapter.*

Configuring FileMaker Server Advanced for IWP

One of the best features of the FileMaker product line is the capability to do web publishing directly from files hosted by FileMaker Server Advanced. Using FileMaker Pro as an IWP host works well for development, testing, and some limited deployment situations, but for many business applications, you'll find that you want the added power and stability that come from using FileMaker Server Advanced for this purpose.

Using FileMaker Server Advanced as your IWP host provides several significant benefits. The first is simply that it scales better. With FileMaker Pro, you are limited to five concurrent IWP sessions; with FileMaker Server Advanced, you can have up to 100 IWP sessions. FileMaker Server Advanced can also host up to 125 files, compared to FileMaker Pro's ten.

Even more important, you have the option to use SSL for data encryption when using FileMaker Server Advanced as the web host. FileMaker Server Advanced is a more reliable web host as well. It is more likely that the shared files will always be available for web users, that they'll be backed up on a regular basis, and that the site's IP address won't change when you use FileMaker Server.

note

Even in organizations that use dynamic addressing for desktop machines, servers are typically assigned static IP addresses.

Chapter 27, "FileMaker Server and Server Advanced," covers in detail the various components and installation options of FileMaker Server and the Web Publishing Engine. Here, we assume that you have all the required components in place and merely touch on the relevant configuration screens in the FileMaker Server Admin Console.

FileMaker Server Admin Console is a Java configuration tool that allows you to attach a Web Publishing Engine to a FileMaker Server and configure it. As shown in Figure 24.7, you turn on Instant Web Publishing for FileMaker Server simply by checking the box on the Instant Web Publishing pane of the Web Publishing screen.

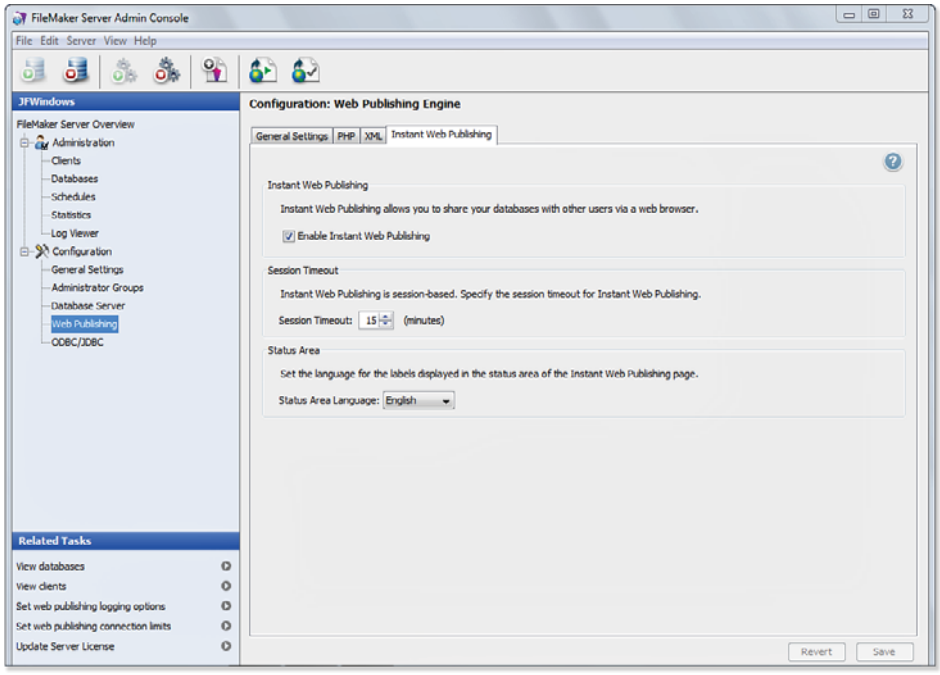


Figure 24.7 Use the FileMaker Server Admin Console to allow FileMaker Server Advanced to host IWP-enabled databases.

You can see a list of the databases accessible via IWP on the server by going to the Databases page, shown in Figure 24.8. For a database to be "IWP accessible," one or more privilege sets needs to have the `fmiwp` extended privilege enabled in the database itself, as described in the following section. There's no configuration or setup that you need to do in FileMaker Server Admin Console, or to the files themselves, before hosting them with FileMaker Server Advanced. In fact, even while a file is being hosted by FileMaker Server, a user with the privilege to manage extended privileges can use FileMaker Pro to open the file remotely and edit the privilege sets so that the file is or isn't IWP accessible.

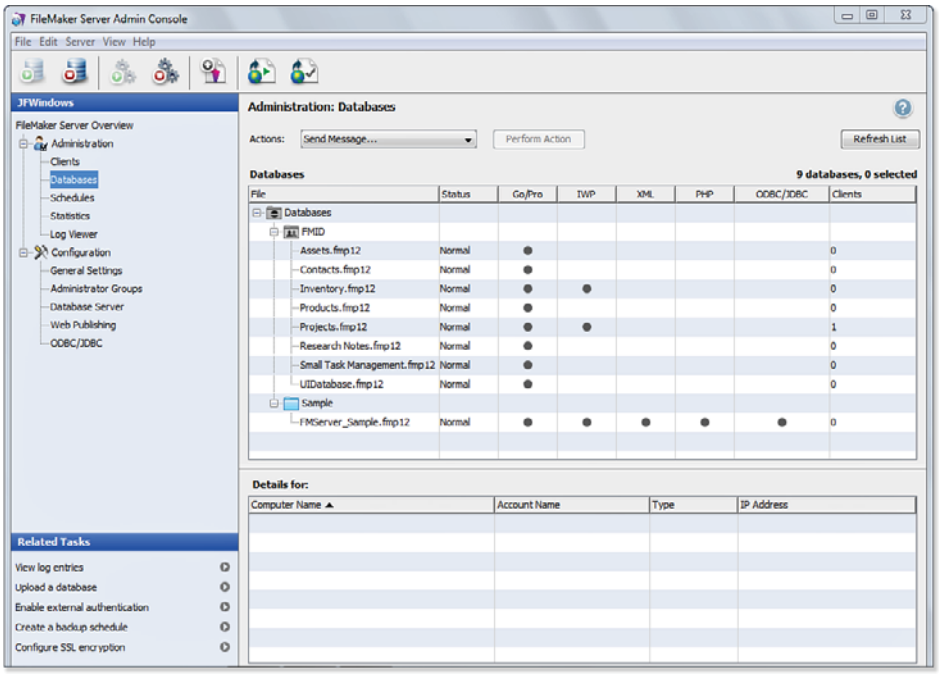


Figure 24.8 FileMaker Server Admin Console lists all the web-accessible databases on the server, but you don't need to do any configuration here at the file level to allow something to be shared to IWP.

note

If you want a file to be accessible via IWP but not to show up on the Database Homepage, you have to open the file with FileMaker Pro (open it directly, that is, not simply as a client of FileMaker Server) and go into the Instant Web Publishing configuration screen. After you are there, select the file and then check the Don't Display in Instant Web Publishing Homepage check box. You do not need to actually enable IWP or add any extended privileges to privilege sets to have access to this setting.

Sharing and Securing Files via IWP

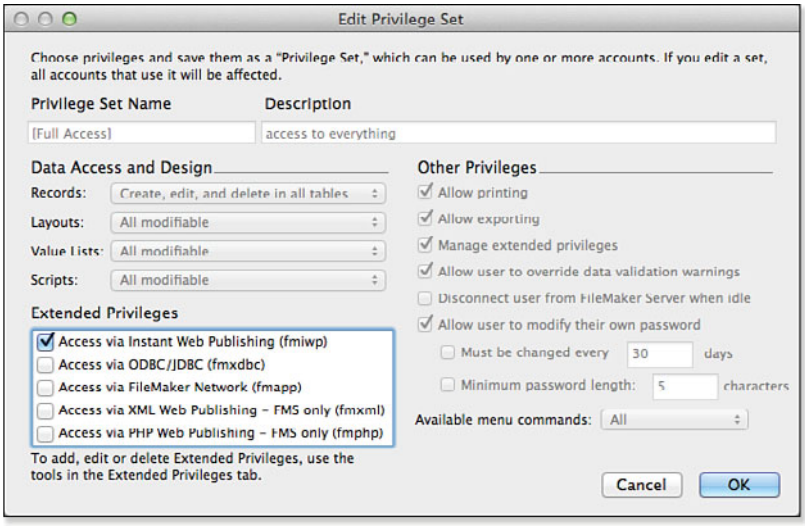
Security for Instant Web Publishing users is managed the same way it's managed for FileMaker Pro users: via accounts and privileges. Accounts and privileges also dictate which database files are accessible via IWP. To be shared via IWP, a particular file must be open, and one or more privilege sets in that file must have the fmiwp extended privilege enabled. This is true regardless of whether you plan to use FileMaker Pro or FileMaker Server Advanced as the web host. You assign the `fmiwp` extended privilege to a privilege set in any of three ways:

- Go to File, Manage, Security. On the Extended Privileges tab, you'll see a list of the various extended privileges and be able to assign fmiwp to any privilege sets you want.

For more information on what extended privileges are and how to assign them to a privilege set, see "Extended Privileges," p. 354.

- Also in File, Manage, Accounts & Privileges, on the Privilege Sets tab, you can select a privilege set to edit from the list of privilege sets. Then, you can select `fmiwp` as an extended privilege for the currently active privilege set, as shown at the lower left in Figure 24.9.

Figure 24.9
Make certain that `fmiwp` is an extended privilege for the relevant privilege sets.



- On the Instant Web Publishing setup screen (refer to Figure 24.5), the bottom half of the screen shows a list of open database files. When you select a particular database, you can manage the `fmiwp` extended privilege right from this screen. If you select All Users or No Users, the `fmiwp` extended privilege is granted or removed from all privilege sets in the file. You can also select Specify Users by Privilege Set to select those privilege sets that should have access to IWP. Although the words *extended privilege* and *fmiwp* never appear on this screen, it functions exactly the same as the Extended Privilege detail screen. This screen is intended to be more user friendly and convenient, especially when you are working with multiple files.

note

To assign extended privileges in any of these ways, a user must be logged in with a password that grants rights to Manage Extended Privileges.

note

Any changes made in the sharing settings and privileges for a file take effect immediately; you do not need to restart FileMaker or close the file.

The other sharing option you can configure on the Instant Web Publishing setup screen is whether the database name appears on the Database Homepage. In a multifile solution, you might want to have only a single file appear so that users are forced to enter the system through a single, controlled point of entry.

Users can get to your IWP site from a browser by typing **<IP address or domain name>/fmi/iwp/res/iwp_home.html**. When they do this, the first thing they'll see is the IWP Database Homepage,

an example of which is shown in Figure 24.10. The Database Homepage lists, in alphabetical order, all files on the host machine that have at least some privilege sets with the `fmiwp` extended privilege enabled. The Database Homepage cannot be suppressed, although it can be customized or replaced, as explained later in this chapter.

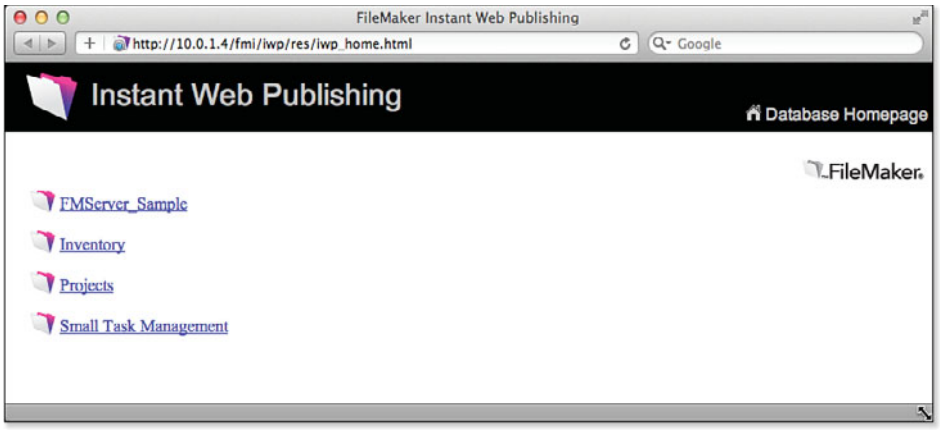


Figure 24.10 The Database Homepage provides users with a list of accessible files.

Users aren't prompted for a password on their way to the Database Homepage. The password prompt occurs (unless they are logged in as guests, as described in the following bulleted list) when users first try to interact with a database. IWP now uses an HTML forms-based interface for entering a username and password, as shown in Figure 24.11. To be authenticated, users must enter an active, valid username and password, and their accounts must be associated with a privilege set that has the fmiwp extended privilege enabled.

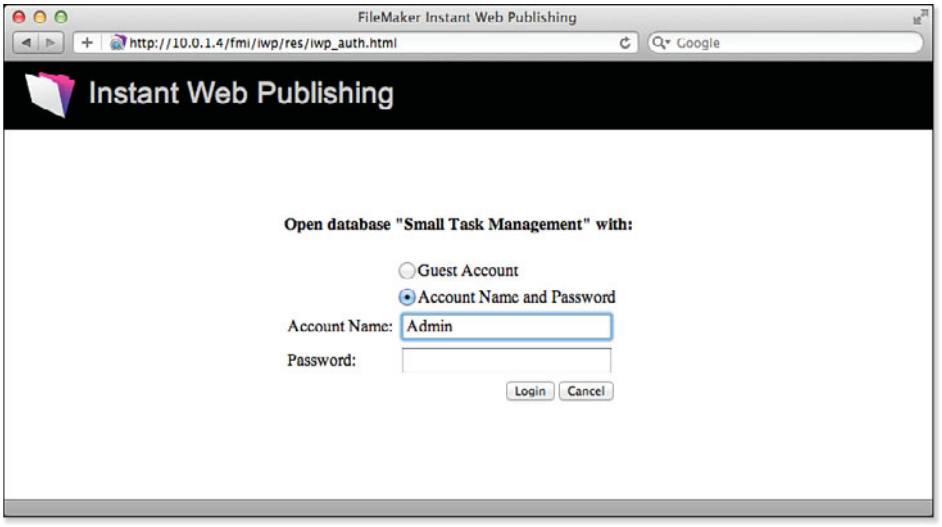


Figure 24.11 You need to log in to the database to access it.

You should know a number of things about how accounts and privileges are authenticated under IWP:

- As in regular FileMaker authentication, the password is case sensitive (although the account name is not).
- IWP ignores any default login account information that has been set up under File Options.
- IWP does not support the Account option to require users to change their passwords after the next successful login. Changing passwords is not a feature supported by IWP. If this option has been set, a web user who tries to log in with that username and password receives Error 211 (Password Has Expired) and cannot enter the system.
- If the Guest account has been activated and given the `fmiwp` extended privilege, users might not be prompted for a username/password to access the database. For the users to skip the login screen, though, it's necessary that the `fmiwp` extended privilege be assigned only to the [Read-Only Access] privilege set (the privilege set used by the Guest account). Anyone automatically logged in in this fashion will have the privileges of the Guest account. Such a configuration would typically be used only for websites that need to be accessed by the general public.

After a user is authenticated as a valid user of the file, that user's privilege set then controls which actions can be performed, just as it does for users of the FileMaker Pro desktop application. Field and layout restrictions, record-level access, creation and deletion of records—all of these are managed exactly the same for IWP users as for FileMaker Pro users. The capability to make use of this unified security model is truly one of the best features of FileMaker IWP and makes it much simpler to deploy robust and secure IWP solutions.

tip

Remember that many solutions leave the security settings unchanged, which means that the user is Admin and the password is blank. When coupled with auto-login in File, File Options, this can mean that you never have to use an ID and password. When it comes to IWP, however, you will need to do so. If you don't know the ID and password, try the defaults. And, remember that because it's so simple to get into an unprotected database, you should make certain that your databases do use good user IDs and passwords.

tip

You can create a script that uses the account management script steps to create your own customized login routine. Users would use Guest privileges to get to your login screen, and then your script would use the `Re-login` step to reauthenticate them as different users.

*For more information about setting up user accounts and privileges, **see** Chapter 12, "Implementing Security."*

Designing for IWP Deployment

The preceding sections discussed how to enable IWP at the application level and how to set a file so that users can access it via IWP. Although this is enough for IWP to function, there are usability issues to consider as well. Not all layouts and scripts translate well to the Web, and some FileMaker features simply don't work via IWP. The following sections discuss the constraints that you, as a

developer, must be aware of when deploying an IWP solution. We also discuss a number of development techniques that can make an IWP solution feel more like a typical web application.

Constraints of IWP

Most of the core functionality of FileMaker Pro is available to IWP users. This includes being able to view layouts, find and edit data, and perform scripts attached to buttons. However, a number of FileMaker features are not available to IWP users. It's important to keep these points in mind, especially when trying to port an already existing solution to the Web:

- As noted previously, only the Classic theme is available in IWP. Other themes will not open, so you might have to move layouts back to Classic before deploying them in IWP.
- IWP users have no database development tools. This means IWP users can't create new files; define tables, fields, and relationships; alter layouts; manage user privileges; or edit scripts.
- IWP users can't use any of the FileMaker Pro keyboard shortcuts. Be sure that you leave the IWP controls visible or provide your users with ample scripted routines for tasks such as executing finds and committing records.
- There is no capability to import or export data from an IWP session. In general, any action that interacts with another application, the file system, or the operating system is not possible via IWP.
- IWP has no Preview mode. This means that sliding and multicolumn layouts, all of which require being in Preview mode to view, are not available to IWP users. Similarly, printing is not supported. IWP users can choose to print the contents of the browser window as they would any other web page, but the results will not be the same as printing from FileMaker Pro. (That is, headers and footers won't appear on each page, page setups will not be honored, and so on.)
- There are a few data-entry differences for IWP users. For instance, web users can't edit rich text formatting in fields. That is, they can't change the style, font, or size of text in a field. They can generally, however, see rich text formatting that has already been applied to a field.
- Most window manipulation tools and techniques do not translate well to IWP. The user's browser can show only the contents of the currently active window in the virtual FileMaker environment. That environment can maintain multiple virtual windows and switch between them, but a user can't have multiple visible windows in the browser, and cannot resize or move windows except to the extent allowed by the browser. In other words, the users can manually resize their browser windows, but precision movement and placement of windows using script steps such as `Move Window` are not supported in IWP.
- None of the FileMaker Pro toolbars from the Status toolbar is available via IWP. IWP does offer its own toolbars in the IWP controls, however, and they contain some of the same functionality found in the FileMaker Pro toolbars.
- Spell-checking is not available via IWP.

- Many graphical layout elements are rendered differently, or not at all, on the Web. This includes diagonal lines, rounded rectangles, rotated objects, ovals, and fill patterns. The sections that follow discuss this topic in greater detail.
- IWP users can't edit value lists through a web browser.
- There is no built-in way for users to change their passwords via IWP, even if they have the privilege to do so. If you need this sort of functionality, you have to use the account management script steps and come up with your own scripted routine.

Scripting for IWP

IWP supports more than 70 script steps, and scripts can be of any length and complexity. Also, because IWP is session based, scripts executed from the Web operate within what might be thought of as a virtual FileMaker environment. This means that changes to the environment (active layout, found set, and so on) are persistent and affect the browser experience, which is a good thing.

Even though IWP script support has come a long way, there are still some behaviors, constraints, and techniques you should be aware of.

Unsupported Script Steps

ScriptMaker itself has an option that makes identifying unsupported script steps quite easy. When you use the Show Compatibility pop-up menu, all the unsupported script steps are dimmed. This affects both the list of script steps and the steps in whatever script you're viewing. Figure 24.12 shows an example of what script step dimming looks like. The Show Compatibility pop-up menu has no effect other than showing you which steps are not supported; how you choose to use that information is up to you (although unsupported script steps are dimmed out, you can still add them to a script). Additionally, its status is not tied to any particular script. That is, it is either turned on or off for the entire file, and it remains that way until a developer changes it. We point this out explicitly because the check box right next to it, Run Script with Full Access Privileges, is a script-specific setting.

Additionally, the option to perform with a dialog is not supported in a number of supported script steps. They include Delete Record/Request, Replace Field Contents, Omit Multiple Records, and Sort Records. These steps are always performed without a dialog via IWP, regardless of which dialog option has been selected in ScriptMaker.

Error Capture

The outcome of running a script (from the Web) that contains unsupported script steps depends on whether the Allow User Abort setting has been turned on or off. If it's not explicitly specified, a script executes on the Web as if Allow User Abort had been turned on. So, not specifying any setting is the same as explicitly turning it on.

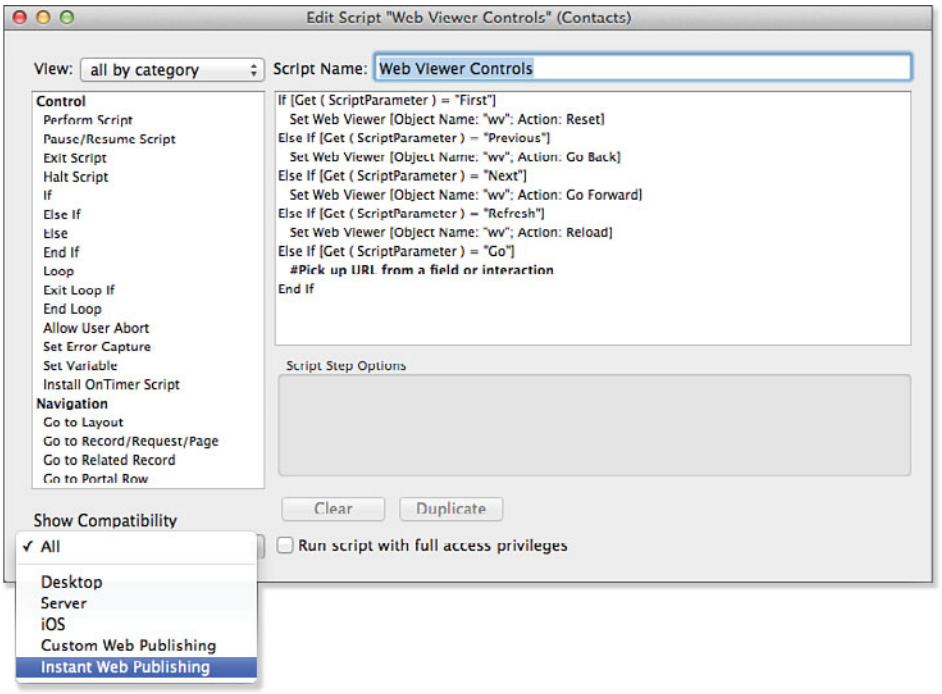


Figure 24.12 When you are writing scripts that will be used via IWP, turn on the Indicate Web Compatibility check box to dim out incompatible script steps.

If user abort is on (or not set at all), script execution halts when an unsupported step is encountered. Steps before the offending script step are performed as normal. If you've chosen to log script errors, the offending step is logged as an error in the application log. The user does not see any error message or have any knowledge that anything is amiss.

If user abort has been turned off, a script simply bypasses any unsupported scripts and attempts to perform subsequent steps. It's performed as if the offending step were simply not there. No error is logged to the application log when this occurs.

note

Script steps with the unsupported "perform with dialog" options discussed earlier are not affected at all by the error capture setting. These script steps will always be performed as if Perform Without Dialog had been checked, regardless of error capture.

Committing Records

If a script run via IWP causes a record to be altered in any way (such as using a `Set Field` script step), be sure that you explicitly save the change by using the `Commit Record/Request` step sometime before the end of the script. If you don't, your web user will be left in Edit mode and, provided that the IWP controls are visible, will have the option to submit or cancel the changes, which is likely not an option you want to offer at that point. Canceling would undo any changes made by the script.

Startup and Shutdown Scripts

If you have specified a startup script for a file, it is performed for IWP users when the session is initiated. Similarly, IWP also switches to a particular layout on startup if you've selected that option. The shutdown script is performed when the user logs out, even if the logout is the result of timing out.

caution

The startup script executes only once per session, when the user navigates there from the Database Homepage or follows an equivalent link from another web page. The startup script is not run if a file is activated through the performance of an external script.

Performing Subscripts in Other Files

A script can call a subscript in another file, but that file has to be open and enabled for IWP for the subscript to execute. Calling a subscript does not force open an external file, as happens in the FileMaker Pro desktop application.

If your subscript activates a window in the external file, the IWP user sees that window in the browser. Unless you provide navigation back to the first file, a user has no way of returning, except by logging out and logging back in. You should make sure that any record changes are fully committed before the user navigates to a window in another file. It's possible that the record will remain in an uncommitted, locked state, even though the IWP user has no idea this has occurred.

Testing for IWP Execution Within a Script

If you have a solution that will be accessed by both FileMaker Pro desktop users and IWP users, chances are they'll use some of the same scripts. If those scripts contain unsupported script steps, you might want to add conditional logic to them so that they behave differently for IWP users than they do for FileMaker users. You can do this by using the `Get (ApplicationVersion)` function. If the words *Web Publishing* are found within the string returned by this function, it means the person executing the script is a web user. It's not possible to discriminate between an IWP user and a CWP user with this function; you simply know you have a web user. The actual syntax for performing the test is as follows:

```
PatternCount (Get (ApplicationVersion); "Web Publishing")
```

Layout Design

Most layouts you design in FileMaker Pro will be rendered almost perfectly in a web browser via Instant Web Publishing. IWP does this by using the absolute positioning capability of Cascading Style Sheets, Level 2. The CSS requirements of IWP are the reason there are browser restrictions for its use. We've already mentioned a few layout elements that don't translate well to IWP—we'll recap them here as well—but there are several additional points to keep in mind when you are creating or modifying layouts for IWP use.

"View As" Options

Web users have the same ability that FileMaker desktop users have to switch between View As Form, View As List, and View As Table on a given layout, unless you restrict that ability at the layout level. To do so, go into the Layout Setup options, shown in Figure 24.13, and simply uncheck any inappropriate views. The additional Table view options that can be specified all translate well to IWP, except for resizable and reorderable columns.

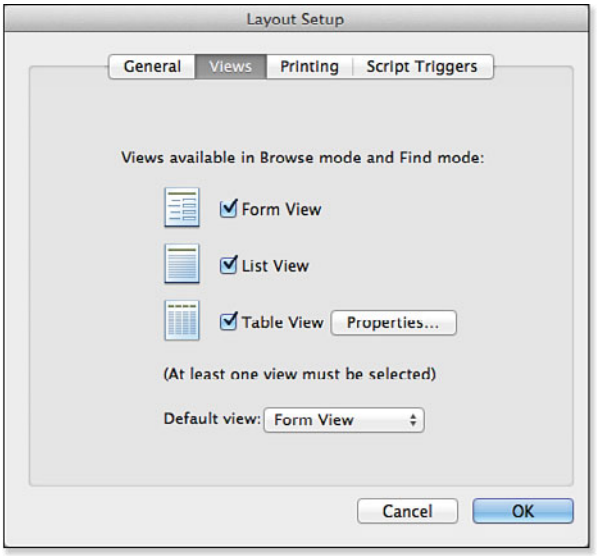


Figure 24.13
In Layout Setup, you can specify the types of views to which a user should be able to switch for a given layout.

You should be aware of a few special characteristics of List and Table views in IWP. By default, View As List shows a set of at most 25 records, and View As Table shows a set of at most 50 records. You cannot change these settings. Also, while in List or Table view, whenever a user clicks a record to edit it, the active record jumps to the top of the set. This response can be slightly disconcerting for users habituated to working with lists of records in FileMaker. For instance, if a user is viewing records 6–10 of a set as a list, and clicks record 8, record 8 jumps to the top, and the screen then displays records 8–12.

Layout Parts

IWP can render any and all parts that compose a layout. There are a few differences, however, between how and when parts display in IWP and how and when they display in the FileMaker desktop application.

First of all, in Form view, the vertical size of a part displayed via IWP is the size that the part was defined to be. It doesn't stretch to fill the vertical space. This is different from how FileMaker Pro behaves. In FileMaker Pro, the last visible part expands to fill any remaining vertical space. Say, for instance, that you have a layout that consists of only a single, colored body part. Via IWP, if a

user resizes a browser window so that it's larger (vertically) than the body part, the space between the bottom of the part and the bottom of the browser is a white void. This also means that if your layout has a footer part, it won't necessarily (indeed, won't likely) be displayed at the bottom of the browser window.

View As List in a browser also has some differences from its FileMaker Pro counterpart. In FileMaker Pro, a header or footer part is locked on the screen at the top or bottom. The area in between displays as many body records as space permits. In FileMaker Pro, leading and trailing grand summary parts display in List view, but title header, title footer, and subsummary parts do not (in Browse mode).

As we've mentioned, in a browser, List view always contains 25 records (except, of course, when the found set is fewer than five or if the active record is one of the last four of the found set). The header and footer are not fixed elements as they are in FileMaker Pro. If the 24 records of the list take up less than the full browser window, the footer simply shows up in the middle of the screen; if they take up more than the full window, a user would need to scroll to see the footer. Another major difference is that title header, title footer, and subsummary parts are all visible in the browser at all times (in List view).

Container Fields

You should know about a few special restrictions and considerations when using container fields in an IWP solution. Most important, there is no capability to add or edit data in a container field via IWP; these fields are strictly view-only. The capability to enter and update pictures, sounds, QuickTime movies, files, and objects is available only to regular FileMaker Pro users.

The visibility and/or accessibility of a container field's contents are dependent on the types of objects they are and how they were entered into the container field in the first place:

- Graphic images that have been directly stored in a container field (that is, not stored as a reference) are visible through a browser. Images should be stored as pictures, not as files.
- Graphic images that have been stored as a reference are visible to IWP users only if the images are stored in the web folder of the FileMaker Pro application (if FileMaker Pro is the IWP host) or if they are stored in the root folder of the web server (if FileMaker Server is the IWP host).
- QuickTime movies can't be accessed directly from the web browser. If you insert them as files rather than as QuickTime, however, a user can play or download them.
- Files stored directly in a container field render to an IWP user as a hyperlink. Clicking the link begins a download of the file. No icon or other graphic representation of the file is visible to web users.
- Sounds that have been stored directly in container fields cannot be played via IWP.

Application Flow

We've discussed many of the technical limitations and details of how various FileMaker features translate to the Web. We turn now to more practical development matters. Certain routines and development habits that work well in the FileMaker desktop application don't work as well from a

web browser. The following sections discuss how the constraints of IWP will influence how you develop solutions.

If you're designing a new solution and you know that you'll have IWP users, you might consider thinking about how you would develop the solution if it were a web application. Because there are more constraints placed on designing for the browser, anything you build for the browser should work well for FileMaker users also.

Explicit Record Commits

HTTP—the underlying protocol of the Web—is a stateless protocol. This means that every request a browser makes to a web server is separate and independent from every other request. Put differently, the web server doesn't maintain a persistent connection to the web client. After it has processed a request from someone's browser, it simply stands by waiting for the next request to come in. To make HTTP connections appear to be persistent, web programmers need to add information to each request from a single client and then let some piece of web server middleware keep track of which client is which, based on this extra request data. This technique is referred to as *session management*.

The client/server connection between FileMaker Pro and FileMaker Server is persistent. The two are constantly talking back and forth, exchanging information and making sure that the other is still there. FileMaker Server is actively aware of all the client sessions. When FileMaker Server receives new record data from any client on the network, it immediately broadcasts that information to all the other clients. And when a user clicks into a field and starts editing data, FileMaker Server immediately knows to consider that record as locked and to prevent other users from modifying the record.

The fact that IWP is now capable of performing session management means that FileMaker maintains information about what's happening on the Web in a virtual FileMaker environment. Even though this doesn't change the fact that HTTP is stateless, using sessions gives IWP a semblance of persistence. Essentially, the server stores a bunch of information about each IWP user; each request from a user includes certain session identifiers that enable the server to recognize the IWP guest and to know the context by which to evaluate the request. One of the benefits of this session model is that IWP users can lock records, and they are notified if they try to edit a record that a regular FileMaker Pro user has locked.

Still, the statelessness of the Web makes the application flow for something even as basic as editing a record much different in IWP than it is in FileMaker Pro. In FileMaker Pro, of course, a user just clicks into a field, makes some changes, and then clicks out of the field to commit (save) the change. On the Web, editing a record involves two distinct transactions. First, by clicking an editable field or using the Edit Record button in the IWP controls, the user generates a request to the server to return an edit form for that record and to mark the record as locked. As we discussed earlier in this chapter, Edit mode in the browser is distinctly different from Browse mode.

The second transaction occurs when the user clicks the Commit button in the IWP controls (or clicks a similar button you've provided for this purpose). No actual data is modified in the database until and unless the record is committed explicitly.

This transaction model for data entry might feel alien to users who are accustomed to working with a FileMaker Pro interface. As you evaluate the web-friendliness of existing layouts or build new

layouts for IWP users, try to make the application flow work well as a series of discrete and independent transactions. One common way to do this is by having tightly controlled routines that users follow to accomplish certain tasks. For instance, instead of letting users just create new records anywhere they want, create a "new record" routine that walks users through a series of screens where they enter data and are required to click a Next Screen or Submit button to move forward through the routine.

Hiding the IWP Controls

As when designing a solution for FileMaker users, you have the option to leave the IWP controls visible for your IWP users or to hide these controls from them. And as with regular FileMaker, unless you lock it open or closed, users can toggle it themselves.

By default, the IWP controls are visible for your IWP users. The script step `Show/Hide Status Area` enables you to programmatically control the visibility of the IWP controls. Typically, if you want to hide the IWP controls, you do so as part of a startup script.

There are certainly benefits to having the IWP controls visible. Most important, the IWP controls provide a wealth of functionality for the IWP user. Navigation, complex searching, and a host of record manipulation tools are all features that come at no charge in the IWP controls.

There are also reasons that developers want to hide the IWP controls from users. The first is simply to constrain users' activities by forcing them to use just the tools you give them. This is generally why developers hide the Status toolbar for FileMaker desktop deployments as well. Hiding the IWP controls also makes your application more web-like. If you are using IWP alongside an existing website or plan to have the general public access your site, you'll probably want to hide the IWP controls. Public users are more likely to expect a web experience than a FileMaker experience.

note

The script step `Show/Hide Status Area` shows or hides the status area in pre–FileMaker Pro 10 software and shows or hides the Status toolbar in FileMaker Pro 10 and later. The name of the script step was kept the same for version compatibility.

tip

If you do decide to hide the IWP controls, you must provide buttons in your interface for every user action you want or need to allow, including committing records, submitting Find requests, and logging out of the application. Because users have no keyboard shortcuts—including using the Enter key to submit Find requests and continue paused scripts, for example—and no pull-down menus, you'll probably need even more buttons than you would when designing without the Status toolbar for FileMaker Pro users.

Portals

Instant Web Publishing does an astonishingly good job of displaying portals in a browser, complete with scroll bars, alternating row colors, and the capability to add data through the last line of a portal (providing, of course, that the underlying relationship allows it). Another nice feature of portals in IWP is that you can edit multiple portal rows at once and submit them together as a batch.

When you are designing an IWP application that requires displaying search results as a list, consider whether you can use a portal instead of a List view. A portal gives you flexibility as far as the number of records that display, and you can use the space to the left or right of it for other purposes.

tip

The best way we've found to make a portal display an ad hoc set of records, such as those returned by a user search, is to place all the record keys of the found set into a return-delimited global text field by using the Copy All Records script step and then to establish a relationship between that field and the file's primary key.

Because you can let the portal scroll, you don't strictly need to create Next and Previous links, but it would make your application more web-like if you did. One option to do this is to take the return-delimited list of record IDs and extract the subset that corresponds to a given page worth of IDs. The `MiddleValues` function comes in handy for this task. You would simply need to have a global field that kept track of the current page number. Then the function

```
MiddleValues (gRecordKeys; (gPageNumber-1) * 8 + 1; 8)
```

would return the eight record IDs on that page. Substitute a different number of records per page in place of 8, of course, if you want to have a hitlist with some other number of records on it. The scripts to navigate to the next and previous pages then simply need to set the page number appropriately and refresh the screen.

Creating Links to IWP from Other Web Pages

The IWP Database Homepage provides a convenient access point for entering web-enabled databases. It's possible also to create your own links into a file from a separate HTML page, which is perhaps more desirable for publicly accessible sites. To do this, you simply create a URL link with the following syntax:

```
http://ip address:port number/fmi/iwp/cgi?-db=databasename&-loadframes
```

If you are using FileMaker Pro itself as your IWP host (as opposed to FileMaker Server Advanced), you can place static HTML files and any images that need to be accessible to IWP users in the web folder inside the FileMaker Pro folder. The web folder is considered the root level when FileMaker Pro acts as a web server. If you had, for example, an HTML page called foo.html in the web folder, the URL to access that page would be the following:

```
http://ip address:port number/foo.html
```

If you develop a solution that uses FileMaker Pro as the host and later decide to migrate to FileMaker Server Advanced, you should move the entire contents of the web folder (if you've put any documents or images there) to the root folder of your web server.

Creating a Custom Home Page

You can override the default page with a page of your own devising. The new file must be called iwp_home.html. It can be used when serving files via IWP either from FileMaker Pro (in which case it belongs in the web directory inside the FileMaker Pro application folder) or from FileMaker Server Advanced (in which case it belongs in the FileMaker Server/Web Publishing/iwp folder).

There are several approaches to creating such a file. You could devise your own file from scratch, creating your own look and feel, and populate that file with hard-coded links to specific databases, as described in the preceding section. Or if you want a file that dynamically assembles a list of all available databases, the way the default home page does, you'll want to customize the default page. An example of that default page can be found on the FileMaker Pro product CD.

The default page makes heavy use of JavaScript and in particular of JavaScript DOM function calls, so familiarity with those technologies will be desirable if you want to customize the IWP home page.

note

For the curious, an example of the default page can also be found in the FileMaker application folder: On Windows, it's found in Extensions/Web Support/Resources/iwpres. On OS X, it's found in Extensions/Web Support/FM Web Publishing/Contents/Resources/iwpres. Note that Extensions/Web Support/FM Web Publishing is an OS X package, not a directory, so you'll have to right-click it and select Show Package Contents to drill deeper.

Troubleshooting

Problems Ending IWP Sessions

FileMaker Pro thinks that there are active IWP sessions, but I know that all the users have closed their browsers.

Closing the browser window or quitting the browser application does not end a session, so be sure to train your users to click the Log Out button (or an equivalent button that you provide). One of the problems you could run into is that an IWP user might quit his browser but still have a record lock. Until the session times out, no other user can modify that record. If you experience this problem, try reducing the session timeout setting to something like 5 minutes.

This page intentionally left blank

25

CUSTOM WEB PUBLISHING WITH PHP AND XML

About Custom Web Publishing

Custom Web Publishing (CWP) is one of two technologies you can use to dynamically publish your FileMaker data on the World Wide Web. Custom Web Publishing comes in two variations: One lets you work with XML, and the other lets you work with PHP. Both are described in this chapter.

In addition to Custom Web Publishing, Instant Web Publishing, which you can read about in Chapter 24, "Instant Web Publishing," also lets you publish your database to the Web.

This chapter assumes a basic familiarity with PHP and XML—or just one of them, if that's all you want to use. Additional documentation is installed with FileMaker Server in its Documentation folder: You will find separate documents for XML and PHP publishing. On OS X, FileMaker Server is installed in /Library/FileMaker Server/Documentation rather than in /Applications. On Windows, it is \Program Files\FileMaker\Filemaker Server\Documentation.

note

In FileMaker Pro, it's possible to use Instant Web Publishing (IWP) to publish data from a client copy of FileMaker Pro. IWP can also be used in FileMaker Server Advanced, though, and can support many up to 100 users in that way.

*For more detail about the two different flavors in the FileMaker Server product line, FileMaker Server and FileMaker Server Advanced, **see** Chapter 27, "FileMaker Server and Server Advanced."*

Without web publishing, the only access to your databases will be directly through the FileMaker Database server using FileMaker Pro and FileMaker Go clients. If you have FileMaker Server Advanced, you will be able to enable ODBC/JDBC, which will allow access to your databases from ODBC/JDBC clients other than FileMaker.

Most installations do enable web publishing if only because they allow so many deployment options, and because FileMaker Server supports web publishing except for Instant Web Publishing and ODBC/JDBC, which require FileMaker Server Advanced. When you install FileMaker Server or FileMaker Server Advanced, you deploy it over one, two, or three computers.

FileMaker Server provides the web publishing capabilities for CWP with PHP or XML. For Open Database Connectivity/Java Database Connectivity (ODBC/JDBC) or Instant Web Publishing beyond a single computer, you need FileMaker Server Advanced.

FileMaker Server need not have web publishing enabled. However, if it is not enabled, you will not be able to use Instant Web Publishing or Custom Web Publishing with PHP or XML.

Understanding the Three Parts of FileMaker Web Publishing

Regardless of the number of computers you use, you must know about these three parts of FileMaker web publishing:

- **Web server**—Apache (OS X) or IIS (Windows). During installation, the FileMaker Web Server Module is installed into the web server. In addition, the FileMaker PHP application programming interface may be installed on the web server.
- **Web Publishing Engine (WPE)**—The code that handles web publishing. It includes the Web Publishing Core, which routes requests from the Web Server Module inside the web server to the FileMaker Database Server. It also includes the Custom Web Publishing Engine (CWPE), which routes return information to the web server.
- **FileMaker Database Server**—The basic server to which you connect from FileMaker Pro and FileMaker Go clients.

What is critically important is that the users use web browsers to access the web server just as they normally would to access any web page. The Web Server Module installed by FileMaker routes the requests to the WPE and the FileMaker Database Server.

Custom Web Publishing Versus Instant Web Publishing

If you've read about Instant Web Publishing already in Chapter 24, you'll be aware that the IWP capabilities of FileMaker are quite extensive—so extensive, in fact, that you might wonder whether

IWP would suffice for all your web publishing needs. But CWP has a number of important advantages over IWP. Here are some of the most significant ones:

- With CWP, integrating FileMaker data with other websites or providing FileMaker data to others in the form of a web service is straightforward. CWP makes a strong distinction between the raw data (which is returned as XML) and the final presentational form (which can result from writing PHP code). By contrast, in IWP, data and presentation combine in a way that makes it all but impossible to use the data itself in other contexts.
- CWP is best for sites that need to conform to the conventions of the World Wide Web. IWP presents data in a FileMaker-driven way: It's easy, using IWP, to reproduce a complex FileMaker layout on the Web. But it would be quite difficult to, for example, display a set of search results in a two-column list or break up a large set of search results into multiple results pages—both of which are common presentation styles on the Web.
- IWP has a number of built-in limitations. For example, it cannot reproduce FileMaker's Preview mode. In addition, the IWP list and table views are limited to displaying 25 and 50 records at a time, respectively. CWP can overcome these limits.

In general, IWP is best for making some portion of the functionality of an existing FileMaker *database* accessible to remote users. IWP's chief strength is in bringing the FileMaker *experience* into a web browser. The most likely targets for this technology are remote users of a FileMaker system who might not be able to be in the same building or same site as the server, but require ready access. This is likely to cater to a relatively small group of users—hundreds, say, rather than the thousands and tens of thousands that a public website can reach.

CWP, on the other hand, is best when you want to present FileMaker data in a non-FileMaker style, either as familiar-looking web pages or in some other form. It enables you to make FileMaker data available over the Web as raw XML, to integrate FileMaker data into an existing website, or to build a new website around FileMaker data while preserving all the conventions of web presentation. In addition, the rich set of PHP functions provided by FileMaker let you enhance the user experience tremendously.

Preparing for Custom Web Publishing

CWP requires you to have FileMaker Server or FileMaker Server Advanced installed alongside a web server. Beyond that, you must prepare each database that will use CWP, and you must enable the appropriate CWP technologies on FileMaker Server.

*For more information on FileMaker Server and FileMaker Server Advanced, **see** Chapter 27, "FileMaker Server and Server Advanced."*

Getting Your Databases Ready for CWP

To get your FileMaker databases ready for CWP, you need to do a few specific things with access privileges in each file you want to share via CWP. If you're familiar with previous versions of FileMaker Pro, you'll recall that the various publishing options for a database (web sharing, local ODBC, remote ODBC) were all accessed via a file's sharing options. Enabling or disabling a sharing method was simply a matter of checking or unchecking a sharing option.

Access to a FileMaker database using XML is handled via the security and privilege system. You can allow or deny XML access to a file based on whether a user has the appropriate privilege, as well as control that user's rights and privileges down to the record or field level.

To enable CWP in a file, you must enable the correct extended privileges for each type of CWP access you want to allow. To allow access to data from the file as raw XML, enable the extended privilege with the keyword `fmxml`, as shown in Figure 25.1.

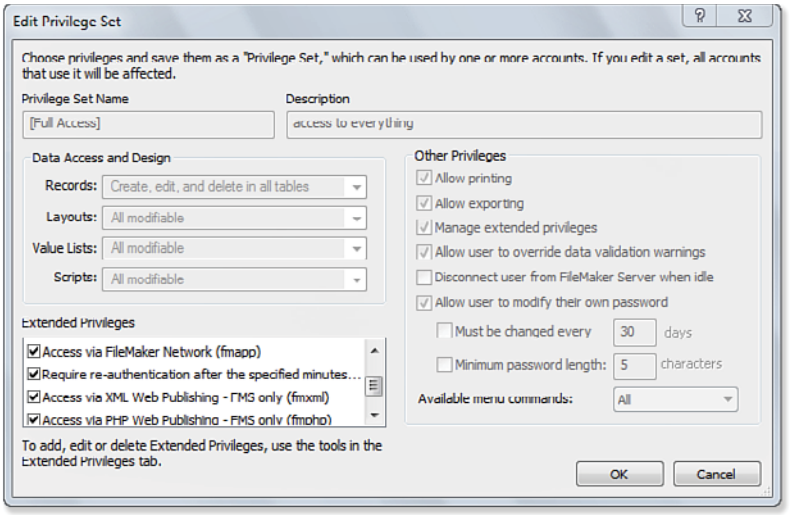


Figure 25.1
You must enable certain extended privileges to enable Custom Web Publishing with XML.

To recap, each database that you want to share via CWP must have the appropriate extended privileges created and added to one or more privilege sets.

If you expect to see a database served via CWP and it doesn't appear, check to make sure that the appropriate extended privileges are enabled. ***See*** *"Getting the Right Privileges" in the "Troubleshooting" section at the end of this chapter.*

Getting FileMaker Server Ready for Custom Web Publishing

You also must configure FileMaker Server for the CWP technologies you will be using. The general settings in Admin Console are shown in Figure 25.2. The section for each CWP technology shows you what the settings are.

tip

In Figure 25.1, one important point to note is that FileMaker network access is enabled. If it is, you can use the Open Remote command in FileMaker Pro to access the database being shared by FileMaker Server. This is an important safety valve for you because it enables you to log on and make changes to accounts and privileges, for example. In general, providing yourself with at least one account that has network access to each shared database is a good idea.

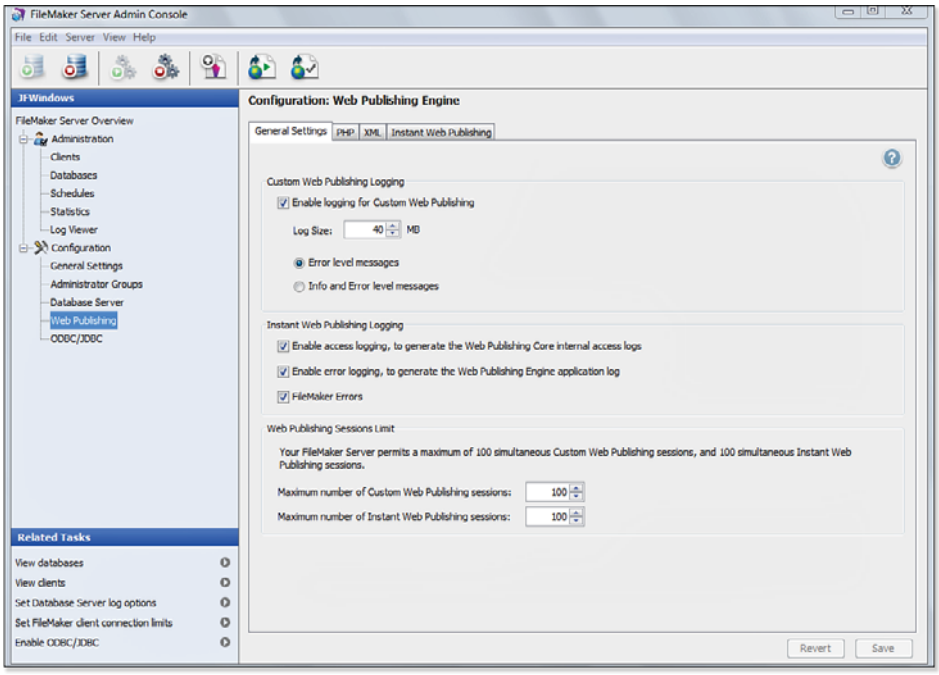


Figure 25.2
Set general CWP settings.

Use the General Settings tab of Web Publishing to adjust the settings. At least for development, you usually do want the logs enabled.

Choosing a Custom Web Publishing Technology

Both XML and PHP are powerful tools for Custom Web Publishing (CWP). Choosing between them depends exactly on what you are trying to do and what your resources are. Perhaps most important, if the developer of your CWP site is more comfortable with PHP than XML, or vice versa, your choice is basically made for you. Let the technology work around the people, not the other way around. Here are some general considerations you might want to contemplate:

- PHP is an object-oriented procedural scripting language. The XML combination is more declarative. Some web developers prefer one style to the other.
- It might be easier to integrate PHP into a site that uses other web technologies.

There are also some considerations you don't have to worry about in making your choice. These apply to both forms of CWP:

- FileMaker handles all security using accounts, privileges, and extended privileges.
- Both XML and PHP are controlled by FileMaker Server. They are also available on FileMaker Server Advanced, but the basic FileMaker Server product can serve up both forms of web publishing.

Using Custom Web Publishing with PHP

CWP requires that you have FileMaker Server or FileMaker Server Advanced installed alongside a web server. In addition to that, you have to prepare each database that will use CWP, and you must enable the appropriate CWP technologies on FileMaker Server.

For more information on FileMaker Server and Server Advanced, **see** *Chapter 27, "FileMaker Server and Server Advanced."*

For more information on Custom Web Publishing with PHP, **see** *the FileMaker documentation installed in the Documentation folder of FileMaker Server.*

Getting Your Databases Ready for Custom Web Publishing with PHP

Access to a FileMaker database via PHP is handled via the security and privilege system. You can allow or deny PHP access to a file based on whether a user has the appropriate privilege, and you can control that user's rights and privileges down to the record or field level.

Setting the Extended Privilege for PHP

To allow access via PHP, enable the extended privilege with the keyword `fmphp`. Figure 25.1 illustrates the use of these extended privileges.

tip

If you expect to see a database served via CWP/PHP and it doesn't appear, check to make sure the appropriate extended privileges are enabled.

Setting Other Security Measures for the Database

In addition to setting the extended privilege for PHP, you can also take a variety of other steps if you choose. Still in the area of security and access, you might want to review the accounts and privileges you have set for the database. Even if you are behind a corporate firewall, you often want to limit web access to your database to the absolute minimum. If you are publishing public data in a totally public environment, it is still worth occasionally reviewing your database to make certain that, in its ongoing maintenance and modification, no confidential data has appeared there.

note

This security issue is particularly relevant to organizations such as schools where the internally public data (class lists, student addresses, and so forth) is generally never shown in public.

Getting FileMaker Server Ready for Custom Web Publishing with PHP

You also must use Admin Console to enable PHP publishing, as shown in Figure 25.3. The other options in this window are described later in this chapter.

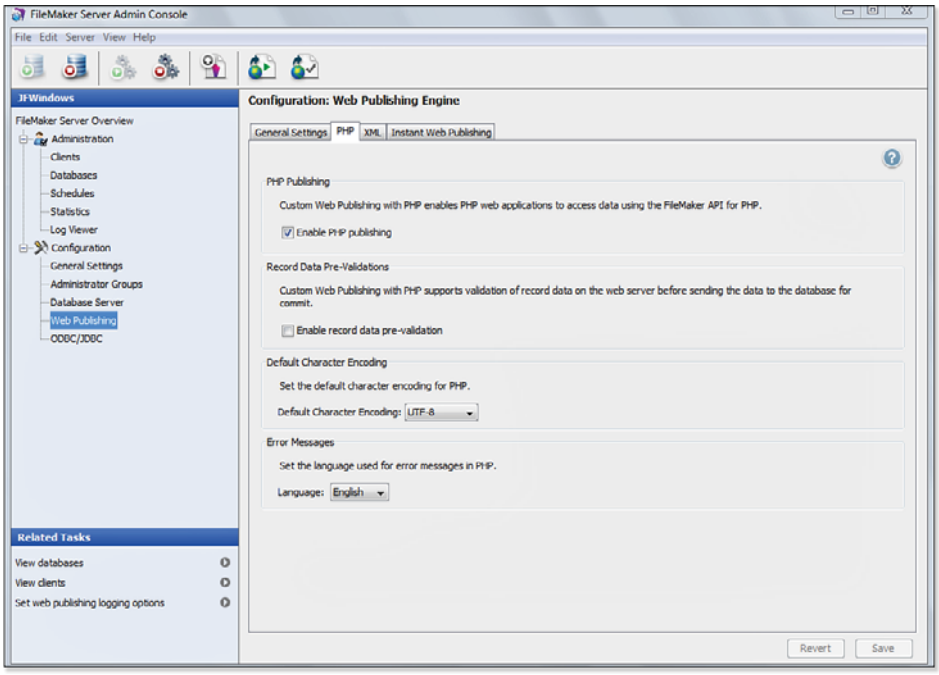


Figure 25.3 Enable PHP publishing in Admin Console.

Placing Files on the Web Server

The two types of files you need to worry about placing on the web server are the PHP files themselves and the files that are referred to by container fields.

Placing the PHP Files

When you are using Custom Web Publishing with PHP, your PHP files live in the normal web publishing folder on the web server. If you are using multiple machines, the Admin Console setup establishes the link between the web server and the machine running the Web Publishing Engine.

Dealing with Container Fields

Beginning with FileMaker 12, it is possible to specify the location of container field data. By default, container files are placed in a folder adjacent to the databases. This means that on FileMaker Pro server, your databases folder (which may contain a subfolder) contains databases and a folder for the container content, as you see in Figure 25.4. On Windows, the databases are in \Program Files\FileMaker\FileMaker Server\Data\Database\<optional subfolder>\. On OS X, the location is /Library/FileMaker Server/Data/Databases/<optional subfolder>/.

Within the optional subfolder, you will find a folder for each database, as shown in Figure 25.5.

*For more on container fields, **see** Chapter 3, "Defining and Working with Fields and Tables."*

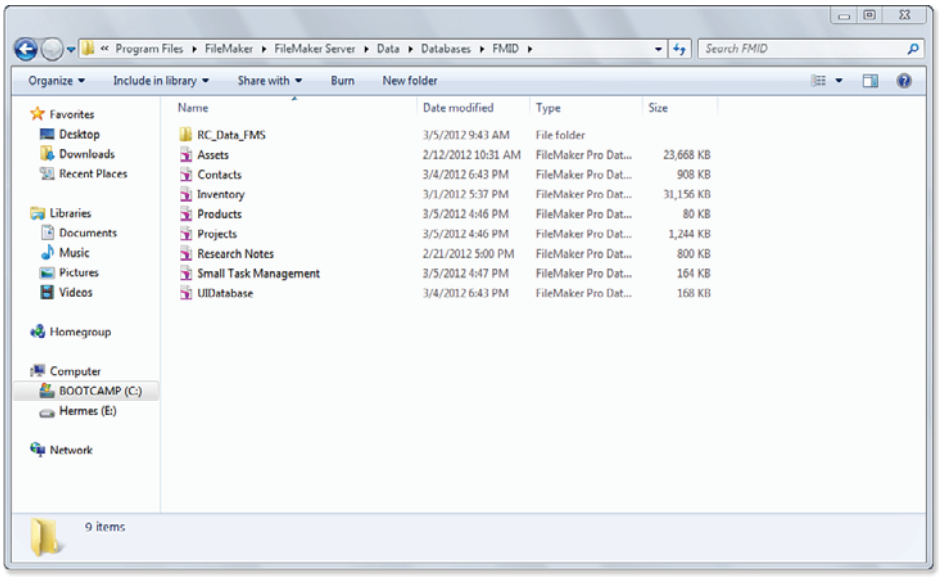


Figure 25.4 Location container folders.

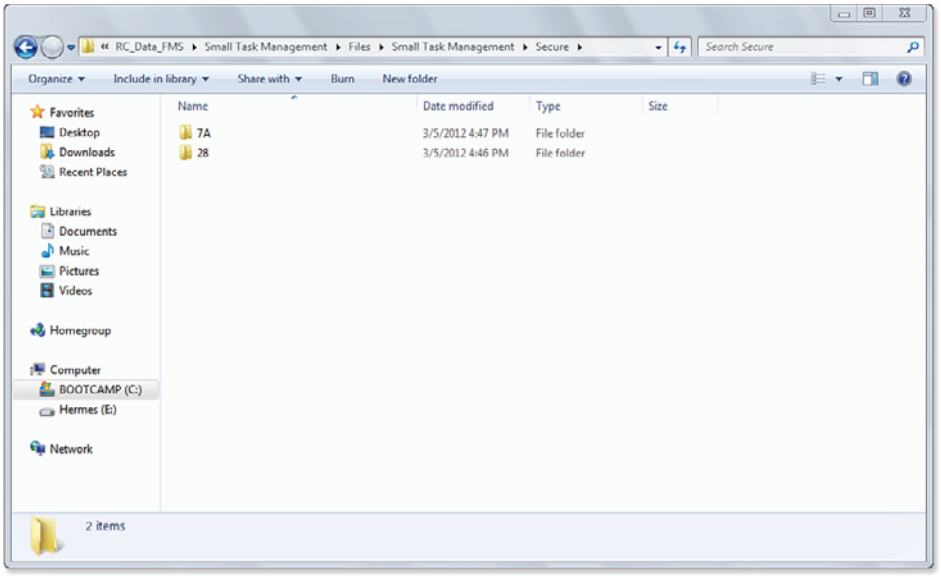


Figure 25.5 Subfolders for each database hold container files.

Writing the PHP code for the FileMaker PHP API

The heart of the FileMaker PHP implementation is the `FileMaker` class. Typically, you include it in each of your FileMaker PHP files, as shown here:

```
require_once ('FileMaker.php');
```

The file is located in Applications/FileMaker API for PHP/ along with examples and further documentation. It is worth reviewing that file as you plan your code: All of the methods are located in that file and they are documented thoroughly.

The structure of FileMaker PHP files is usually the same. The FileMaker API for PHP/examples/API/examples/viewRecord.php file is a good example. The file consists of six sections, which are detailed next. Note that some spacing has been modified for the book's layout.

Typical PHP Headers

The beginning of the file consists of the standard opening code for a PHP file:

```
<html>
<?php
```

Comments and Documentation

Next, comments and documentation of the file are provided (you do provide comments, don't you?):

```
/*
 * viewRecord.php
*
 * Copyright  © 2005-2006, FileMaker, Inc. All rights reserved.
 * NOTE: Use of this source code is subject to the terms of the FileMaker
 * Software License which accompanies the code. Your use of this source code
 * signifies your agreement to such license terms and conditions. Except as
 * expressly granted in the Software License, no other copyright, patent, or
 * other intellectual property license or right is granted, either expressly or
 * by implication, by FileMaker.
*
 * Example PHP script to illustrate how to view a particular record in a
 * database using PHP API.
*
 * Requirements:
 *   1. Working FileMaker Server Advanced installation
 *   2. 'FMPHP_Sample' database hosted in FileMaker Server
*
 */
```

Get the FileMaker Object and Store It

Next, you include `FileMaker.php` and access the FileMaker object. Then you set up some other variables that you might need. Most commonly, you want to get a database.

Here is the standard code to access a database and store it in a variable:

```
// Include FileMaker API
require_once ('FileMaker.php');

// Create a new connection to FMPHP_Sample database.
// Location of FileMaker Server Advanced is assumed to be on the same machine,
//  thus we assume hostspec is api default of 'http://localhost' as specified
//  in filemaker-api.php.
// If FMSA web server is on another machine, specify 'hostspec' as follows:
//   $fm = new FileMaker('FMPHP_Sample', 'http://10.0.0.1');
$fm = new FileMaker('FMPHP_Sample');
```

The last line stores the FileMaker object in a local variable (`$fm` is the name commonly used). Note that there are only two executable lines of code in this snippet:

- `require_once ('FileMaker.php');` is usually the first executable line of code. It is unchanged in your various FileMaker PHP files.
- `$fm = new FileMaker` accesses a database and stores it in a variable. The argument is the name of the database.

Get a Record from the FileMaker Object (`$fm`) as Needed

Store data such as a record in a local variable and close the PHP section:

```
// Since we're passed in recid via param (i.e. viewRecord.php?recid=n), use
// FileMaker::getRecordById() to directly get record object with recid accessed
// via $_GET[] array
$record = $fm->getRecordById('Form View', $_GET['recid']);

if (FileMaker::isError($record)) {
    echo "<body>Error: " . $record->getMessage(). "</
    ➥body>";
    exit;
}
?>
```

Now you can use `$record` in your code.

tip

In many cases, two people write this code: One knows FileMaker and the other knows HTML. You can use this formatting so that they stay out of each other's way. Perhaps the HTML author writes the code for the table and inserts comments for the FileMaker access. Alternatively, the FileMaker author can write code for the PHP accesses of the necessary fields, and the HTML author works around that code to put the data into a table or other page elements.

Get Data from a Record

Finally, you write the HTML code for the page head and body. The code has been reformatted to make it more obvious that you are using PHP snippets to move data from `$record` into the page layout structures (a table and the page title in this case).tip

```
<head>
<title>
  <?php echo $record->getField('Title'); ?>
</title>
<!-- declare charset as UTF-8 -->
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
<link rel="stylesheet" href="style.css">
</head>
<body>
<table>
<tr><th id="table-title" colspan="3">America 24/7 Collection</th></tr>
<tr><th>Title</th><td>
  <?php echo $record->getField('Title'); ?>
  </td></tr>
<tr><th>Author</th><td>
  <?php echo $record->getField('Author'); ?>
  </td></tr>
<tr><th>Publisher</th><td><?php echo $record->getField('Publisher'); ?>
  </td></tr>
<tr><th>Cover Photo Credit</th><td><?php echo
  $record->getField('Cover Photo Credit'); ?>
  </td></tr>
<tr><th>Number of Pages</th><td>
  <?php echo $record->getField('Number of Pages'); ?>
  </td></tr>
<tr><th>Status</th><td>
  <?php echo $record->getField('Status'); ?>
  </td></tr>
<tr><th>Quantity in Stock</th><td>
  <?php echo $record->getField('Quantity in Stock'); ?>
  </td></tr>
<tr><th>Description</th><td>
  <?php echo $record->getField('Description'); ?>
  </td></tr>
<tr><th>Cover Image</th><td>
  <?php if ($record->getField('Cover Image')) {?>
    <IMG src="containerBridge.php?path=
    <?php echo urlencode($record->getField('Cover Image')); ?>
    ">
    <?php } ?>
```

```
  </td></tr>
<tr><td colspan="2" style="text-align: center">
  <a href="editRecord.php?recid=<?php echo
    $record->getRecordId(); ?>">Edit this record
  </a></td></tr>
<tr><td colspan="2" style="text-align: center">
  <a href="deleteRecord.php?recid=
    <?php echo $record->getRecordId(); ?>">Delete this record</a>
  </td></tr>
<tr><td colspan="2" style="text-align: center">
  <a href="displayRecords.php">Back to Record List</a>
  </td></tr>
</table>
</body>
```

Close the HTML Section

As always, you have to terminate the HTML element that you opened on the first line of the file:

```
</html>
```

Using Custom Web Publishing with XML

Preparing for XML Publishing

To publish FileMaker data as XML via the Web Publishing Engine, you must configure your databases for XML publishing as described in the previous section. You also need to enable XML publishing in the Configuration tab of Admin Console, as shown in Figure 25.6.

Introduction to XML Publishing

After you have enabled XML publishing in the database and in Admin Console, you can draw XML data from a served database by opening a web browser and entering a URL like the following:

```
http://192.168.100.101/fmi/xml/fmresultset.xml?-db=Animal&-lay=web&-findall
```

This URL, 192.168.100.101, is the address of the web server that we've configured to work with the Web Publishing Engine. The path to fmresultset.xml indicates that we want the results returned in the `fmresultset` grammar. The URL also instructs the WPE to access a database called Animal, via a layout called Web, and then to find all records and return them in the select `fmresultset` grammar.

note

You do not need to include the filename suffix (.fp7 or .fp12) when referencing the database name in the URL.

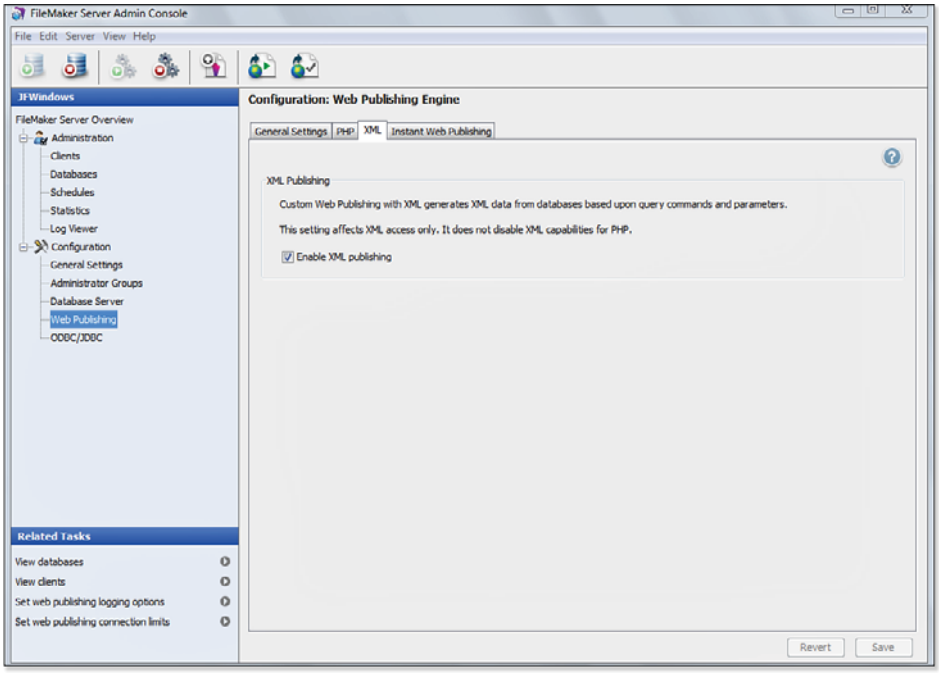


Figure 25.6 Enable XML publishing in Admin Console.

If you had a database called Animal open under FileMaker Server, and if it had privilege sets with the extended privilege for XML enabled, and it had a layout called Web, the Web Publishing Engine would return an XML document to your browser. If you're using a browser capable of displaying XML (which includes Firefox, Safari, and Internet Explorer), you would see something like the code in Listing 25.1.

Listing 25.1 XML Formatted with the `fmresultset` Grammar

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE fmresultset PUBLIC "-//FMI//DTD fmresultset//EN"
➥ "/fmi/xml/fmresultset.dtd">
<fmresultset xmlns="http://www.filemaker.com/xml/fmresultset" version="1.0">
    <error code="0">
    </error>
    <product build="12/10/2003" name="FileMaker Web Publishing Engine"
     ➥version="7.0v1" />
    <datasource database="animal" date-format="M/d/yy" layout="web"
     ➥table="Animal" time-format="h:mm:ss a" total-count="17" />
    <metadata>
        <field-definition auto-enter="no" global="no" max-repeat="1"
         ➥name="date_birth" not-empty="no" result="date" type="normal" />
        <field-definition auto-enter="yes" global="no" max-repeat="1"
        ➥name="id_animal" not-empty="yes" result="text" type="normal" />
```

Listing 25.1 Continued

```
        <field-definition auto-enter="no" global="no" max-repeat="1"
         ➥name="id_father" not-empty="no" result="text" type="normal" />
        <field-definition auto-enter="no" global="no" max-repeat="1"
         ➥name="id_mother" not-empty="no" result="text" type="normal" />
        <field-definition auto-enter="no" global="no" max-repeat="1"
         ➥name="name" not-empty="no" result="text" type="normal" />
        <field-definition auto-enter="no" global="no" max-repeat="1"
         ➥name="weight_birth" not-empty="no" result="number" type="normal" />
        <field-definition auto-enter="no" global="no" max-repeat="1"
         ➥name="weight_current" not-empty="no" result="number" type="normal" />
        <field-definition auto-enter="no" global="no" max-repeat="1"
         ➥name="HerdID" not-empty="no" result="text" type="normal" />
        <field-definition auto-enter="no" global="no" max-repeat="1"
        ➥name="gender" not-empty="no" result="text" type="normal" />
    </metadata>
    <resultset count="17" fetch-size="17">
        <record mod-id="6" record-id="1">
            <field name="date_birth">
                <data>4/23/1994</data>
            </field>
            <field name="id_animal">
                <data>A1</data>
            </field>
            <field name="id_father">
                <data></data>
            </field>
            <field name="id_mother">
                <data></data>
            </field>
            <field name="name">
                <data>Great Geronimo</data>
            </field>
            <field name="weight_birth">
                <data>107</data>
            </field>
            <field name="weight_current">
                <data>812</data>
            </field>
            <field name="HerdID">
                <data>H1</data>
            </field>
            <field name="gender">
                <data>Male</data>
            </field>
        </record>
        [ ... multiple additional records ...]
    </resultset>
</fmresultset>
```

In general, when you want to access XML data from an appropriately configured FileMaker file, you do so by entering a URL in the following format:

```
<protocol>//<server-ip>[:<port>]/fmi/xml/<grammar>.xml?[<query-string>]
```

- `protocol` indicates a web protocol, either HTTP or HTTPS.
- `server-ip` is the IP address of the web server that serves as the point of entry to the Web Publishing Engine. Note that if the Web Publishing Engine is installed on a different machine from the web server, you must specify the IP address of the web server machine here, providing the address of the Web Publishing Engine does not work.
- `port` is an optional part of the URL. In general, your web server will be running on the default HTTP port of 80 or the default HTTPS port of 443. If for any reason you've configured your web server to run on a different port than the protocol default, you need to specify that port number here. This port has nothing to do with any of the WPE-specific ports (in the 16000 range) or the FileMaker Server port (5003) that you might have encountered in the Web Publishing Engine documentation; it refers strictly to the port on which your web server accepts incoming requests.
- `grammar` refers to one of two FileMaker XML grammars: `FMPXMLRESULT` or `FMPXMLLAYOUT`. Only the first of these grammars is available via XML export; the second is available only via Custom Web Publishing.
- `query-string` refers to a series of one or more specific pieces of information you pass to the Web Publishing Engine to form the substance of your request. Among the pieces of information you would pass in the query string are the name of the database to access, the name of the layout you want to work with, and the name of a database *action* (such as "find all records," expressed in the sample URL by the `-findall` command).

In general, then, you'll use specially formatted URLs to access FileMaker data as XML via Custom Web Publishing. These URLs can be manually entered in a web browser, they can be linked from a web page, or they can be used by other processes or applications that want to consume FileMaker data as XML.

Understanding Query Strings

A lot of the action in a Custom Web Publishing URL occurs inside the *query string*—that odd-looking set of commands at the end of the URL. Here again is the sample URL from the previous section:

```
http://192.168.100.101/fmi/xml/fmresultset.xml?-db=Animal&-lay=web&-findall
```

The query string is the portion of the URL that comes after the question mark. A query string consists of multiple *name-value pairs*, with each name-value pair taking the form *`name=value`*. If there are multiple name-value pairs in a URL, an ampersand character (&) separates additional pairs from the first one.

note

Query strings are not peculiar to FileMaker or to Custom Web Publishing; they're an HTTP standard for passing information to a server-side program via a URL.

The sample URL passes three name-value pairs. Table 25.1 shows the names and their corresponding values.

Table 25.1 Name-Value Pairs in a Sample CWP URL

Name	Value	Meaning
`-db`	Animal	Which FileMaker database to access
`-lay`	Web	Which layout in the specified database to use
`-findall`	(no associated value)	What action to perform

In general, any Custom Web Publishing URL has to specify at least a database name, a layout name, and a database action to perform. In fact, you can omit the database name and layout name in the case of a few specialized database actions. But, at a minimum, you will usually provide a `-db` value, a `-lay` value, and the name of some database action.

A few more notes on query string syntax: The *order* of the name-value pairs within the query string doesn't matter, as long as all the required pairs are present. The initial dash (`-`) in the various names *is* significant, however, and can't be omitted. You'll notice that the database action consists of a name without a value (which is perfectly legitimate in an HTTP URL query string); database actions always consist of a name with no value attached.

If you have spaces in your field, layout, or database names, this might cause trouble. ***See "Dealing with Spaces" in the "Troubleshooting" section at the end of this chapter.***

Performing Specific Searches with CWP URLs

The CWP URLs we've looked at so far are simply querying a FileMaker database table, finding all records, and returning the results as raw XML according to the selected XML grammar. But what if you want to query different tables within the chosen database, or select only certain records rather than all records, or apply a sort order to the results? All these things are possible with CWP.

Specifying the Table

One of the reasons it's so important to supply a layout name with your CWP URLs (via the `-lay` parameter) is that the active table is determined by the active layout, via that layout's *table context*. You might recall that there's a Show Records From option in the Layout Setup dialog for each layout. This enables you to select a table occurrence that will provide the layout's table context. When you specify a layout in a CWP URL, you are implicitly setting the active table as well. All commands in the query string are considered to be applied to whatever table underlies the chosen layout.

Finding Specific Records

The Custom Web Publishing URL can also be used to search for specific records. To do this, use `-find` as the database action, instead of `-findall`. You also need to specify one or more search criteria, which are also supplied as name-value pairs.

For example, if you're working with a database of animals, and there's a `name` field for the animal's name, you can use the following URL to search for any animals named Hector:

note

The database in question has only a single table, also called Animal, and that table is the table context for the web layout.

```
http://192.168.100.101/fmi/xml/fmresultset.xml?-db=animal&-lay=web
➥&name=Hector&-find
```

This code snippet specifies a database action of `-find` and adds one more parameter to the query string. We say `name=Hector` to cause the Web Publishing Engine to search for only records where the name is Hector. If there are any such records, they'll be returned in the chosen XML grammar. If there are no matching records, we get back a response that looks a bit like Listing 25.2.

Listing 25.2 Sample Error Response

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE fmresultset (View Source for full doctype...)>
<fmresultset xmlns="http://www.filemaker.com/xml/fmresultset" version="1.0">
    <error code="401" />
    <product build="12/10/2003" name="FileMaker Web Publishing Engine"
     ➥version="7.0v1" />
    <datasource database="" date-format="" layout="" table="" time-format=""
     ➥total-count="0" />
    <metadata />
    <resultset count="0" fetch-size="0" />
</fmresultset>
```

You can see that in the case where no records are found, the XML returned by the Web Publishing Engine contains an error code appropriate to the situation. In this case, the code is a standard "no records found" error. Note that the exact format of the error response varies depending on which XML grammar you specified in the URL.

Specifying an Exact Match When Searching

In the previous example, the search appeared to be for all animals named Hector. This is not exactly true. The previous URL will have exactly the same effect as entering Find mode in the regular FileMaker client, typing **Hector** into the name field, and performing the search. FileMaker, when searching text fields, searches on a "starts with" basis, so this search actually finds animals named Hector, Hector II, Hectorax, and so on. To specify that you want an exact match, rather than a "starts with" match, you need a bit more precision. In FileMaker's regular Find mode, you'd type

=Hector in the search field, with the equal sign indicating an exact match. In a CWP URL, you would write the following:

```
http://192.168.100.101/fmi/xml/fmresultset.xml?-db=animal&-lay=web&name=Hector
➥&name.op=eq&-find
```

Another parameter has been added to the query string here. The new parameter specifies what kind of operator we want to apply to one of the search fields. The syntax for this new parameter is

```
<field-name>.op=<operator>
```

Here, `field-name` is the field to which you want to apply the operator, and `operator` is a short character string indicating one of nine different possible operators. Here, the operator we've chosen is `eq` for an exact match. Other possible operators are `cn` for *contains*, `bw` for *begins with* (the default), and `ew` for *ends with*. So, if you wanted to find all animals with a name ending in *tor*, you could use this URL:

```
http://192.168.100.101/fmi/xml/fmresultset.xml?-
db=animal&-lay=web&name=tor&name.op=ew&-find
```

This query string instructs the Web Publishing Engine to treat the search on the name field as an "ends with" search.

note

The operators available to you in Custom Web Publishing are similar to, but not identical to, the list you would find in the FileMaker client if you entered Find mode and clicked the Operators drop-down menu in the Status toolbar. Both FileMaker Find mode and the Custom Web Publishing find syntax contain operators unavailable in the other. Table 25.2 lists all the operators available in Custom Web Publishing.

Table 25.2 Comparison Operators for the `-find` Command

Operator	Significance	FileMaker Find Equivalent
`eq`	`Equals`	`=value`
`cn`	Contains	`*value*`
`bw`	Begins with	`value*`
`ew`	Ends with	`*value`
`gt`	`Greater than`	`>value`
`gte`	Greater than or equal	`>=value`
`lt`	Less than	`<value`
`lte`	Less than or equal	`<=value`
`neq`	Not equal	(Omit check box)

Performing a Numerical Comparison Search

Consider a database that contains some numerical fields. The Animal database used as an example so far contains a field called `weight_birth` for an animal's birth weight. Suppose that you want to find all animals with a birth weight less than 100 pounds. The following URL would do it:

```
http://127.0.0.1/fmi/xml/fmresultset.xml?-db=animal&-lay=web&weight_birth=100
➥&weight_birth.op=lt&-find
```

Here, `100` is specified for the `weight_birth` search field, but we go on to say that the *operator* for that search field is the less-than operator, symbolized by the code `lt`.

Searching on Multiple Criteria

Suppose that you want to construct a more narrowly tailored search. You want to find all *male* animals with a birth weight less than 100. This is the equivalent of filling in two fields in FileMaker's Find mode, instead of just one. You would use a URL like the following:

```
http://127.0.0.1/fmi/xml/fmresultset.xml?-db=animal&-lay=
➥ web&weight_birth=100&gender=Male&weight_birth.op=lt&-find
```

Here, we've simply added one more search field: `gender=Male`. This constitutes a further limit on the search you saw in the previous example. This search finds only records for male animals with a birth weight less than 100.

Creating Multiple Find Requests

The preceding example showed how to use multiple criteria to narrow a search. But what if you want to use multiple criteria to *broaden* a search? We've searched for animals with a birth weight less than 100. What if you also want to find, in the same search, any animals that have a current weight less than 500? You might recognize this as the equivalent of creating additional Find requests in the regular FileMaker Pro software.

To explain this kind of search, we need to introduce the concept of a *logical operator*. In the search demonstrated previously, for a record to be included in the search, *all* the search criteria in the query string had to be true. That is, an animal would not be included in the search results unless it was both male *and* had a birth weight less than 100. This kind of search is therefore often referred to as an *and* search or an *all-true* search.

On the other hand, when you think about also finding animals with a current weight less than 500, you have a situation in which an animal will be included in the search results if *any* of the search criteria are true. In other words, a record will be found if the animal had a birth weight of less than 100 *or* it has a current weight of less than 500. This type of search is thus often called an *or* search or an *any-true* search.

By default, the Web Publishing Engine treats all searches as *and* searches. To perform an *or* search, you use a URL like this one:

```
http://127.0.0.1/fmi/xml/fmresultset.xml?-db=animal&-lay=web&weight_birth=100
➥&weight_current=500e&weight_birth.op=lt&weight_current.op=lt
➥&-lop=or&-find
```

Here, you supply two search criteria. You also have to supply the field-level operator for each search field. In both cases, you're performing a *less-than* search, so you need to specify an operator of `lt` for each field. The new element in the query string is the `-lop` parameter, which stands for *logical operator*. `-lop` can have a value of *and* (the default) or *or* (the one used here). The `-lop` parameter here instructs the Web Publishing Engine to treat the search as an *or* search.

note

In FileMaker proper, you can construct a search that's a complex mixture of *and* and *or* searches by entering multiple Find requests, each with more than one field filled in. Such searches can't readily be reproduced with Custom Web Publishing: The `-lop` command can be applied only to all the search fields taken together. There is also no way to invoke the additional FileMaker search options of Constrain or Extend Found Set.

Specifying a Sort Order for Search Results

When you make a request to the Web Publishing Engine, you can specify how the results should be sorted. You can specify one or more fields to sort on, as you can in the regular FileMaker application, and you can specify whether each sort field should be sorted in ascending or descending order. Consider a URL that will find all records in the Animal table and ask that the records be sorted by name:

```
http://192.168.101.100/fmi/xml/fmresultset.xml?-db=Animal&-lay=web
➥&-sortfield.1=name&-findall
```

The new query string command here is called `-sortfield`. You'll notice we also added the suffix `.1` to this parameter. This indicates the sort field's *precedence*. The concept of precedence is meaningful only if you have more than one sort field, as you'll see in a moment. Despite this fact, you can't omit the sort precedence, even for a one-item sort; otherwise, the records won't be sorted at all.

Suppose that you wanted to sort the records by gender and to sort within each gender by current weight from highest to lowest. You would do that like this:

```
http://127.0.0.1/fmi/xml/fmresultset.xml?-db=Animal&-lay=web
➥&-sortfield.1=gender&-sortfield.2=weight_current&-sortorder.2=
➥ descend&-findall
```

Here, two sort fields are specified. The first sort is by gender, the second by `weight_current`. There's also a new parameter, called `-sortorder`. Like `-sortfield`, `-sortorder` also takes a numeric suffix. Here, it's used to indicate which sort field is being referred to. By default, each field will be sorted in ascending order. If you supply a value of `descend` for the second sort field, you ensure that the animals will be sorted, within each gender group, from heaviest to lightest.

Applications of Custom Web Publishing with XML

The preceding sections show how to use the Web Publishing Engine to query a database and publish the results as raw XML in one of several XML grammars. But what use is this capability, exactly?

Well, the most obvious significant use is to allow FileMaker to act as a web service provider. If you provide a web service client with an appropriate URL, remote services and programs can query your FileMaker database via the Web Publishing Engine and extract whatever information you choose to let them see.

Other Query Parameters

In addition to a single database command, Custom Web Publishing URLs can contain other parameters. Some are mandatory, such as `-db`, and (generally) `-lay` and `-grammar`. Others, such as `-lop` and `-sortfield`, are particular to specific commands. Table 25.3 shows a list of the most important ones.

Table 25.3 Other Custom Web Publishing URL Parameters

Parameter Name	Parameter Effect
`-db`	Name of the database on which to act. Mandatory for all commands except for `-dbnames` and `-process`. Do not include a filename extension (such as .fp7 or .fp12) when using this parameter.
`-field`	Use the `-field` parameter with the name of a container field to request the contents of the container field.
`fieldname`	Use plain, unadorned field names as query parameters when sending data for use with the `-new`, `-find`, and `-edit` commands. Refer to the "Performing Specific Searches with CWP URLs" section, earlier in this chapter.
`Fieldname.op`	Sets the comparison operator for `fieldname` when performing a search. Refer to the table of operators, Table 25.2, earlier in this chapter.
`-lay`	Specifies which layout and, therefore, which table context to use for the request. Mandatory with all commands except `-process`, `-dbnames`, `-layoutnames`, and `-scriptnames`.
`-lay.response`	Enables you to use one layout for processing the command contained in a URL and a different layout for generating the XML that comprises the response. For example, you might want to process your request (an Add, say) via a layout with certain hidden fields on it, but process the response via a layout that omitted those fields. Data could thus be added to the hidden fields, but that hidden data would then be omitted from the response.
`-lop`	Used with the `-find` command, specifies whether to treat the search as an *and* search or an *or* search.

Table 25.3 Continued

Parameter Name	Parameter Effect
`-max`	Used with the `-find` command, specifies the maximum number of records to return. Sending a parameter of `-max=all` permits all records to be returned. This is the default.
`-modid`	FileMaker's modification ID is an internal number that increments every time a record is changed. Use the `-modid` parameter to ensure that the record you're editing has not been edited since the time you last checked the modification ID. This is useful for prohibiting different users' changes from overwriting each other.
`-recid`	Specifies which record should be affected by a given action. This parameter is mandatory with `-edit`, `-delete`, and `-dup`, and can also be used with `-find`.
`-script`	Use this parameter to run a FileMaker script during the processing of the request. By default, the script runs after the query command and any sorting have occurred. For example, if you run a script in your FileMaker solution after each new record is created, you can create a URL with the `-new` command that also includes the `-script` parameter for that post-creation script.
`-script.param`	Use this parameter to pass a parameter into a script.
`-script.prefind`	If your command URL involves any kind of find request, use this parameter to request a script to be run before the specified search takes place.
`-script.prefind.param`	Use this to pass a parameter into the `-script-prefind` script.
`-script.presort`	If your command URL involves any kind of find request and a sort, use this parameter to request that a script be run after the specified search takes place, but before sorting.
`-script.presort.param`	Use this to pass a parameter into the `-script.presort` script.
`-skip`	Used with the various search commands, specifies that records should be returned starting elsewhere than at the first record. If you specify `-skip=10`, the records are returned starting with the 11th record.
`-sortfield.[1-9]`	Specify any of up to nine different fields to sort by.
`-sortorder.[1-9]`	For a given sort field, specify whether it should sort ascending or descending.
`-styletype`	Used with `-stylehref`. Use this parameter to specify a client-side stylesheet for additional processing. The most common choices would likely be CSS. For these choices, you would specify `-styletype=text/css`.
`-stylehref`	Use this with `-styletype` to specify the location of a stylesheet for client-side processing. Note that this option and the previous one are effective only when the user's client (generally a browser) supports some form of client-side stylesheet processing.
`-token.[string]`	Use this to pass additional data from one stylesheet to another. Refer to "Using Tokens to Share Data Between Stylesheets," earlier in this chapter, for more detailed information.

About Sessions

If you've read Chapter 24, you've already encountered some discussion of the concept of *sessions*. To recap briefly: The connection between a web browser and the Web Publishing Engine is very much *unlike* the connection between a client copy of FileMaker and the FileMaker Server. FileMaker Server can at any time reach out and push data to any connected client. It knows at all times what its connected clients are, where they are in the system, and at what network address they can be found. A web server, by contrast, retains no memory of a client from one connection to the next.

This is not a good thing for database work! I need my website to remember the contents of my shopping cart as I shop around the site. This is possible only with *session management*. Session management is generally a middleware feature. Web programming languages such as PHP and JSP offer the programmer different means of managing sessions.

In general, under session management, each incoming web request is associated with a key of some kind. The key can be passed in the URL (if you've ever seen a long ugly string such as `?jsession=A9238Ajasdj9mAEd` in a web URL, odds are you were looking at a session key), or it can be passed behind the scenes in an HTTP cookie. FileMaker's Custom Web Publishing session implementation lets you choose between these two methods. Whatever the means, the middleware on the web server has a way of associating that key to other information about the client. In the shopping cart example, the key might hook up to a database record that stores the actual contents of your cart as you navigate around the site.

FileMaker's Custom Web Publishing, like other middleware solutions, enables you to manage sessions for your users behind the scenes. You would use this capability any time you wanted to store important information about the user that would be carried from screen to screen. An experienced HTML programmer could get away with passing a lot of data from page to page via the URL or via an HTML form. But there are limits to the amount of data than can be passed by URL, and there are limits to the *type* of data that can be passed by either method—generally just plain text strings.

FileMaker's session implementation is quite elegant because it allows you to pass around XML fragments behind the scenes. This allows for much richer data structures than you could pass with regular HTML.

Managing Sessions

In addition to passing around XML information by means of sessions, FileMaker's session implementation allows you to keep track of the state of the FileMaker client session as well. Database sessions are an additional capability on top of regular session management. In addition to "sessionizing" user information of your choice, they enable you to keep track of FileMaker-specific information such as global fields or the current script state.

So, for example, if your stylesheets modify a global field and you have database sessions enabled, the global field retains its new value for the specific current user as that user navigates from page to page. Or, if you use a script to change some aspect of the user's state (for example, by using the `Re-login` script step to change the user's privileges), this state is maintained across sequential requests.

Session management is a large topic, and we don't have space to do it justice. The FileMaker documentation helps you get a better grip on the specific functions and commands that Custom Web Publishing uses for session management. If you intend to make heavy use of global fields or call scripts from your stylesheets that would change the state of a user's privileges, you should configure the Web Publishing Engine to enable database sessions.

Troubleshooting

Getting the Right Privileges

I can connect to my Web Publishing Engine and FileMaker Server via the Administration Console, but I don't see the databases I expect to see.

Make sure that for every database you want to make available via XML-CWP, you have attached the appropriate extended privilege (fmxml) to at least one privilege set.

Dealing with Firewalls

My web requests mysteriously time out, as though something were blocking them.

If you can get to the Administration Console, but your Custom Web Publishing URL requests appear to get no response, you might have a firewall in your way. If you suspect a firewall might be involved, consult your network administrator to explore this question. If it turns out that your machines are set up such that your web server is on one side of a firewall and your Web Publishing Engine machine or FileMaker Server machine is on the other, you need to open certain ports in the firewall. Here are the rules:

- When the web server and Administration Console are on one machine and the Web Publishing Engine on another, traffic must be able to flow between the two machines on ports 16016 and 16018.
- When the Web Publishing Engine is on a different machine from FileMaker Server, traffic must be able to flow between the two machines on port 5003.

Dealing with Spaces

The Web Publishing Engine doesn't seem to see my entire URL. I enter a long URL, and the web server appears to truncate it and reports that the shorter URL can't be found.

If, despite the cautionary notes in this chapter, you have left any of your databases, fields, or layouts with spaces or any other nonalphanumeric characters in their names, your Custom Web Publishing URLs might very well break. If a web server or browser encounters a space in a URL, it might assume that the URL ends there. Other nonalphanumerics have different but equally irritating effects.

If you must work with URLs with spaces in them, you can get by with replacing all spaces with the string `%20` whenever you need to write out a URL. Your stylesheet then might generate an HTML page with the following link:

```
<a href="http://192.168.101.100/fmi/xsl/process-this.xsl?
➥-db=Too%20Many%20Spaces&-lay=Spaces%20Here%too&-findall
```

tip

If at all possible, we strongly encourage you to use only alphanumeric characters for database, layout, and field names and to avoid the use of whitespace. Extend this caution to script names if you are planning to call scripts from the Web.

This page intentionally left blank

26

DEPLOYING AND EXTENDING FILEMAKER

FileMaker Deployment Options

One of the strengths of FileMaker is that a solution can be deployed in various ways. It can be used by a single user running FileMaker Pro or FileMaker Pro Advanced, it can be shared with peer-to-peer networking, it can run on FileMaker Server, or with FileMaker Server Advanced it can be published to the Web. This chapter explores runtime solutions, kiosk solutions, and plug-ins—additional ways to deploy FileMaker databases.

note

Perhaps the most popular deployment option today involves FileMaker Go and the iOS devices (iPhone, iPad, and iPod touch). That topic is covered in Jesse Feiler's book, *Building Data-Driven iOS Apps for iPad and iPhone with FileMaker Pro, Bento by FileMaker, and FileMaker Go*. As you have seen, although there are some considerations for mobile devices that you take into account in building layouts for FileMaker, the major work is provided for you automatically by FileMaker Go.

tip

These deployment options are not mutually exclusive; that is one of the great features of FileMaker Pro. If you use the suggested architecture of separating your database file from interface files, you can create a solution with separate interface files for different types of deployment: Custom Web Publishing, Instant Web Publishing, Kiosk mode, Runtime, and standard. Each interface file can have its own set of accounts and privileges, along with layouts designed for optimal use and display in each of the deployment vehicles. Your underlying database shouldn't even know what type of deployment is going on. Even better, such multiple deployments can run simultaneously (although runtime and kiosk solutions cannot be networked and must run in single-user configurations unless they are hosted on FileMaker Server and accessed by FileMaker Pro clients).

Renaming Files

Renaming files might sound trivial, but don't let the apparent simplicity here deceive you. Multifile solutions in FileMaker depend on filenames to maintain internal references. If you arbitrarily rename one of the files in a given solution via your operating system, FileMaker prompts you with a "File could not be found" error when it next tries to resolve a reference to that file. You risk breaking table occurrence references, script references, value list references, and more by renaming your files manually. We very strongly recommend against manually renaming individual files within a solution. You can, however, place all the solution's files within a single folder (which is generally a good idea) and then copy or rename the entire folder without worrying. It is the renaming of individual files, not their enclosing folder, that causes the problems.

tip

If you run across a file that shows signs of having been incorrectly renamed or lost altogether, the Database Design Report is a great place to turn to root out "file missing" problems.

You can name your files by using the Developer Utilities dialog. Notice, as in Figure 26.1, that you will need to add all the files for a given solution to the dialog. This is important: You have to add both the file you want to rename and all the files that reference it. Then set new names for however many files you need to change. For example, suppose that you have a system composed of ten linked files. Load all the files into the Developer Utilities dialog. Rename just the one file you intend to rename by typing in a new name and clicking Change. When you click Create, FileMaker generates new files in your destination project folder, leaving the old files unchanged. In the ten-file example, the one file would have its name changed, and all ten files would have any references to that file updated to use the new name.

After you click Create, the files are copied to the new location that you set by clicking Project Folder and selecting (or creating) a new folder. Note that you can choose to overwrite files with the same names.

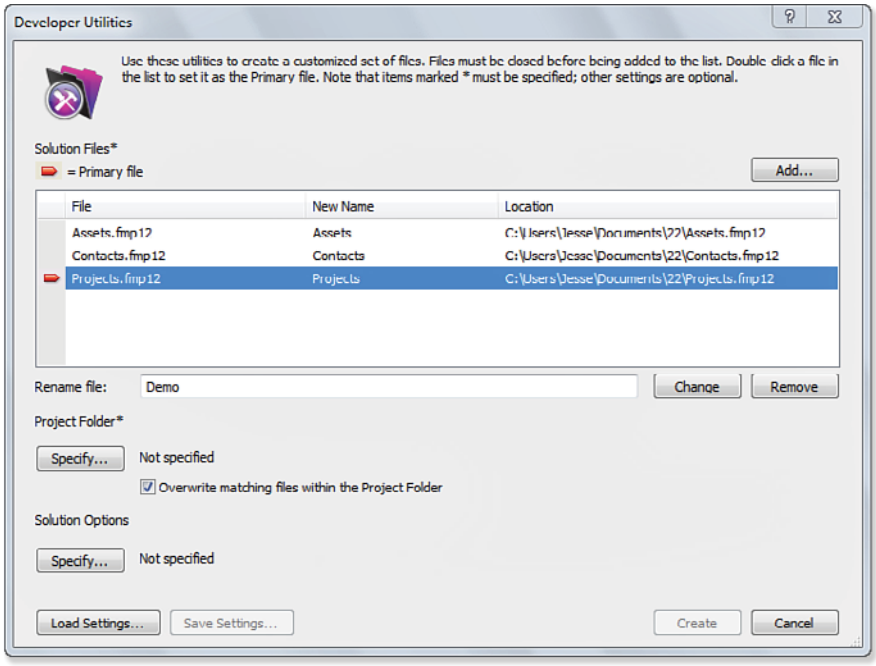


Figure 26.1
Use the Developer Utilities dialog.

The consequence of changing the names, setting a new project folder, and clicking Create is to copy the files to the new location and to change their names. But much more important is the fact that FileMaker changes the file path list appropriately. No-longer-needed file paths are removed. Particularly if you manually rename individual files, over time the file paths can become long lists of files no longer needed. In time, it is even possible for files that you no longer need and have long ago renamed to reappear as you clean up your hard disk; lo and behold, your FileMaker solution can break.

To learn how to manually address filename and reference problems, **see** *"File Reference Errors" in the "Troubleshooting" section at the end of this chapter.*

Runtime Solutions

For some solutions, the best deployment option is as a bound, runtime solution. A runtime solution can be distributed to users who can run it without having a copy of FileMaker Pro on their machine. Runtime solutions are created with the Developer Utilities, which are available only in FileMaker Pro Advanced.

Another deployment option that's available via the Developer Utilities is to create a kiosk from your FileMaker solution. When run as a kiosk, a solution takes up the entire screen. Users don't even have access to the Status toolbar or any menus, which means you must provide buttons for every conceivable action they might perform. This topic is discussed later in this chapter in the section "Developing Kiosk Solutions," **p. 633.**

A typical example of a solution that you might deploy as a runtime solution is a product catalog. Perhaps you developed a FileMaker database of all your products, and you want to send it to all your customers. You could create a runtime version of the files and do just this. Your customers would be able to browse and search for items, maybe even print or email orders to you, all without having a copy of FileMaker on their machines.

You'll have to keep some conditions in mind. On its own, a bound runtime version does not support further development; a runtime solution does not include Layout mode, ScriptMaker, and the Manage Database functions, thus disallowing further editing of the files. However, unless you have bound it with the option to remove Admin privileges, you can open a bound runtime version with FileMaker Pro to make such changes. A runtime solution works only with the files bound with it; it may not be linked to other databases, either other runtimes or files hosted via FileMaker Server. Finally, a runtime solution is single-user only. If end users want to share the files, they have to turn to FileMaker Server and standard copies of FileMaker Pro.

tip

If you need to distribute a runtime solution to both Mac and PC users, you must bind a separate version for each platform, and you therefore need access to both a Mac and a PC during development—either on separate computers or on an Intel-based Macintosh that can run both operating systems.

note

It's sometimes thought that a runtime file is necessarily read-only, but this isn't the case. Assuming that the database user has the correct permissions, a runtime can be used to create, edit, and delete records just as with the regular FileMaker client. The misconception might stem from the fact that bound files are often distributed on CD, and such files are indeed read-only until they're copied from the CD to a writable medium such as a hard drive.

Solution Options

You start by using the Developer Utilities dialog shown previously in Figure 26.1. Remember that all files must be closed at this time. Add the files you will be using to the dialog as you would do to rename them. Then, to create a solution, use the Specify button under Solution Options, to set the range of actions that FileMaker Pro Advanced can perform as it creates a new solution and a new set of files. All these options generally pertain to readying your files for deployment; you would not necessarily use them during development, but rather at the end when you're preparing files for hand-off to users. Figure 26.2 shows the Specify Solution Options dialog.

At the top of the Solution Options dialog, you choose the option you want; in this case, you want to create a Runtime solution. After you choose it, you get options specific to that choice:

- **Runtime Name—**The runtime name will be used to name the resulting solution directory, and it will also be the name of the master file created for the runtime (more details on the master file follow this list).

tip

Although you do not need to create your runtime solution until the development process is complete, it is a good idea to do periodic reality checks by creating runtime solutions along the way. This will help you find problems before they become show-stoppers at the last minute. This advice also applies to any other deployments you may be planning, including the Web.

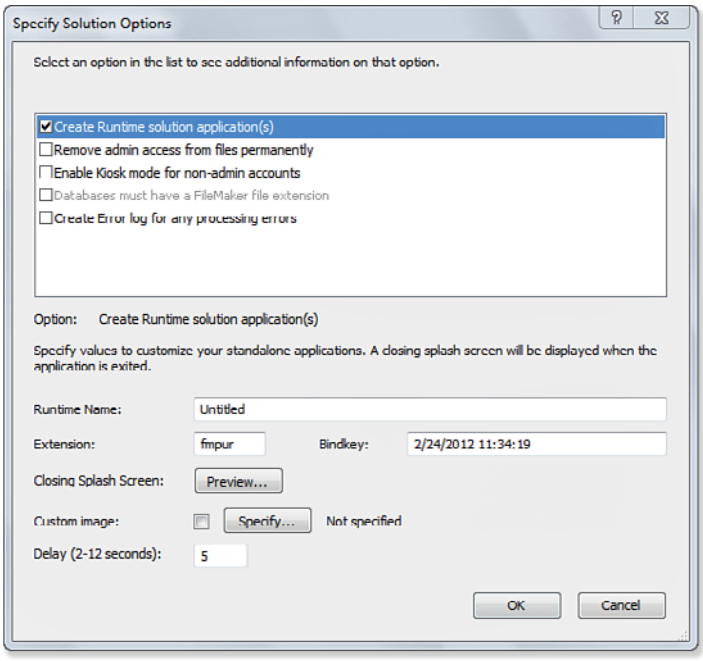


Figure 26.2
Solution options enable you to prepare a set of files for deployment.

- **Extension**—To distinguish the runtime files from regular FileMaker files, which in many senses they still are, the binding process adds a custom file extension to each of the solution files. You can choose your own extension; otherwise, a default extension of .usr will be applied.

  The extension for FileMaker-bound runtime solutions determines, in both Mac OS X and Windows, what application becomes associated with your individual solution files—which by definition is the runtime application you're in the process of creating. These file extensions simply help identify the application that should open your files and differentiate them from other FileMaker Pro documents.

- **Bind key**—For the runtime application to recognize its associated files, the bind key in a given file needs to match the bind key of the application. This simple pairing ensures that a given application will authorize use of specific FileMaker Pro files. Notice in Figure 26.2 that FileMaker Advanced inserts a timestamp, by default, as a bind key. (If you are going to be binding both Windows and Mac OS X versions, you may choose to use the same bind key for both. This means that if you use the default timestamp bind key for the first solution that you bind, you should retype it exactly—or copy and paste it—into the second solution that you bind.)

To replace or add a file to a solution that has already been bound, use the same bind key when preparing that new file, and users will be able to drop the file in question directly into their solution folders. You need not replace the entire solution.

Consider cases in which you would want to be able to add files to a solution to upgrade functionality or address bugs. This introduces the complex issue of upgrade paths in a FileMaker Pro solution. You must remember that after someone begins using your solution, he will be adding and storing data in your files. If you were to simply replace those files with no concern for exporting or managing that data, users would open their applications and discover an empty shell waiting again for the creation of the first records.

- **Closing Splash Screen—**When users close your solution, they will see a small closing splash screen. You can determine how long the screen will be visible (2–12 seconds).
- **Custom Image—**By default, the closing splash screen shows a FileMaker logo. You can instead include an image of your own for display on the closing splash screen. If you choose to include a custom closing image, size it for 382×175 pixels at 72 dpi. JPEG and GIF both work best in cross-platform environments; we don't recommend any other file type.

After you've chosen your solution options, you can click OK to start the process of creating the solution. The solution files are written into a directory with the same name as the runtime name you established previously. It's a common misconception about the runtime binding process that the result is one single, monolithic file. Try the process for yourself and you'll see that this is not the case (remember, it creates a new set of solution files, so there's no need to worry about hurting your current files). On the Mac, you'll get a sparse file set, whereas with Windows you'll get dozens of supporting DLLs. Don't be surprised by the differences between the two platforms.

Regardless of platform, each bound solution contains a master file, of which you'll want to take special note. The file has the name *solution_name.extension*, where *solution_name* is the solution name you chose when binding, and *extension* is the custom extension you chose. If you were creating a solution called Sales, and chose the default .usr extension, the master solution file would be called Sales.usr.

In addition to the master file, there will also be a single additional file for each FileMaker file that went into the solution. Each will be named with your chosen file extension. So, if your Sales solution was made up of files called Contact, Company, and Order, the bound solution would contain the following files: Sales.usr (the master file), Contact.usr, Company.usr, and Order.usr.

Using the Master File

The master file is significant because this is the file that must be run to gain access to the solution. For example, if you were packaging the runtime onto a CD, the CD might contain your solution directory, but also a shortcut to the master file at the root level of the CD. You'd rather users not have to rummage around in a directory full of files to find the right one.

The individual database files (as opposed to the master file) are actually not much changed by the binding process. The database files within the application remain FileMaker Pro files, accessible from FileMaker Pro proper, assuming that you haven't disabled such access via the Remove Admin Access solution option covered later. You could continue to work with these files in FileMaker Pro

or FileMaker Advanced, add features, and simply redeploy the altered files without having to re-create a runtime solution each and every time a change is called for. Likewise, you can have some users make use of the runtime applications and still others access separate copies of the files (or share files) with full versions of FileMaker Pro. It's rare that you would build a database that could be used in both single-user and multiuser modes, but the point here is that it's possible.

Note that the preceding discussion somewhat mitigates the point that bound solutions are platform specific. This is true of the solution as a whole, but the constituent database files remain for all intents and purposes FileMaker files and can be edited as such on either platform.

Some bound runtime solutions require a good bit of data entry prior to their being ready to distribute to a wide audience. It can be convenient to host the files—just as they are—on FileMaker Server to allow multiple people to enter data. The fact that the FileMaker files themselves are unaltered by the binding process means that you can swap them between a bound runtime application and FileMaker Pro or Server as needed.

Removing Admin Access

Removing admin access often goes hand-in-hand with creating a runtime solution, but it doesn't necessarily have to. To prevent anyone—including yourself—from changing the files in a given solution (regardless of whether you intend to bind them into a runtime), it is possible to remove all admin (or better, perhaps, developer) access to a set of files.

You'll remove access to the Manage Database, Value Lists, File References, Accounts & Privileges, and Custom Functions dialogs. Access to Layout mode and ScriptMaker is also removed.

In addition, removing admin access removes any accounts set up explicitly with the [Full Access] privilege set. This is quite important because it actually modifies the account and privilege settings of your files. Your "developer" account will be removed. If you have written scripts that depend on a certain account being there, you must be careful in how you accomplish such functions. You also have to ensure that you have a password that will allow you into the solution after you run this process.

It's possible to define an account and assign a custom privilege set that has the equivalent of full access without assigning it to the built-in [Full Access] set, but keep in mind that, again, the capability to use all editing functions will be removed from the files. Those menu options, regardless of the account you used to sign in, will be grayed out.

There's no going back after you remove admin access, so be certain that you have all the kinks worked out of your solution, and keep your original files backed up!

Make certain there's a good way to export all data from a solution before removing admin access. Just write a scripted routine that saves all records to XML files. Doing this at least ensures that you can extract data from a locked-down version of your solution.

For a complete understanding of security in FileMaker, **see** *Chapter 12, "Implementing Security."*

Polishing Your Custom Solution

When distributing a custom solution, you can better tailor its look and feel by creating a custom menu scheme that reflects and supports the identity of your application. You can implement a completely customized menu scheme.

A custom menu scheme allows you a very high degree of control over your solution: You could write a complete help system that might include opening a FileMaker Pro file or interface in itself. Users might then be able to perform Find requests and employ other familiar approaches to using your system. The About menu could be as simple as a window with an image or a logo that is brought forward, or you could get as fancy as a QuickTime movie that is played within a container field. FileMaker Pro Advanced gives you the opportunity to truly customize a solution so that it takes on an identity of its own.

For more information on custom menus, **see** *"Working with Custom Menus,"* **p. 396.**

caution

Note that a custom menu scheme will not be of any use in a solution destined for Kiosk mode because Kiosk mode removes menu access.

tip

If you take pains to give your solution a name and add even simple levels of customization, end users will more easily accept the system that they will presumably spend a good percentage of their work lives using.

It's also somewhat helpful in getting users and IT folks to differentiate FileMaker Pro—the technology—from your specific solution. If you name and modestly customize it, you foster a better sense of differentiation by creating an identity other than "the FileMaker database."

Error Log

As Developer Utilities runs, it can keep track of any errors it encounters. To generate a log, simply turn on this option in the Solution Options. A text file named LogFile.txt is created in your solution folder. Some Developer Utilities processes run into errors that don't prompt dialogs, so it's a good idea to check the log before wrapping up a solution for end users. The following are the errors you'll find in the log:

- Updating File Specs for this destination file skipped due to a previous fatal error.
- Destination file could not be created, and all further processing on it was skipped. File:
- Skipped runtime generation, due to missing or damaged resources.
- Destination folder could not be created, and all further processing was skipped. Folder name:

As you can see, these messages aren't particularly illuminating and generally indicate that you have a significant problem with the interaction between your OS and FileMaker's processes. In testing for these conditions, a full hard drive was the cause for some of these issues. If you see such messages, verify that it's possible and practical to create a solution directory in the place you chose (meaning, check for a full disk, restrictive permissions, and the like), and verify that the source files open correctly and don't appear corrupted.

Developing Kiosk Solutions

The completely encapsulated user experience of kiosk mode often looks very much like a FileMaker database that is opened in FileMaker Go on an iOS device.

Kiosks are good ways to present users with a completely encapsulated user experience. As an example, one favorite project is building a kiosk-based wine recommendation service for grocery stores using touch-screen input.

Kiosk mode allows FileMaker Pro to open full-screen, with no toolbars or menus. On Windows and Mac OS X, the taskbar and Dock, respectively, become unavailable as well. This has the effect of taking over the entire computer environment and allowing you to build complete appliances that serve a specific purpose. If you combine Kiosk mode with an alternative means of data input—touch-screen input, barcode readers, or other devices—the result can be something that very much departs from what you might think of as a database.

Securing Kiosk Mode

Kiosk mode does not completely lock down a computer. On Windows, users can still use Alt+Tab to access different running processes. The way to avoid this is simply to establish FileMaker Pro as the only running application. Also on Windows, Ctrl+Alt+Delete calls forward the Windows Task Manager, and the Windows key on current keyboards brings forward the Start menu. You need to take additional steps to lock down Windows.

If you plan on deploying many kiosks, this would be tedious, but you can use a system utility such as gpedit on Windows XP to lock access to various elements such as the Alt key and Start menu. Another approach is to use a third-party utility such as Win Control: www.salfeld.com/software/wincontrol/index.html.

On Mac OS X, this is not as much an issue. Kiosk mode properly takes control of the computer environment, but there are still backdoors, not the least of which is simply pulling the power cable of the computer in question.

In general, though, keep in mind that Kiosk mode is meant to facilitate a storefront experience especially geared toward touch-screen input, and it is not focused on delivering a specific level of security.

Preparing a Kiosk Interface

When preparing a solution for Kiosk mode, you need to consider several unique issues, not the least of which are important user interface elements. Because FileMaker's menus are inaccessible in Kiosk mode, a vital requirement is to offer users a means for at least exiting the application. Without a scripted quit routine, users have to force-quit the application and might lose data as a result.

Being able to exit the application, though, is just the first requirement. Any function you'd like users to be able to perform must be scripted and attached to a layout object. You can opt to leave the FileMaker Status toolbar open if you want, but none of FileMaker's native keyboard shortcuts will work (for, say, creating or deleting records).

Most kiosks offer a complete set of scripted functions attached to a custom-crafted user interface, and very rarely do developers opt to leave the Status toolbar open. Therefore, you need to create scripts and buttons for navigating from layout to layout, for creating records, for managing any importing or exporting of data, and for dealing with upgrading the files themselves, if necessary.

Maintaining a Kiosk Solution

After deploying a kiosk to end users (it could just be a copy of a FileMaker database you're distributing widely), you leave the world of modifying and managing workgroup solutions and enter the world of commercial development, where your ability to tweak things becomes exponentially more difficult. This suggests that a solution needs to be completely tested and perfect before it goes out the door; otherwise, you have to craft and implement an upgrade strategy that allows you to pass new functionality to your users without leaving them lost, with no means of preserving whatever data they might have input. This strategy could be as simple as exporting all data from the old version and importing into the new, or you could build a distributed file system in which it's possible to replace certain files without altering the data itself.

For ideas on user interface approaches, **see** *Chapter 14, "Advanced Interface Techniques."*

Plug-Ins

Plug-ins extend FileMaker Pro's capabilities and are quite varied. Their offerings range from charting functionality, OS-level file manipulation, and barcode readers, to scientific math functions, credit-card authentication, help systems, telephony, and more.

Plug-ins are written and compiled in accordance with FileMaker Pro's plug-in API. They're not something many FileMaker developers will ever have to create, and you generally do not have access to the code from which they're built.

note

We encourage you to visit FileMaker's website to explore a wide range of plug-ins. Just go to http://solutions.filemaker.com and click Plug-ins in the For Developers area at the lower right.

If you want to delve into writing your own plug-ins, you must be an expert in either the C or the C++ language. (We don't recommend a FileMaker plug-in as your first C++ project!) You also need a development environment, such as Xcode for the Mac or Visual Studio for Windows, and the plug-in API documentation and sample files that ship with FileMaker Pro Advanced. Plug-ins are platform specific, so if you want your plug-in to work on both Mac and Windows, you need to do at least some reengineering to get your code to compile and run correctly on both Mac and Windows.

As in all third-party software products, we recommend you get to know a given plug-in well and test it along with the rest of your solution before deploying. Another obvious consideration is cost: Some of your clients might benefit from utilizing a plug-in, but remember that this is third-party software that might require a purchasing license.

Understanding Plug-ins

Plug-ins work by adding external functions to your calculation functions list. Generally, but not always, they take a single text parameter (although the parameter may be internally delimited, containing several values). The result of the plug-in operation is delivered in the form of a calculation result.

An actual external call might look like this:

```
XMpl_Add( numberInput1; numberInput2 )
```

If you use this plug-in function, it returns the sum of two numbers. To make use of the function, you generally have to store its result someplace—often in the context of a script step that puts the value into a field or variable:

```
Set Variable [$sum; XMpl_Add( numberInput1; numberInput2 )]
```

The name of the plug-in function is a string specific to the plug-in you're working with. The plug-in governs the string's syntax, and if it follows proper FileMaker, Inc., conventions, the string includes the name of the plug-in as well. In this case, the example is drawn from FileMaker's included sample plug-in (described further in the next section), and XMpl_ is the prefix FileMaker chose. Likewise, the expected parameters passed as text vary widely from none to complex data arrays. FileMaker Pro's data storage limit of 2GB per field means that we could be facing some quite complex programming within the realm of a single text field.

The results of a plug-in are returned as a calculation result, but often some other action might be performed as well. For example, a dialog might appear. Often the calculation field simply serves as a means for passing error conditions.

For example, a plug-in might copy an image file from one directory to another. Or it might display a dialog of some kind. Or it might create a chart image and place it on your Clipboard. The possibilities are nearly endless, and we recommend, again, exploring available plug-ins to understand specific cases.

Installing Plug-Ins

There are two distinct types of plug-ins: purely client-based and client-based with a server-side component. Deployment is consistent between the two, and it has two varieties:

- **Extensions Folder for the Application—**To enable a plug-in, place it in the Extensions folder with the FileMaker application folder for each client, regardless of whether it has a server-side component. In addition to the client-side installation, server-side plug-ins have to be deployed to the Extensions folder on the server as well. This is true for both Windows and Mac OS X platforms.
- **Extensions Folder for the User—**Instead of placing the plug-in in the FileMaker application's Extensions folder, you can place it in an individual user's FileMaker Extensions folder. The locations of these folders are as follows:

  Windows: C:\Users\User Name\AppData\Local\FileMaker\Extensions

  Mac OS X: Macintosh HD/Users/User Name/Library/Application Support

➥ *If your plug-in is not responding, refer to "Plug-in Not Responding or Not Installing" in the "Troubleshooting" section at the end of this chapter.*

Configuring and Enabling Plug-Ins

To enable a particular plug-in, use the Preferences dialog within your FileMaker Pro or FileMaker Pro Advanced application, as shown in Figure 26.3. To use a given plug-in, you have to explicitly enable it by marking its respective check box. The AutoUpdate plug-in is always available in this list; you can choose to enable it or not (if you are using plug-ins, you usually do want to enable it). Some plug-ins offer configuration choices.

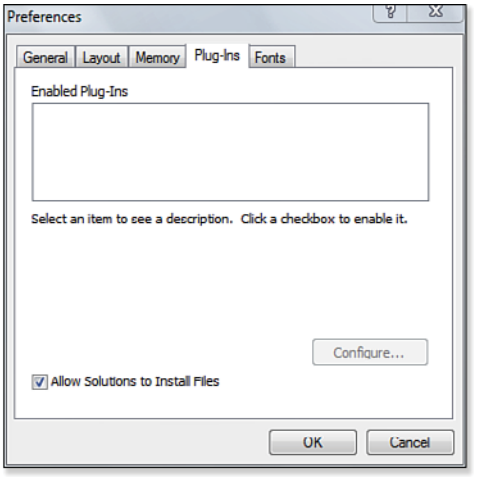


Figure 26.3
Plug-ins are enabled and configured via the Preferences dialog.

tip

If you've just installed a plug-in, you might have to close and restart your FileMaker Pro client to gain access to its external functions.

An exception is the case in which a plug-in is downloaded and installed by the Auto Update function. When this occurs, the plug-in is immediately enabled for use. For a full discussion, **see** *Chapter 27, "FileMaker Server and Server Advanced."*

Troubleshooting

Plug-in Not Responding or Not Installing

My plug-in isn't working. Where do I start to diagnose and fix the problem?

Issues with plug-ins can be difficult to troubleshoot. If a plug-in isn't responding, check first to see that you have the latest version, and make sure that it is enabled on your client computer. Restarting FileMaker Pro (or Developer) after adding a plug-in to the Extensions folder is also a necessary first step.

Beyond that, your testing has to encompass the functionality of the plug-in itself. You might, for example, be struggling with a plug-in written for another version of FileMaker Pro. Some plug-ins are limited to specific versions of FileMaker.

File Reference Errors

I renamed my files but still seem to have problems with missing files. How can I manipulate my file references by hand?

If you encounter problems with file references, where you get "file missing" error messages when FileMaker Pro opens your database solution, or you notice such in your DDR, we recommend first working with the Define File References dialog (Manage, File References, under the File menu). You might be able simply to repoint a file reference to repair some issues. In other cases, you might have to reestablish connections manually; to identify all such places where that will be necessary, refer to the DDR.

Again, this is a symptom of a file having been manually renamed at the OS level. The Rename Files function in FileMaker Developer is a great way to rename files all you like; we encourage you to use it whenever this is necessary.

This page intentionally left blank

27

FILEMAKER SERVER AND SERVER ADVANCED

About FileMaker Server

You use FileMaker Server to make your FileMaker Pro databases available to many users at once across a network. On its own, the FileMaker Pro software can host files for networked access from up to ten users at a time, in what's called a *peer-to-peer* configuration. The stability, security, and management features of FileMaker Server make it a preferred solution even for many environments that could manage with peer-to-peer solutions. However, small shops (particularly those with part-time FileMaker use) can function quite effectively with peer-to-peer configurations. As you learn in this chapter, the actual sharing is accomplished by one user running FileMaker Pro and allowing up to nine other users to connect to it, for a grand total of ten users.

That configuration handles the needs of many small businesses. Because it is ten users at a time, many small businesses with as many as a dozen regular users of FileMaker Pro can manage with the restriction; small businesses with much larger staffs, not all of whom use FileMaker, can also handle this restriction. Using FileMaker Pro allows you to use Instant Web Publishing for up to five users.

However, for larger environments, serious web publishing, or environments where automated management of databases and backups are needed, FileMaker Server is what you need.

No discussion of FileMaker Server would be complete without mentioning Apple's iOS devices (iPad, iPhone, and iPod touch) along with FileMaker Go, which runs on all of those devices. FileMaker Pro or FileMaker Pro Advanced is the tool you use to create sharable FileMaker databases.

As you have seen in Chapter 4, "Working with Layouts," and Chapter 14, "Advanced Interface Techniques," FileMaker Pro now provides extensive support for mobile devices. Many of the Starter Solutions come with alternate layouts for desktop and mobile devices

The FileMaker Server Product Line

Two products are available under the name FileMaker Server:

- **FileMaker Server**—FileMaker Server is used to provide concurrent access to as many as 250 networked users running FileMaker Pro client software; it also provides Custom Web Publishing (but not Instant Web Publishing) for 200 users.
- **FileMaker Server Advanced**—This product supports an unlimited number of FileMaker Pro clients (in practice, the limit is determined by the hardware configuration). In addition, it supports up to 200 XML or PHP web publishing clients, along with up to 100 Instant Web Publishing clients and ODBC/JDBC clients.

*For a discussion on ODBC and JDBC, **see** Chapter 23, "Exporting Data from FileMaker."*

*To find out about Instant Web Publishing, **see** Chapter 24, "Instant Web Publishing."*

Custom Web Publishing is discussed in Chapter 25, "Custom Web Publishing with PHP and XML."

To upgrade from FileMaker Server to FileMaker Server Advanced, you must purchase a new license key that unlocks the FileMaker Server Advanced functionality; there is no installation to be done.

What Is a Server? What Is a Network?

Server has at least three meanings in this chapter; as a result, it is usually qualified as a FileMaker Server, file server, or network server.

FileMaker Server is the simplest term: It is the software product from FileMaker that lets you share databases for more than ten users at a time, the limit for FileMaker Pro. It runs on a computer referred to as the *FileMaker server* (lowercase *s*) or, more awkwardly but precisely, "the computer running FileMaker Server."

A *file server* is a computer on a local area network on which people share files. It can be used for shared storage of corporate documents, as a backup location for individuals, or any other purpose. A network can have more than one file server.

A *network server* is a computer that manages a local area network. It might have shared network applications on it (such as mail). Examples of network server products are OS X Lion Server and Windows Server.

In a small environment, all three computers (FileMaker, file, and network servers) can be the same computer. It can even be a computer on which someone also does ordinary work by running word processing or other applications. This configuration really stresses the hardware, but it is a common situation in development environments where it is important to have all the features of all the computers available, but there is not a great deal of processing going on.

Any other configuration is possible. In fact, if you are using the web publishing features of FileMaker Server Advanced, you can run that set of tools on a separate computer from the FileMaker Server computer.

The best performance of FileMaker (and of networking) comes when FileMaker Server and the network software are the only applications running on their respective computers. The reason is that both FileMaker and many networking tasks are numerous but relatively brief in duration: querying a database, sending an email, downloading a web page, and so forth. Most of the clients of these computers want action as soon as they click or tap a button or press the Enter key, and then they disappear from the server computer environment while they type or read.

If you use some combination of computers and servers, make every possible effort to turn off file sharing on the computer running FileMaker Server. It provides its own sharing, and there can be corruption of databases if ordinary file sharing is turned on.

As for the second question, "What is a network?," the answer is this: a local area network, a wide area network, or the Internet. FileMaker can communicate over any TCP/IP network. You can even use peer-to-peer database hosting over the Internet; if you have a broadband connection, performance can be quite satisfactory.. Furthermore, any of the links in the network can be wireless.

FileMaker Server Versus Peer-to-Peer Database Hosting

With peer-to-peer sharing, FileMaker database files may be served to no more than nine clients at a time. The peer-to-peer method uses a regular copy of FileMaker Pro or FileMaker Pro Advanced as the database host, so a deployment of this type also forgoes important features of FileMaker Server, especially the capability to make regular, scheduled backups of the databases. Although such schedules could be created with operating system–level scripting technologies, it's much simpler to use FileMaker Server's built-in tools.

If you choose to begin with a peer-to-peer configuration for database sharing, we recommend that you still treat this situation as a server-type deployment as far as possible. Give the database host its own dedicated machine on which to run—one that people won't casually use for other daily tasks; make sure that you have a reliable solution for regular backups. Make sure that the machine at least meets the minimum specifications for the FileMaker Pro client software and add a bit more RAM if you possibly can.

Backing Up Open Files

If you're backing up hosted FileMaker files by hand, be aware that you should never make a copy of a FileMaker file while it is open—even if it's not hosted and is in use by only a single user. FileMaker can guarantee that a database file is in a fully consistent state on disk only if the file has been closed properly by the server process. Otherwise, there might be database transactions that exist only in RAM that have not yet been committed to disk.

As you'll read in a later section, FileMaker Server's built-in backup capability handles the details of closing the files before backing them up. If you're working in a peer-to-peer setting, you don't have that luxury. You'll need to make sure that any automated solution you put into place takes into account the need to close each database file before backing it up.

It is common to begin with peer-to-peer networking and then to move on to a FileMaker Server installation. The host computer's software will change, but all the users of the database will connect in the same way: by choosing the Open Remote command. They might have to select a different computer from the list of local hosts, but the new server can be added to the favorites list. Users do not normally know whether they are connecting to FileMaker Pro in a peer-to-peer environment or to FileMaker Server.

FileMaker Server Capabilities

We've talked about some of the features that set the FileMaker Server product line apart as a hosting solution: much greater scalability than the plain FileMaker Pro software and the capability to perform automated tasks such as backups. There are quite a number of other distinguishing features as well. Here are some of the most important:

- **Admin Console**—FileMaker Server comes with Admin Console, a Java application that can be used to administer one or several instances of FileMaker Server, potentially all running on different machines. A separate copy of Admin Console is created for each FileMaker Server instance to be managed. Subject to security constraints, Admin Console can run on any computer that can connect to the server computer over a network.
- **Consistency Checker**—FileMaker Server performs consistency checking on files as it opens them. If the check fails, a message will be written to the application log and the file will not open.
- **Scheduling**—You can create schedules to run automatically to back up or verify databases, send messages, or run scripts.
- **Server-side scripting**—FileMaker Server can run FileMaker scripts as well as operating system scripts. You can schedule these within FileMaker Server. A particularly common use of these is to periodically run an export script that puts files into a known location (perhaps using `Get (DocumentsPath)` where they can be retrieved by other applications.
- **SMTP email**—FileMaker Server can send email on its own directly from an SMTP server rather than through a mail client that in turn accesses the SMTP server.
- **Email notifications**—You can configure FileMaker Server to provide email notifications of conditions and status to one or more email addresses. Thus, instead of having to check the server status, the server status will come to you.
- **External authentication**—FileMaker Server can be configured to check user credentials against a networked authentication source, such as a Windows Active Directory server or an OS X Open Directory server.
- **Secure transfer of data**—When FileMaker Pro clients are used in conjunction with FileMaker Server, the transfer of data can be encrypted with Secure Sockets Layer (SSL).

In addition to these features, FileMaker Server offers a large number of other important functions, such as the capability to send messages to guests, to disconnect idle guests, to limit the visibility of database files based on user privileges, to be run in a scripted fashion from the command line, and

to capture a variety of usage statistics and server event information for logging and analysis. All these features are discussed in the sections to come.

FileMaker Server Requirements

Like any piece of server software, FileMaker Server has certain minimum hardware and software requirements. You'll achieve the best results with a dedicated server; as with any piece of server software, it's best if FileMaker Server is the only significant server process running on a given machine. Forcing FileMaker Server to compete with other significant processes, such as mail services or domain controller services, is likely to hurt Server's performance.

The server machine, in addition to being dedicated as far as possible to FileMaker Server, and having the minimum amount of file sharing enabled (preferably none), also needs the items discussed in the following sections.

Web Server

FileMaker Server requires a web server. On Windows, this is IIS; on OS X, it is Apache. You might have to install and configure IIS before beginning the FileMaker Server installation process. On OS X, Apache is part of the standard installation; you might have to start it if you have not enabled it before. And if it has been removed, of course, you will need to reinstall it.

The web server will need PHP; you can use a version of PHP on your own web server, or you can have it installed as part of the FileMaker Server installation process (it does not require any extra discs or licenses).

Static IP Addresses

Clients must be able to connect to the server computer. It must either have a static IP address or a domain name that is set to a static IP address. If you are running a local area network, it is quite possible that a single Internet connection to your router is shared among all the computers. The router will have an IP address visible from the outside. This might or might not be static. In the case of cable connections, it is frequently renewed once a day with the same or a different address.

FileMaker Server is capable of *multihoming*, meaning that it can take full advantage of multiple physical network interfaces, each with its own IP address. FileMaker Server listens on all available network interfaces. As far as we know, it's not possible to configure FileMaker Server to ignore one or more of the available interfaces; if the interface is available, FileMaker Server tries to bind to port 5003 on that interface and begins listening for FileMaker traffic. The FileMaker client/server port number, 5003, is also not configurable.

note

The computers on the local network share that one changing IP address, but they might have a static IP address beginning with 192.168 or 10.0. This is determined by the configuration of the network server. It is under your control, not the control of your ISP. If the only access to your server computer is internal, you can provide it with a static internal IP address. However, if it has to be accessed from the outside, you must work with your ISP to provide it with a static IP address that, like all IP addresses on the Internet, must be unique.

Fast Hard Drive

Like any database, FileMaker Server is capable of being extremely disk-intensive. For some database operations, particularly those involving access to many records—such as a large update or a report—the speed of the server's hard disk might be the limiting factor. Redundant Array of Inexpensive Disks (RAID) technologies (whereby multiple physical disks are combined into a single *disk array*, for greater speed, greater recoverability, or both) are becoming ever cheaper, and some sort of RAID array might well be the right answer for you. When it comes to FileMaker Server performance, buy the biggest, fastest disk you can.

note

At the time of this writing, a RAID system with two 1TB drives providing RAID 0 support of 2TB or RAID 1 support (redundancy) of 1TB can be purchased for less than $500.

Fast Processors

A fast processor is a fairly obvious requirement for a server machine. But it's worth noting that FileMaker Server can take full advantage of multiple processors.

Lots of RAM

Beginning with FileMaker Server 12, the Database Server and Web Publishing Engine are now 64-bit processes that can use much more RAM than their predecessors. There is a single installer for OS X that contains both 32- and 64-bit software; on Windows there are two installers.

Turn Off Unnecessary Software

You do not want the computer running FileMaker Server to sleep, hibernate, or go into standby mode. A screensaver is also unnecessary—most of the time, your server computer will not even need a monitor, and the simplest screensaver of all—turning off the monitor's power switch—is the best.

Indexing Service (Windows) and Spotlight (OS X) are great tools to help you find information on your computer, but they use resources in the background—both processor power and disks, both of which are needed for FileMaker Server. If the computer running FileMaker Server does not need Indexing Service or Spotlight, turn off those options. These settings apply to all types of servers.

You also should disable antivirus software on the folder where the database files are stored (but only that folder).

Fast Network Connection

FileMaker is a client/server application, which means that FileMaker Pro clients remain in constant contact with a database host such as FileMaker Server. FileMaker Server constantly *polls* (attempts to contact) any connected clients to determine what they're doing and whether they're still connected. In addition, although Server is capable of handling a few more tasks than its predecessors, it still has to send quite a lot of data to the client for processing in certain kinds of operations. All this means that FileMaker is an extremely network-intensive platform that benefits greatly from increased network speed. A switched gigabit Ethernet network will provide good results.

Supported Operating Systems

FileMaker Server runs on OS X and Windows.

*For the latest OS X and Windows version information, **see** www.filemaker.com.*

Java Runtime Environment

You must have the Java Runtime Environment installed. If you do not have it installed, you will be prompted to allow it to be automatically installed. You do not need any extra discs.

Data Center Environment

Although not strictly a requirement for running FileMaker Server, proper care and housing of server equipment is a necessity, one that's often overlooked, especially in the small- and medium-sized business sectors, some areas of education, and among nonprofit groups. These are all key groups of FileMaker users, ones that do not always have sufficient resources to build and maintain anything like a data center.

Ideally, a server of any kind should be housed in a physically secure and isolated area, with appropriate cooling and ventilation, with technical staff on hand 24 hours a day to troubleshoot any issues that arise, and with automated monitoring software that periodically checks key functions on the server and notifies technical personnel by email if any services are interrupted. Some organizations are fortunate enough to be able to house their FileMaker servers in such an environment. But even if you can't provide all those amenities, you can see to the key areas. The server should minimally be up off the floor, well ventilated, and under lock and key if possible. And some sort of monitoring software is nice, and need not break the bank: Nagios (www.nagios.org) is a popular and powerful open source monitoring package.

note

Nagios runs on UNIX but can monitor servers running on almost any platform. Many server monitoring packages exist for Windows deployment as well.

External Data Centers

A number of companies provide FileMaker hosting. Search the FileMaker website for "FileMaker hosting" to see a list. You can use a shared copy of FileMaker Server running at a remote site to run your databases and support your web publishing. The vendor will provide you with the tools to upload your databases and open them. Users will connect using your domain name or the IP address of the FileMaker host. You do not have to purchase FileMaker Server because your monthly payment reimburses the hosting company for its purchase of the software. Your monthly payment also covers its data center environment, backups, and monitoring.

Installing and Deploying FileMaker Server

The process of installing and deploying FileMaker Server is different from installing software such as a word processing application or even FileMaker Pro. FileMaker Server runs in the background

and has no user interface; you interact with it using Admin Console, a Java application that runs on the server computer or any other computer that has network access to the server computer.

caution

Step 1 in the installation process is to uninstall any previous version of FileMaker Server. The easiest way to do this is to use the original distribution discs or disk images from which you did the installation. Follow the instructions in the Getting Started guide and through the links it contains to the FileMaker website. You will need to stop FileMaker Server itself and the Web Publishing Engine (if installed and running), run the uninstall process, and restart the server computer. If the server computer is running a previous version of FileMaker Server or any other applications, you should know that a restart might be required as part of the uninstall process. Unless you are starting an install on a computer that is currently not running any networked applications, you are probably going to find yourself doing this at night, on a weekend, or on a holiday. Doing this when the production environment can be stopped is helpful. It might take only you an hour to install, deploy, and configure FileMaker Server, but that hour will go much faster if users are not poking their heads into the room or sending you text messages asking, "How much longer?"

The Installation Process

On Windows, all files are installed in a directory called Program Files\FileMaker\FileMaker Server. On the OS X, the FileMaker Server components are installed in /Library/FileMaker Server. The default install location can be changed on Windows, but not on the OS X.

Installation is handled by an automated process. All that you have to do is to enter your name and license code. Note that the license code company name must exactly match the name on the license you have received from FileMaker. If there are any misspellings or mistakes in the name, you must either correct them with FileMaker or grit your teeth and enter them in their incorrect version during the install process.

note

After FileMaker Server is installed and deployed, you can manage it using Admin Console on any computer that has network access to the FileMaker Server computer. For the initial installation and deployment, however, you need hands-on access to the FileMaker Server computer. If it normally runs without a keyboard and monitor, attach them before beginning the process. When everything is complete, you can turn off the monitor or even detach it.

FileMaker Server Configurations

FileMaker Server can run in several configurations. There are three components to deal with:

- **Database server**—This is the FileMaker database engine. It controls access to the databases. In FileMaker Server terminology, whatever computer is running the database server is the *master* computer. All others (if any) are *worker* computers.
- **Web Publishing Engine**—This is the FileMaker software that enables Custom Web Publishing and Instant Web Publishing. Instant Web Publishing is available only for FileMaker Server Advanced, but Custom Web Publishing is available in both products.
- **Web server**—This is IIS (Windows) or Apache (OS X). The web server should be configured and set up before you begin the FileMaker Server installation, but you can come back later to com-

plete the configuration if you want. It is the Web Server module that lets the Web Publishing Engine talk to the web server. Also, as part of the FileMaker Server installation process, PHP might be installed. If it is already installed on your web server, you can use that version as long as it meets the compatibility requirements for your version of FileMaker Server.

You can install these components on one, two, or three computers. Because the computers communicate using standard protocols, it does not matter which operating system is used on which computer in a multicomputer environment. Here are the configurations you can use:

- **Single computer**—The database server and the Web Publishing Engine on a single computer along with your web server. This is obviously the simplest installation, and it might be the best one to start with. Certainly, if you are creating a test environment to explore FileMaker Server, it is a logical place to start. This configuration is recommended for no more than 50 FileMaker Pro clients.
- **Two computers: web/database**—With two computers, you can put all the web components on one computer, leaving the database server alone on the other computer. This is generally the highest performing configuration.
- **Two computers: FileMaker/web server**—Another configuration places the main FileMaker components (Web Publishing Engine and database server) together on one computer. The only software that must be installed on the web server is the Web Server module. This configuration might be advisable or even required where the web server is used to provide other services in addition to FileMaker.
- **Three computers**—Each of the components is on its own computer. This is suitable for very heavy loads because the web server does not have to compete for resources with anything else. Tuning the web server's performance might be easier in this environment. Note that because there is intercomputer communication between the web server and the Web Publishing Engine, this configuration can in some cases provide slightly poorer performance than a two-machine configuration.

Except in the very unusual situation in which everything is installed on a single computer and the only clients accessing FileMaker Server are on that same computer (an environment usable only for testing), there has to be communications among computers managing FileMaker Server, Admin Console, the FileMaker Pro clients, and the clients of the web server. These communications should be protected by firewalls on the various computers. Table 27.1 provides a list of the ports, their purposes, and which computers need access to them.

Firewalls

Whenever you are dealing with a network connection, you are usually dealing with firewalls and ports (if you are not, you are running a major security risk). Table 27.1 provides the official FileMaker Server port list. If you open these ports before you install FileMaker Server, things will go faster. The users of the ports are the Web Server (WS), Database Server (DS), Web Publishing Engine (WPE), Admin Console (AC), and FileMaker Pro clients (FPC).

Table 27.1 FileMaker Server Firewall Ports

Port	Used For	WS	DS	WPE	AC	FPC
80	HTTP	X				X
5003	FileMaker sharing		X			X
*16000	HTTP	X				X
16001	HTTPS for Admin Console		X		X	
16004	Admin Console	X	X	X	X	X
16006	FileMaker Server			X		
16008	FileMaker Server			X		
16010	Custom Web Publishing			X		
16012	FileMaker Server			X		
16014	FileMaker Server			X		
16016	Apache Jakarta Protocol			X		
16018	Apache Jakarta Protocol			X		
50003	FileMaker Server service/daemon		X			
50006	FileMaker Server service/daemon		X			

** 16000 is also used for streaming container data in FileMaker 12 and later*

Selecting the Configuration

When you begin the installation process, the installer will present the window shown in Figure 27.1 to begin configuration. Choose whether this will be a single-machine or multiple-machine installation.

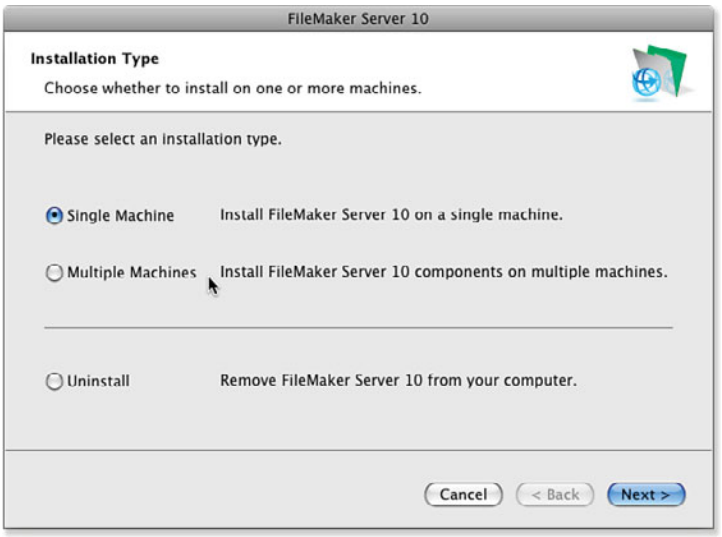


Figure 27.1
Choose a single- or multiple-machine configuration.

If you choose a multiple-machine configuration, you will need to install FileMaker Server on each computer. The window shown in Figure 27.2 lets you specify for each computer whether it is a master computer (with the FileMaker Database Server installed on it) or a worker.

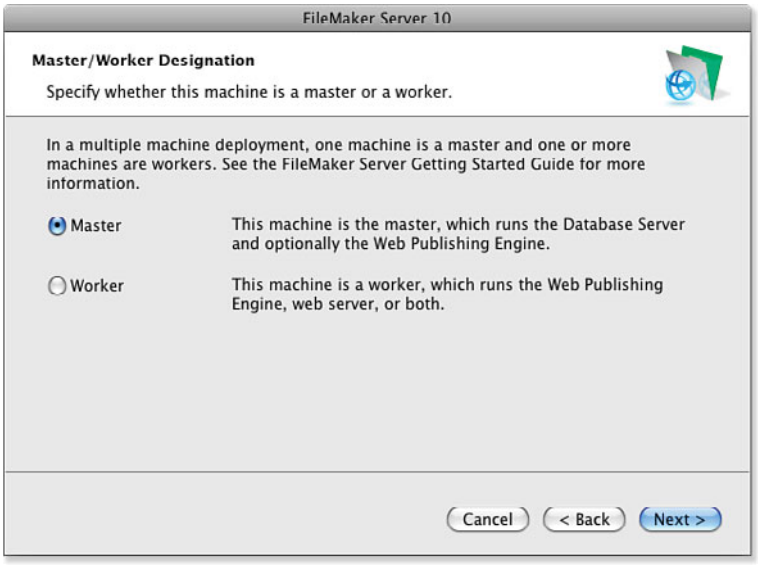


Figure 27.2
For multiple-machine configurations, install FileMaker Server on each computer.

Along the way, you might be asked to confirm various steps in the process, such as accepting a security certificate. These are all normal parts of any installation. With regard to security, the dialogs you see should identify FileMaker as the origin of the software, and if that is the case, you are okay to continue.

The Deployment Process

After you have completed the basic installation process, you will be prompted to continue on to the Deployment assistant. If you elect not to do so, pick up at this point by choosing Start, Programs, FileMaker Server, FMS 12 Start Page (Windows) or by double-clicking the FMS 12 Start Page shortcut that was installed on the desktop. The FileMaker Server Admin Console start page shown in Figure 27.3 will appear. It gives you an opportunity to manually start Admin Server if it has not started automatically.

tip

In part because the Deployment assistant and Admin Console are Java applications and you might not have run Java applications on your computer before, a little fiddling could be required to get them to run. You can find troubleshooting tips at the end of this chapter, and the Getting Started guide that is installed as part of the FileMaker Server documentation is an invaluable resource. You can find additional information on the FileMaker TechNet site. After the installation and deployment process is done, you should not have to worry about these issues again.

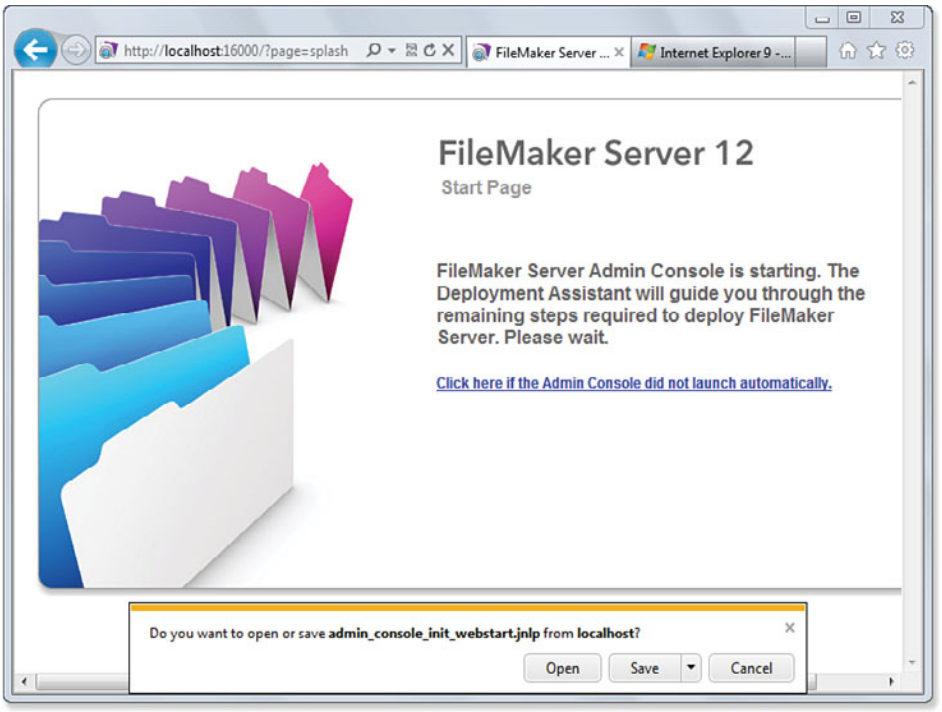


Figure 27.3 Admin Console start page.

The Admin Console Start Page will appear. It gives you an opportunity to manually start Admin Server if it has not started automatically.

Along the way, you might see messages asking you if you want to allow access to specific resources such as the Java application. If you are confronted by messages such as these, remember that you have started a process of installation and deployment of a FileMaker product, so a message asking if FileMaker is a legitimate provider of software should be answered Yes or Always. If you launch an installation of FileMaker and see a message asking you to approve installation of software from another vendor, you might get suspicious—although Java, which is not a FileMaker product, is installed. If you get a warning of software from some other vendor or a name you do not recognize, you might want to contact FileMaker customer support.

There are five or six steps, depending on whether you are doing a single-machine configuration. All of these settings can be changed later.

- Set up the Admin Console account with an ID and password as shown in Figure 27.4. If several people are administering the installation, they can share this account. You also will be able to set up groups of administrators with their own security settings. Groups are available only with FileMaker Server Advanced.
- You have the option to import settings from a previous installation, as shown in Figure 27.5.

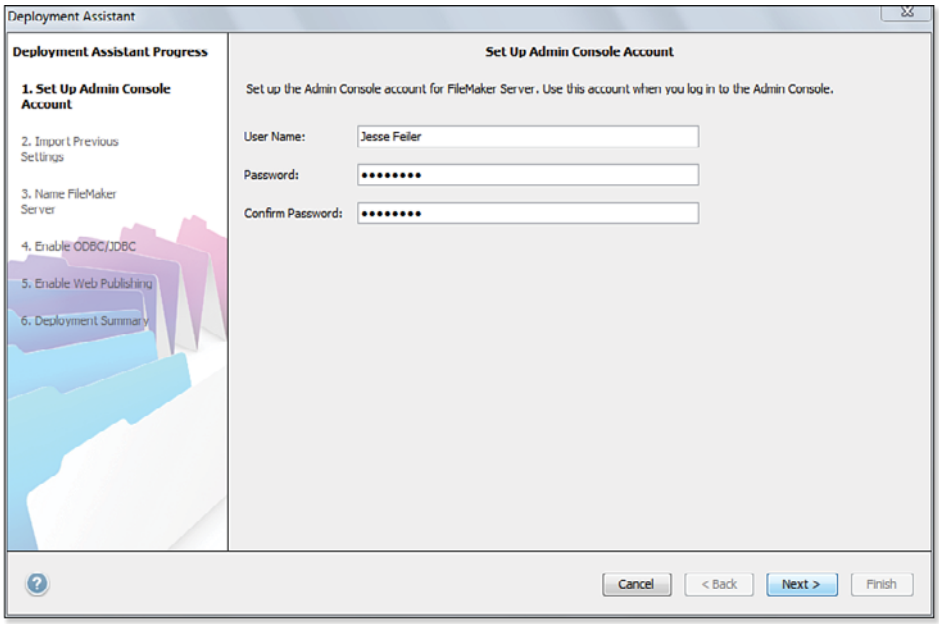


Figure 27.4
Set up the Admin Console account.

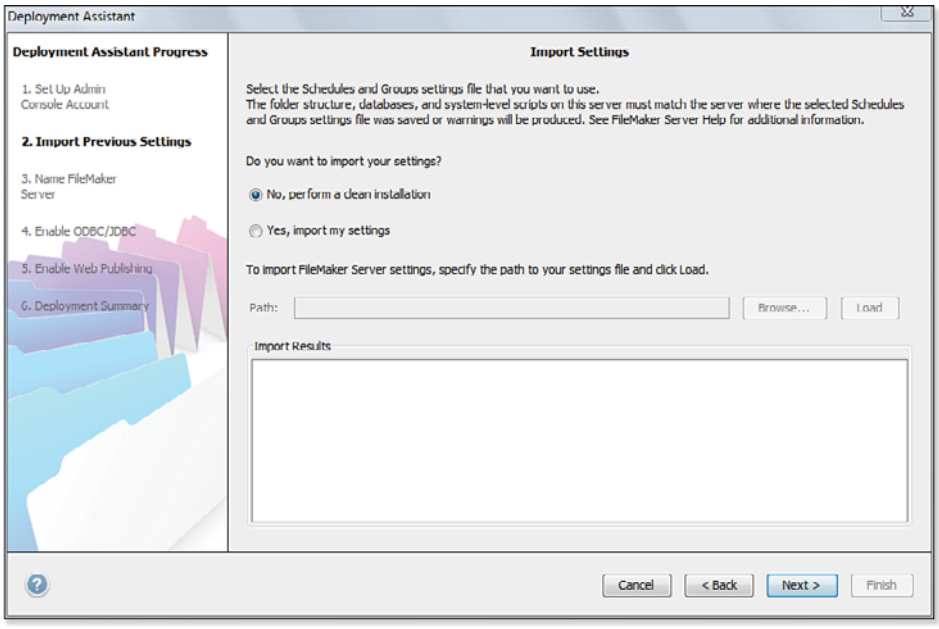


Figure 27.5
You can import settings.

- Name this particular FileMaker Server installation, providing a brief description and the name and contact information for the person responsible, as shown in Figure 27.6.

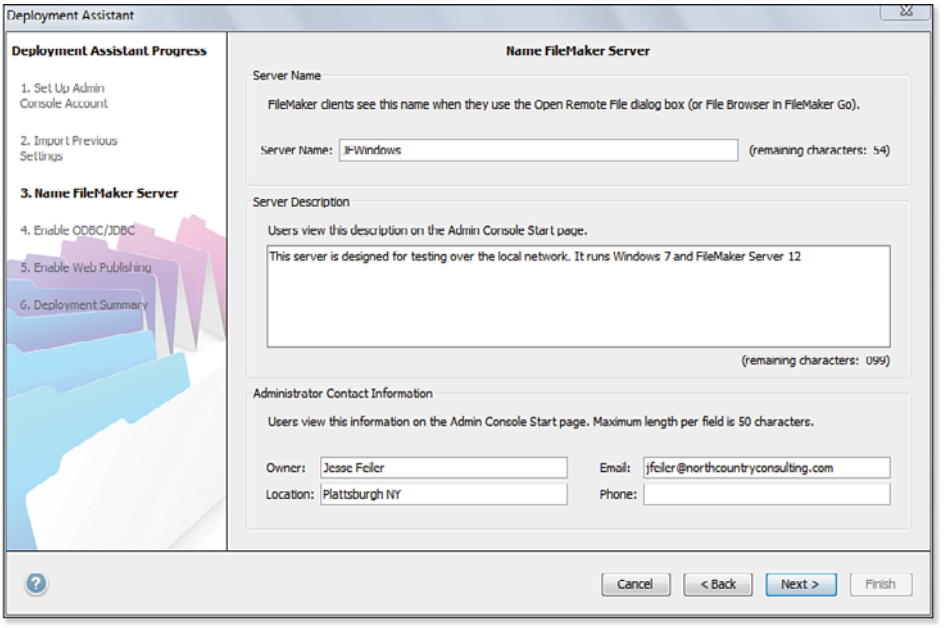


Figure 27.6 Name the FileMaker Server.

- You can enable ODBC/JDBC publishing if you are using FileMaker Server Advanced, as shown in Figure 27.7. You do not need to enable ODBC/JDBC publishing to access ODBC/JDBC data published elsewhere that you and your users want to consume. This interface is simple: Click Yes or No.
- You are asked whether you want to enable web publishing. If you agree, choose which web publishing technologies you want to use (XML, Instant Web Publishing [IWP requires FileMaker Server Advanced], and PHP), as shown in Figure 27.8.

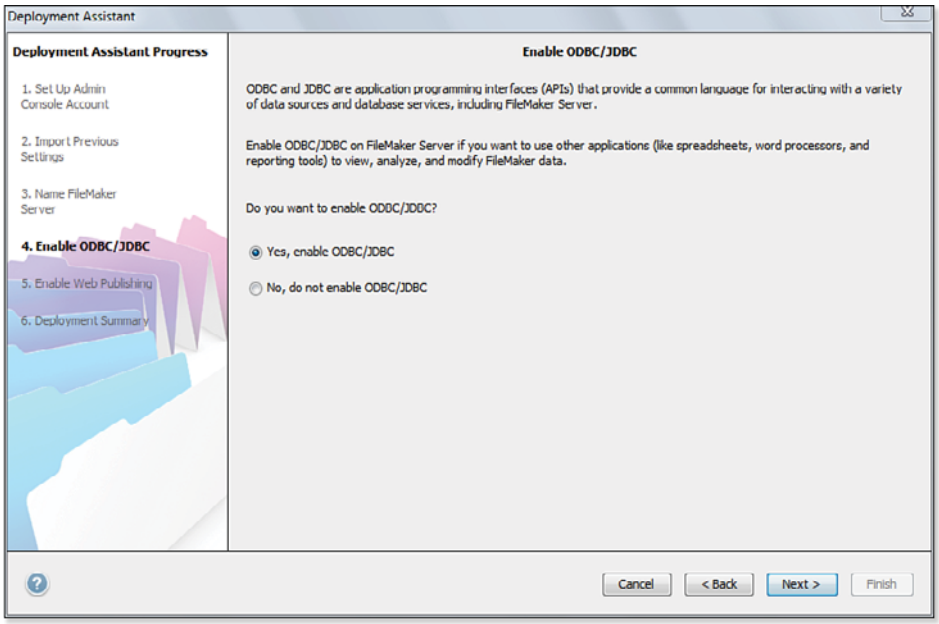


Figure 27.7 Enable ODBC/JDBC if you want.

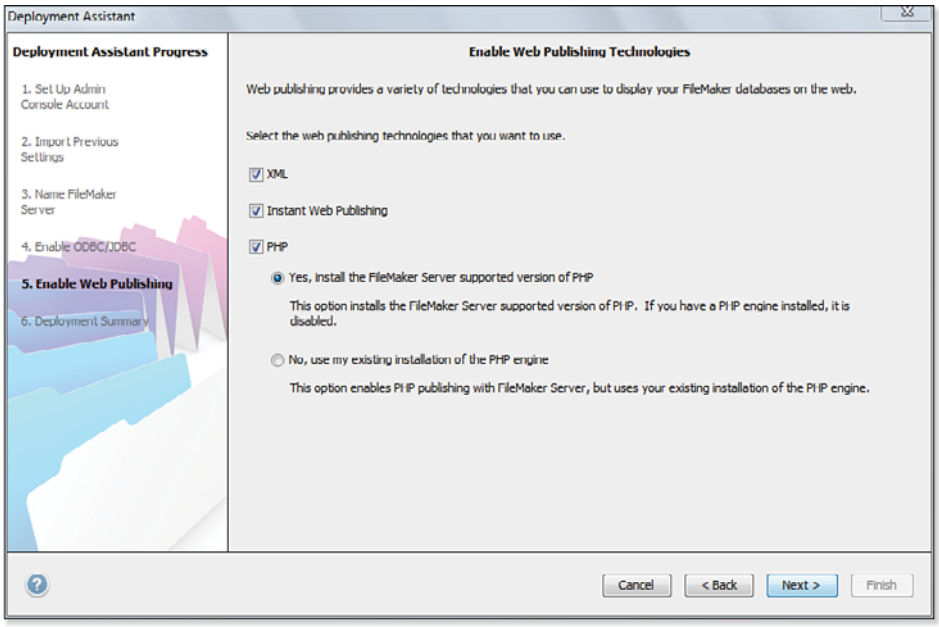


Figure 27.8 Choose your web publishing technologies.

- You can then choose your web server (or choose to select it later), as shown in Figure 27.9.

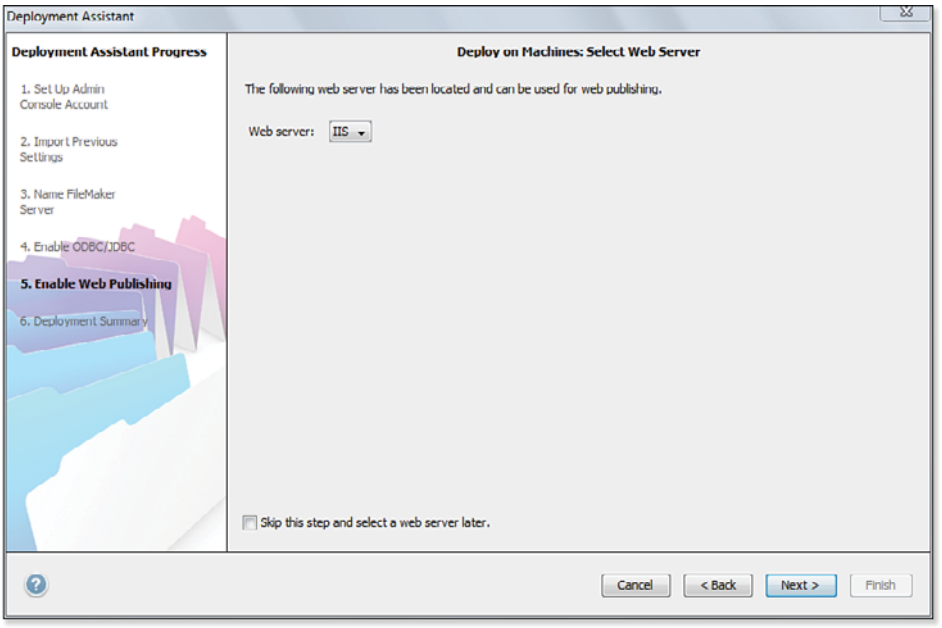


Figure 27.9 Select your web server.

- You can choose which of the configurations you want to install, as shown in Figure 27.10. The deployment type is set in the pop-up menu at the top of the window.
- A final summary screen reviews your choices, as shown in Figure 27.11. You can use the Back button to go back and change them. You also can rerun Admin Console at another time and change many of them. (Note that the initial installation of the software for the master and worker computers, as well as the deployment type shown previously in Figure 27.10, cannot be changed later.)

tip

If the web server is not running (perhaps it is installed but not running), you cannot complete this step and will have to come back to it.

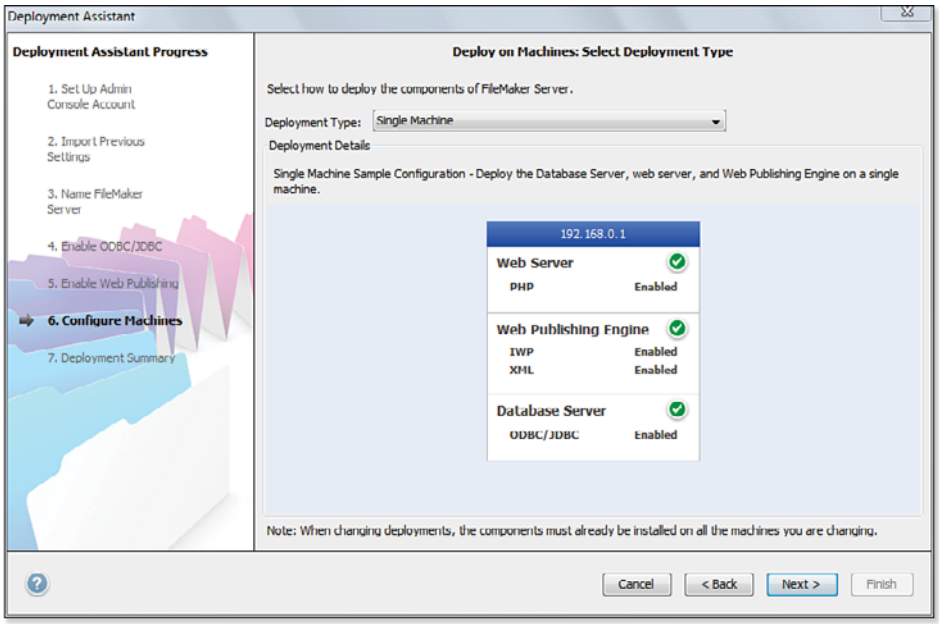


Figure 27.10
Choose your configuration.

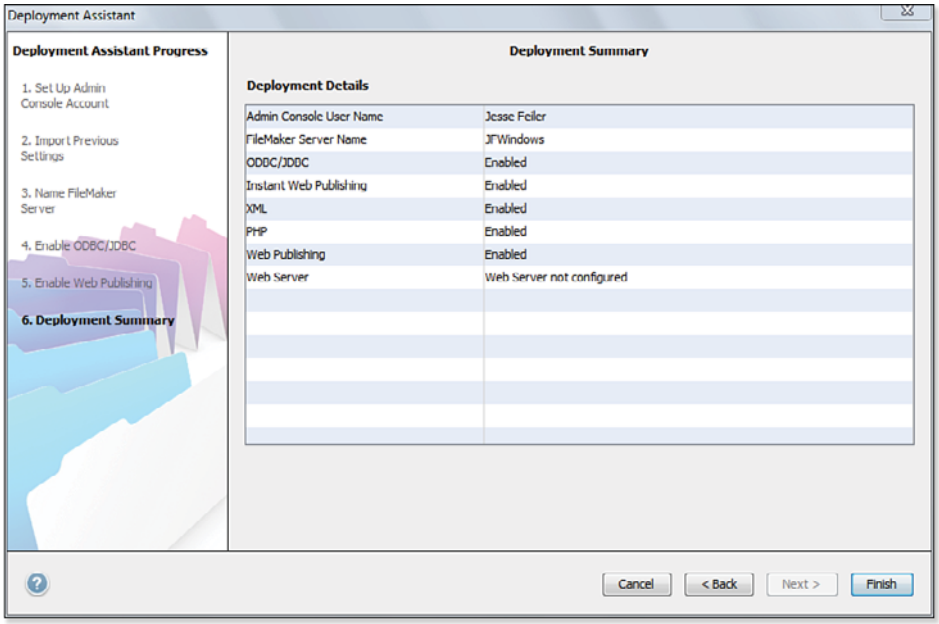


Figure 27.11
Review the installation summary.

Figure 27.12
Run the tests.

You will be provided with information about the progress of the deployment. When it is complete, you will be invited to register (a good idea) and to run the technology tests (a critical step), as shown in Figure 27.12

At this point, FileMaker Server is installed and deployed. You can use the Admin Console on any computer with network access to the FileMaker Server computer to administer it from now on.

Running FileMaker Server

Installing FileMaker Server installs two separate components, both of which run as services: FileMaker Server and the FileMaker Server Helper. They appear as two separate services (Windows) or processes (OS X). FileMaker Server does not function correctly without the FileMaker Server Helper service also running. Installing FileMaker Server Advanced will cause additional services to be added.

Starting and Stopping FileMaker Server

When you install FileMaker Server, you can choose whether to have these services start automatically, in which case they are started every time the server computer itself starts up, or manually, in which case you need to start the services by hand. The Admin Console lets you start and stop FileMaker Server manually.

tip

If you will not be using a monitor or keyboard on the FileMaker Server computer, it is a good idea to take two additional steps before removing them. First, verify that any automated power equipment is working properly to avoid interruptions. In the case of an extended outage, the uninterruptible power supply (UPS) should shut down the server computers gracefully. Make certain that you can power them back on again successfully and that the databases open properly. If there are any problems, check them out before disconnecting the monitor and keyboard. Likewise, make certain that you can access the computer using Admin Console from another computer before disconnecting the monitor and keyboard.

Hosting Databases

When FileMaker Server starts, it looks for files in the default database file directory, and in the alternate database directory if one has been specified. (We discuss how to specify the alternate directory later.) It also tries to open any databases found in the first directory level within either of those two top-level directories. Databases in more deeply nested directories are not opened. You can find the main database directory at c:\ Program Files\ FileMaker\ FileMaker Server\ Data\ Databases (Windows) and / Library/FileMaker Server/Data/Databases (OS X).

tip

Take care to place these directories on hard drives that are local to the server machine. It's not at all a good idea to host files from a mapped or networked drive. In such a configuration, every database access has to be translated into a network call and passed across the network. At the very least, this approach is likely to cause significant loss of performance.

note

In the world of databases, it is common to speak of *starting* and *stopping* databases, which is basically the same as opening and closing them. FileMaker Server often uses the open/close verbs, but if you are working with people from other environments, make certain you are all clear about what is meant by "stopping a database"—in other words, do you mean closing the database and leaving FileMaker Server running, or closing FileMaker Server and all its databases?

Using Admin Console

When you first install FileMaker Server, you might be prompted to install and open AdminConsole. You have a variety of choices about when to do this, and you will have prompts and options to place shortcuts on the desktop. At that time or thereafter, you can open Admin Console manually. To do so, open a browser on any computer with network access to the computer where FileMaker Server is running (even the same computer). Enter the IP address of that computer and port number 16000. Here are three formats for that URL:

```
http://localhost:16000
http://10.0.1.2:16000
http://www.mydomain.com/rex:16000
```

The first is used if you are running the browser on the same computer as FileMaker Server. The second is used to address the server computer by its IP address (either locally or over the Internet). The third is used if you have a domain name and have configured a name for the computer running FileMaker Server. In all cases, you use port number 16000.

FileMaker Server responds by sending a small Java application to your computer. You will be asked to log in, and the database will open. You also might be asked for permission to run it, as you see in Figure 27.13.

note

If you look carefully at Figure 27.13, you will see that FileMaker Server is running on a Windows computer but Admin Console is running on OS X. This (along with the reverse) is a common configuration.

Figure 27.13
Give permission to Admin Console.

FileMaker Server Overview

Figure 27.14 shows FileMaker Server Overview in Admin Console. It is the default screen, and you can always return to it by clicking FileMaker Server Overview at the left.

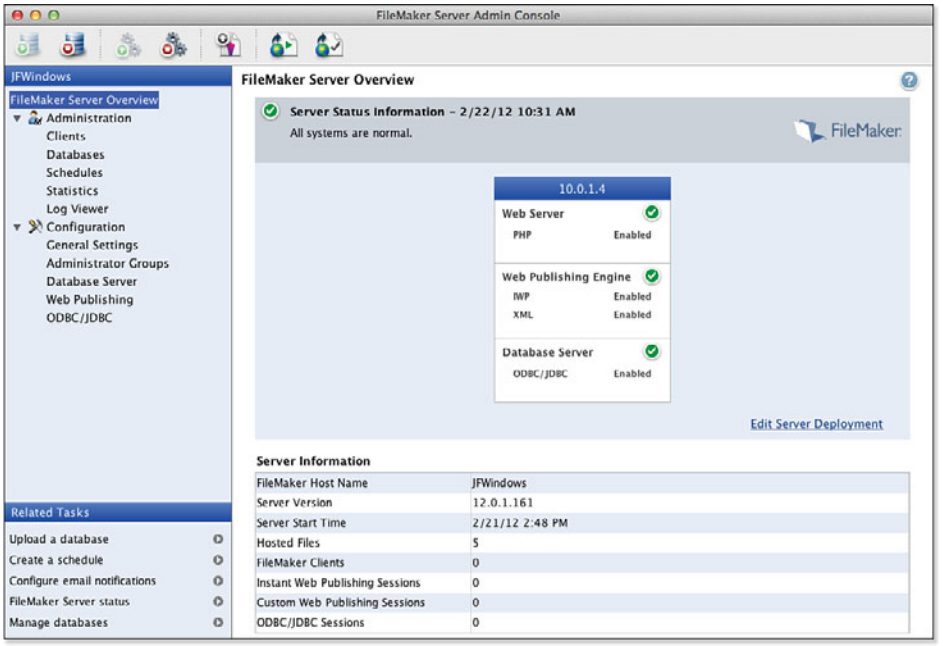


Figure 27.14
FileMaker Server Overview summarizes the status and settings.

All the windows in Admin Console have a similar layout. The navigation pane at the left lets you view and change settings. The center and right of the window contains detailed information about whatever you are viewing; in the lower left, links let you go to related tasks and documentation. At the top of the window, in the toolbar, are seven icons:

- The first two let you start and stop the database engine (not an individual database).
- The third and fourth let you start and stop the Web Publishing Engine (if it is installed).

- The fifth starts the Upload Database assistant.
- The sixth lets you view the FileMaker Server Start page in a browser.
- The seventh opens the test page; you can run any needed tests again.

The status overview at the center right shows what is running on what machine. If you have a two- or three-computer configuration, there will be a slight space between the boxes representing the machines. In Figure 27.14 (a single-computer configuration), the IP address of the single computer is shown. In a multiple-computer configuration, the IP address for each computer is shown.

You should know the IP address of the master computer, but if you do not, write it down the first time you see this display. It is the IP address people will need to connect with Admin Console (remember to add port 16000 to the IP address when you connect with a browser).

tip

FileMaker Server provides the ability to configure and customize your installation, whether it is large or small. If it is small, there are a few settings in Admin Console that you must know about; you can safely ignore the others unless something strange happens. Many people run FileMaker Server for years without changing the default settings. The minimal items to which you need to pay attention are shown in Tips in the following sections.

Minimum: Note the IP address and the status of each of the FileMaker Server components.

Administration

The Administration section lets you manage clients, databases, schedules, and statistics. A summary screen lets you move among them, or you can click each item in the navigation pane at the left.

Clients

You can see the Clients display in Figure 27.15. Select one or more clients and choose an action from the pop-up menu. Click Perform Action to do it. In general, it is not a good idea to disconnect clients; instead, send them a message to log off. However, if remote users do not respond (perhaps because they have gone out to lunch), you might need to disconnect them if you need to stop FileMaker Server.

The action options are

- Send Message (to the selected client)
- Send Message to All Clients
- Disconnect (the selected client)
- Disconnect All Clients

tip

Minimum: Unless your FileMaker Server environment is small and in a confined area, you will probably use the Clients section to send messages to your users.

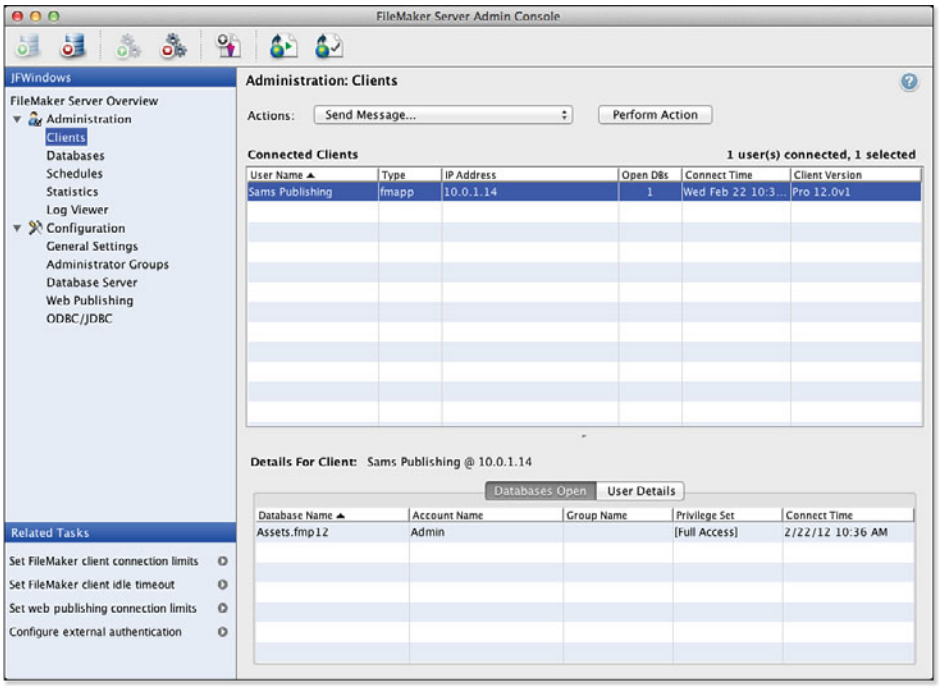


Figure 27.15 Manage clients.

Databases

The Databases section, shown in Figure 27.16, provides information for databases managed by FileMaker Server, whether or not they are open. A similar interface to that in Clients lets you send actions to the databases.

Dots indicate the features enabled in the databases. Note that these report the status; you set sharing and the other features in the databases themselves (in the Sharing submenu). This display is useful when you have first added a database; you should check that the correct features are enabled.

The action choices are

- Send Message (to the users of the selected database)
- Send Message to All (to the users of all databases)
- Open (the selected database)
- Open All

- Close (the selected database)
- Close All
- Verify (the selected database)
- Verify All
- Pause (the selected database)
- Pause All
- Resume (the selected database)
- Resume All
- Remote (the selected database)
- Upload Database

tip

You use the Databases section to open, close, and pause databases, as well as to send messages to database users without having to identify them on the Clients display. Pausing a database leaves it open, but flushes the cache and prevents reading or writing. It is most frequently used to create a copy of the database while it is technically open but not in the middle of processing. You can also right-click (Windows) or Ctrl-click (OS X) on the database file for a contextual menu. Resume is the command to leave the pause state.

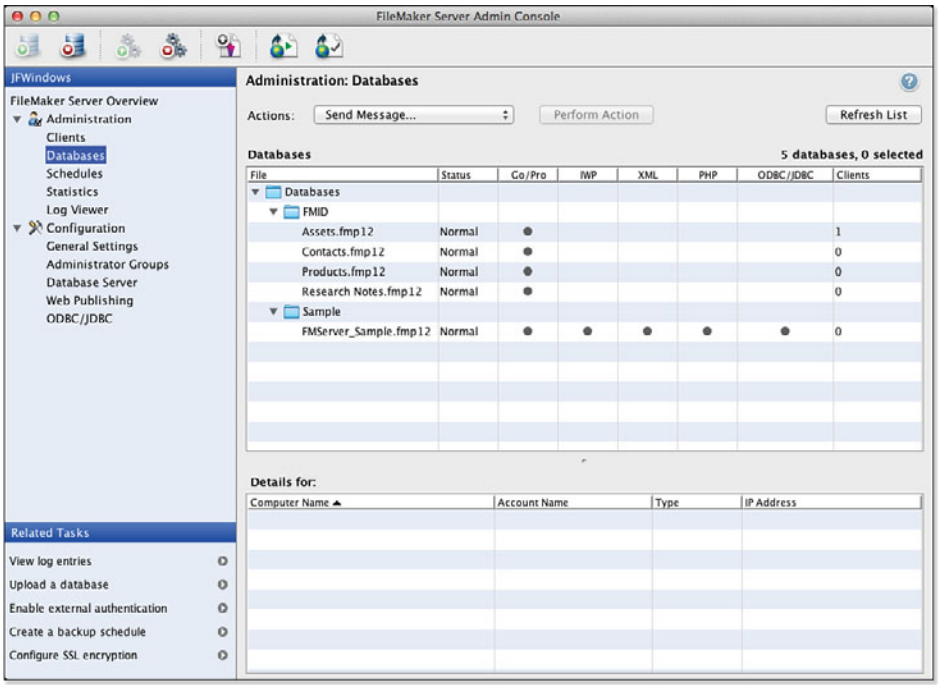


Figure 27.16
Manage databases.

Schedules

FileMaker Server includes a powerful scheduling feature, as shown in Figure 27.17.

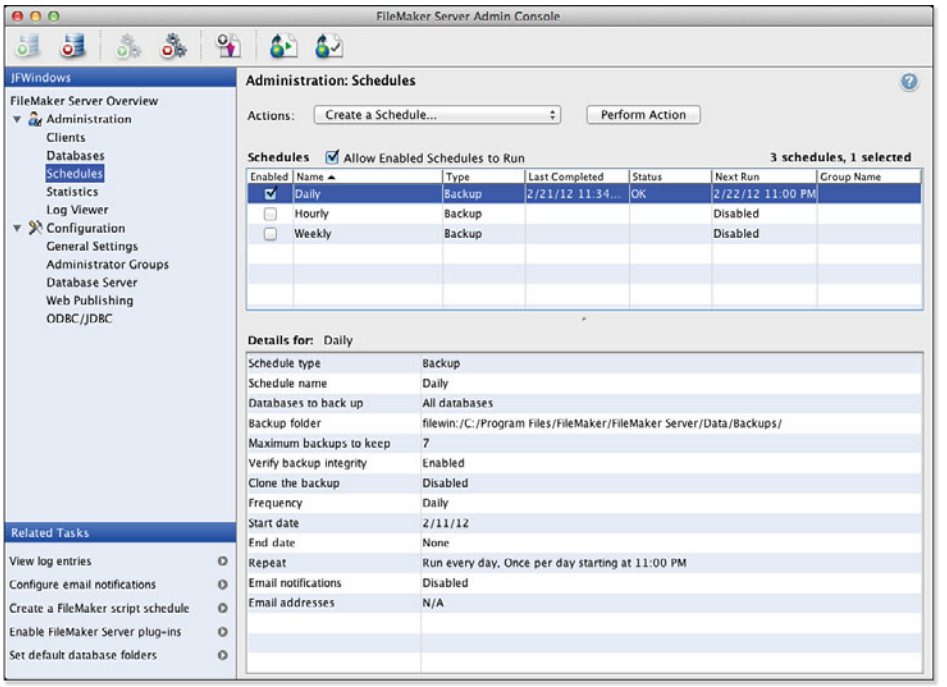


Figure 27.17 Schedule events.

There are four types of events you can schedule:

- Backups
- Verifications
- Execution of scripts and batch files
- Messages

Using the scheduling feature, you can create, duplicate, delete, or edit schedules; you also can select them and execute them manually. Note that you can enable each schedule with the check box to the left of its name.

tip

You can also enable all individually enabled schedules with the check box at the top of the list. This capability allows you to temporarily turn off all schedules without changing each one's status. You might want to do this for diagnostic purposes if you are experiencing slowness.

You will be walked through your choices when you create or edit a schedule. For a backup schedule, those choices are as follows:

- Select a task (backup, database verification, run a script, or send a message).
- Select from several predefined schedules (daily, weekly, and so forth).

- Select the databases to back up.
- Select the location to which they will be backed up. You also have a choice to verify the backup. This step takes more time, but it means that the backup copy is correct. If you can schedule backups to run overnight when there are few users of the system, this is a useful option. If you are automatically backing up databases, set your tape backup to back up the files from the backup folder, not the live databases. That way, you can avoid issues with backing up live databases that may temporarily be inconsistent.
- Choose the number of backups to be kept. They will be in a single time-stamped folder at the path you specified in the previous bullet. You also have the option to verify or clone the backups. That provides you with more security but it puts a greater burden on the server. If you can schedule the backups for the middle of the night, this may not be an issue.
- When you choose the predefined schedule, you are given the opportunity to refine or change the schedule, giving it a start and stop date, selecting specific days of the week (such as omitting weekends), and specifying how often each day the schedule should be run. You can also specify the number of backups to keep.
- Name the schedule.
- You can specify an email notification to be sent. If you have asked to verify the backups, the verification status will be shown in the email. Note that you can have several recipients for the email notification. What this can mean is that instead of someone having the task of checking the backup schedule in the morning, all that has to happen is a check of email—not an extra step involving the database.
- The last step is a summary of the schedule. This information is also shown at the bottom of the Schedules window shown previously in Figure 27.17.

tip

Set up a backup schedule, ideally at least once a day with an email notification. Back up the databases to a known location, and then, if you have an automated file backup, copy the backup files to another disk, tape, or whatever storage you are using. Make certain that the schedule of the file backup is set for a sufficiently long time after the database backup so that the files are created. For example, schedule the FileMaker database backup for 1 a.m. and the file backup for 5 a.m. Check periodically to see how long the FileMaker backup is taking (the email notification will help). Back up the folder that is created. FileMaker will name it with the backup and the date (Daily_2012_02-21-2334, for example). If you are backing up the backups to tape or some other storage medium, you may simply be able to back up the entire contents of the Backups folder.

FileMaker Server now performs a live backup that requires significantly less time when the databases are unavailable. At the beginning of the backup process, FileMaker Server flushes the cache so that any data saved in memory is written to disk. Then it creates a dirty copy of the file. Users can still access and modify the original file while this copy is being made. After that's finished, the live database is paused and compared to the dirty copy; incremental changes are made to the copy so that it reflects the current state of the live file. The pause required for the incremental update is usually quite short and may not even be perceptible to users.

The Schedule assistant for scripts and email messages is quite similar. Note that you can run scripts from FileMaker or from your operating system, so the capabilities are quite large. In the case of FileMaker scripts, you select the database and provide the login information; in the case of system scripts, you choose the file.

*For an additional backup strategy, **see** "Using Progressive Backups," **p. 670**.*

Statistics

The Statistics window provides statistics, as shown in Figure 27.18.

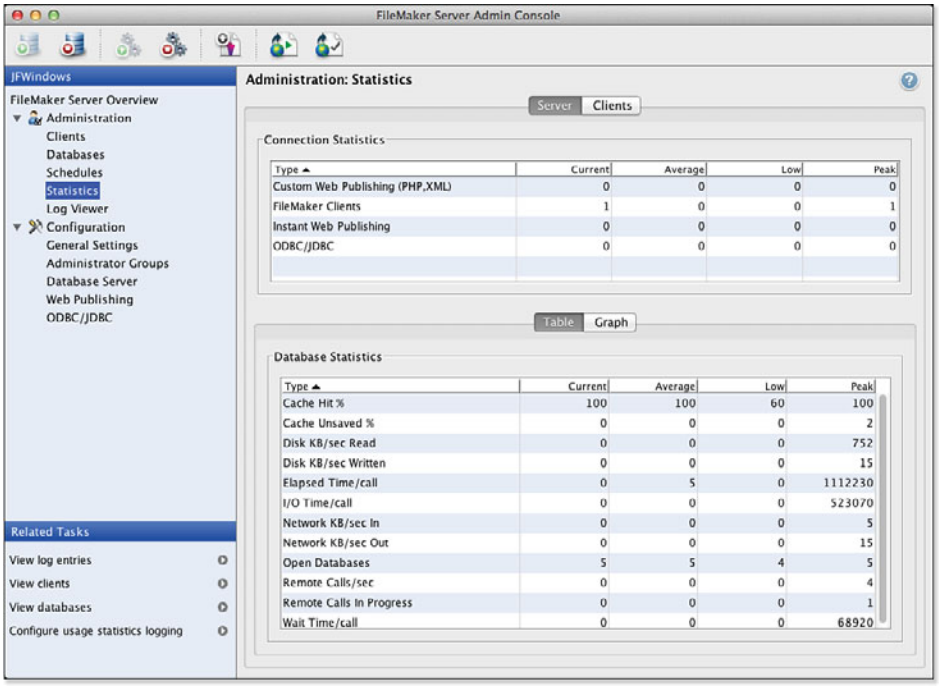


Figure 27.18 Review statistics periodically.

For each parameter, you can see the current, average, low, and peak values. Here's a list of some of what's monitored. Other items, such as times per call, are self-explanatory:

- **Clients**—This tells you the number of connected FileMaker Pro, Instant Web Publishing, Custom Web Publishing, and ODBC/JDBC clients.
- **Cache Hit %**—This number indicates how often FileMaker Server is finding the data it's looking for in the cache. Here, you want to see a number over 90%. Much less than that, and FileMaker is looking to the disk too often. In that case, it's a good idea to increase the size of the RAM cache on the Database tab of Database Server configuration. If the RAM cache is already as high as it can allowably go, you might want to consider adding more RAM to the machine, unless you've

already reached the limit of 800MB of cache memory, which will be reached at 4GB of system RAM.

- **Cache Unsaved %**—Like many database servers, FileMaker Server sets aside an area of RAM (of a size configured by the administrator) to use as a cache. When a user makes a request for data, FileMaker Server checks first to see whether the data is in the cache, and if so, it fetches it from the cache, more quickly than it could fetch it from disk. Over time, the contents of the cache are written out to disk. The period over which this occurs is governed by a setting on the Databases tab's Database Server configuration. The setting is Cache Flush Distribution Interval. For example, if that value were set to 1 minute, FileMaker would attempt to write the whole cache out to disk over the course of a minute. The Cache Unsaved % should ideally be around 25% or lower. If it's much above that, you might want to shorten the length of the cache flush period. Having too much unsaved data in the cache increases the odds of data corruption in the event of a crash.
- **Disk KB/Sec**—This gives you some idea of how much data is actually being written to disk over a given period. This is to some degree a measure of the extent to which the database files are being changed. If the files are being predominantly read from, the disk write activity should be low. If the files are constantly being written to, disk activity will be high. Keep an eye on this number if you expect that hard disk performance may be a bottleneck.
- **Network KB/Sec**—Average data transfer per second. This number tells you the extent to which the raw network bandwidth of the machine is being used up.

If you are having performance problems, this is the raw material that will help you to track down whether they are network, processor, memory, or other problems.

Log Viewer

The Log Viewer is shown in Figure 27.19. You can filter log entries by date and by the specific modules you want to view. Buttons are available to refresh the list as needed and to export data.

Log Viewer makes it easy to view the logs and to track down issues. If something unusual happens, look at the Log Viewer before anything else. The columns are sortable by clicking their titles. In addition, clicking a single log entry can provide more details at the bottom of the Log Viewer.

Configuration

The Configuration section of Admin Console lets you adjust settings as necessary. Most of the time, you will set and forget them: It is the Administrative settings that you use on a routine basis. Figure 27.20 shows the Configuration section. As you can see, there are four sections of the display, with links within each one. As you will see, those links take you directly to tabs.

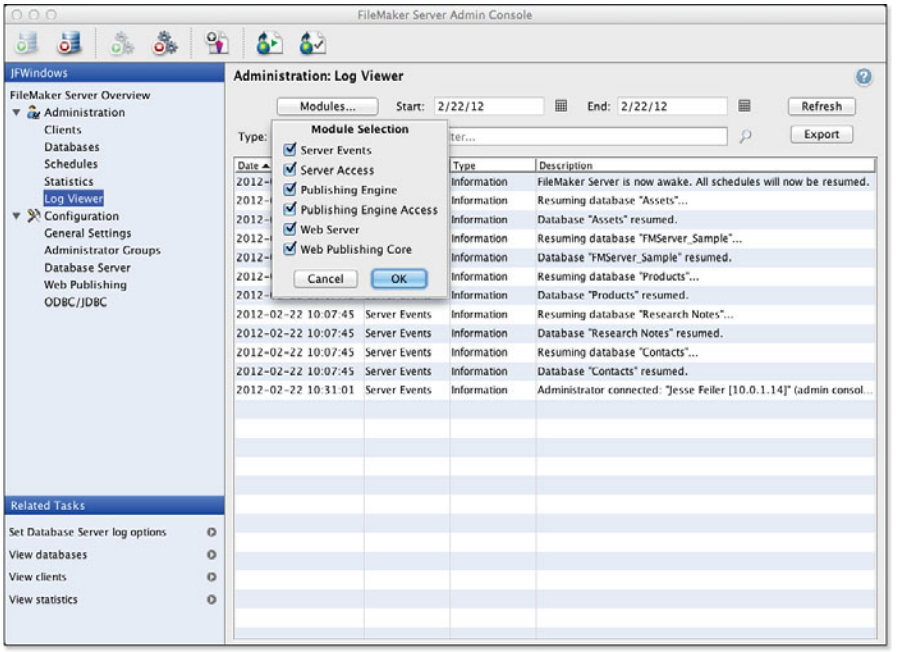


Figure 27.19
Use the Log Viewer to troubleshoot issues.

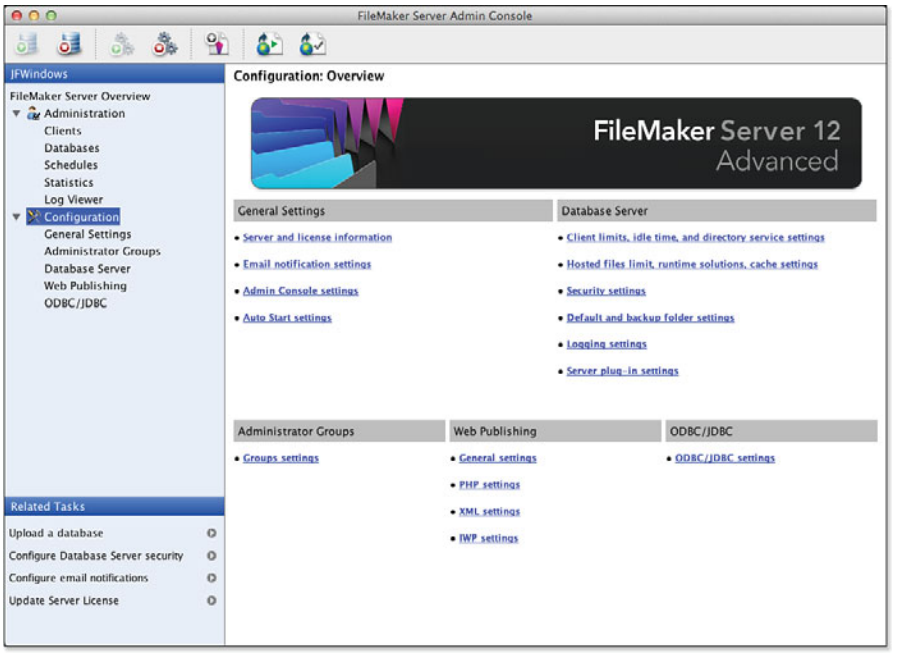


Figure 27.20
Configure the databases.

General Settings

The general settings are shown in Figure 27.21. You might want to compare the tabs with the General Settings links shown previously in Figure 27.20.

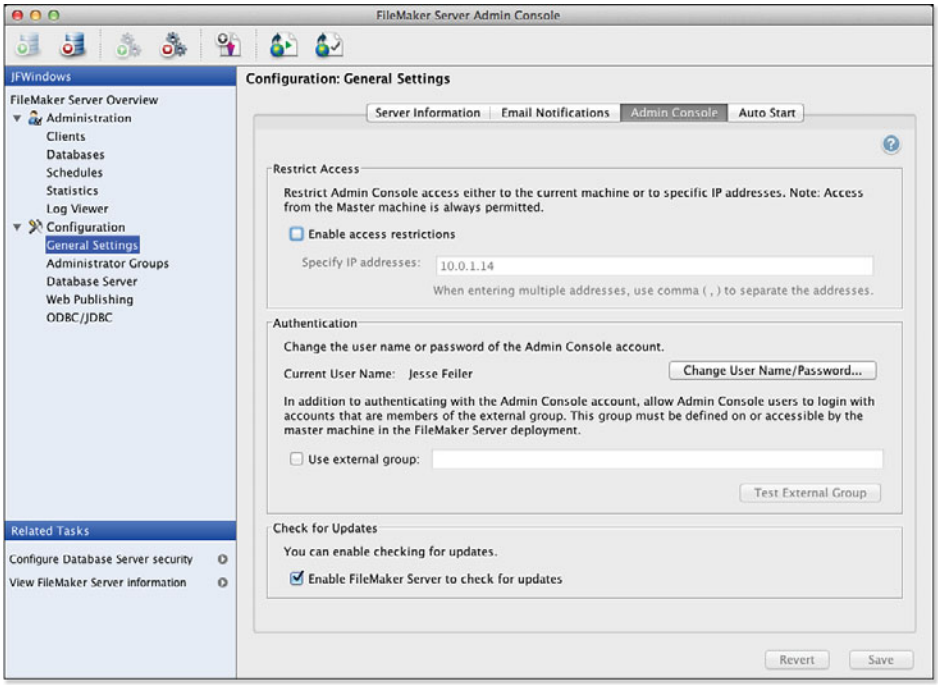


Figure 27.21 Control administrative settings.

Also in this area you can rename the FileMaker Server or change the administrator's name and address; you can configure Admin Console to be able to be run only from certain IP addresses, and you can set the Auto Start options so that the Database Server and Web Publishing Engine start up when the computer starts. You can also change the Admin Console password and account name.

The Email Notifications tab lets you set up email to be sent from FileMaker Server, as you see in Figure 27.22. Note that these messages are sent directly from FileMaker Server to an SMTP server—you do not need to install a separate mail application on your server computer.

Administrator Groups

You can set up administrator groups so that various people can control FileMaker Server. The password you create for the initial installation need not be shared by everyone who manages the server if you use administrator groups. As you see in Figure 27.23, you can add any number of groups. Each group can manage a folder in your databases directory. Also, as you can see in Figure 27.23, you can set up a start page for each group. Note that Administrator Groups are available only with FileMaker Server Advanced.

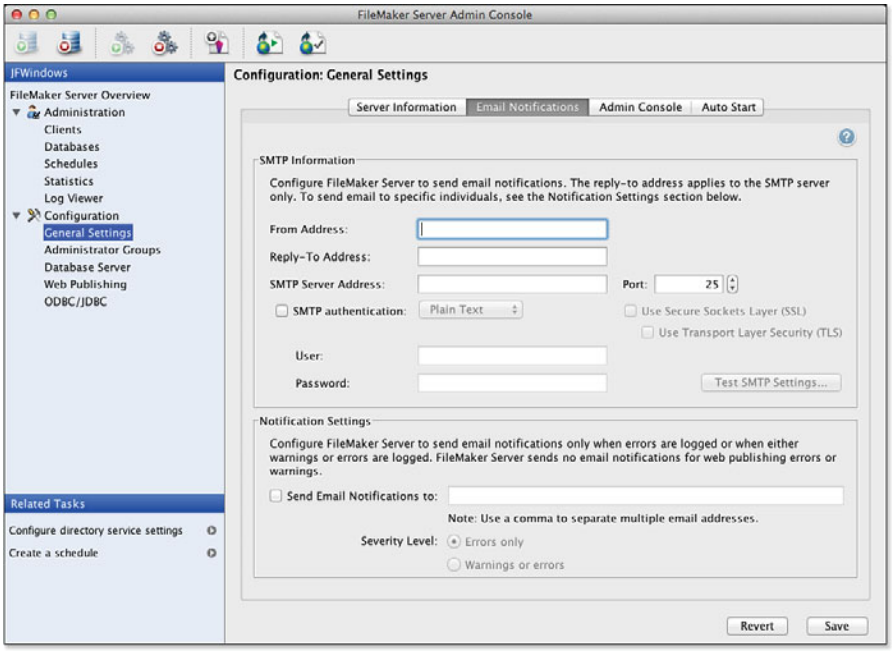


Figure 27.22
Send SMTP mail from FileMaker Server.

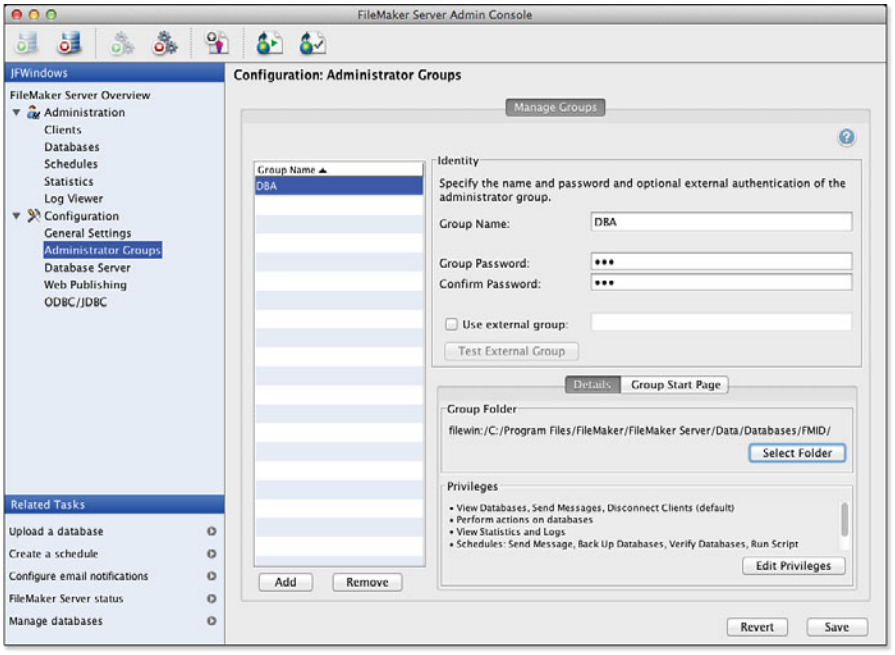
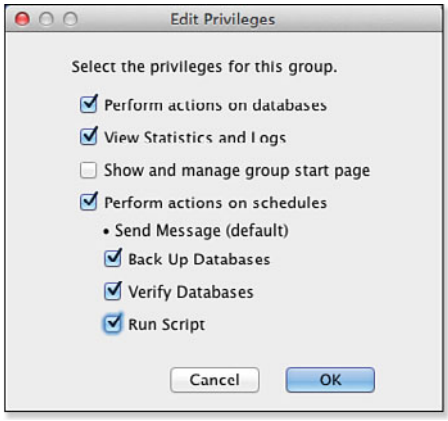


Figure 27.23
Set up administrator groups.

In addition, you can set specific privileges for a group, as shown in Figure 27.24. This enables you to show that page when a member of a group logs in. Particularly in the case of settings that are not often modified, a reminder of how to set them (not just how to use FileMaker Server but how to abide by the organization's standards and policies) is useful.

Figure 27.24
Set privileges for a group.



Database Server

The Database Server section provides the heart of your configuration settings. Here, you can set limits on the number of users, turn on secure communications, and set the default folders for backups and additional databases. Under FileMaker Pro Clients, you can use the Directory Assistant to select an external LDAP directory to be used.

You can specify the maximum number of FileMaker Pro clients that can connect at one time on the FileMaker Pro Clients tab, and on the Databases tab you can control the maximum number of files that FileMaker Server will try to open. (You control the maximum number of web clients in the Web Publishing section.)

You can also specify the amount of RAM to set aside for a database cache on the Databases tab. Admin Console lets you know what it thinks the maximum allowable cache size is, based on total available RAM. You can improve performance on 64-bit systems by setting the cache to the maximum of (MB of Physical RAM − 128MB) / 4.

Using Progressive Backups

In addition to scheduled backups, you can take advantage of progressive backups which were implemented first in FileMaker Server 12. You turn them on from the Folders tab of Database Server as shown at the bottom of Figure 27.25.

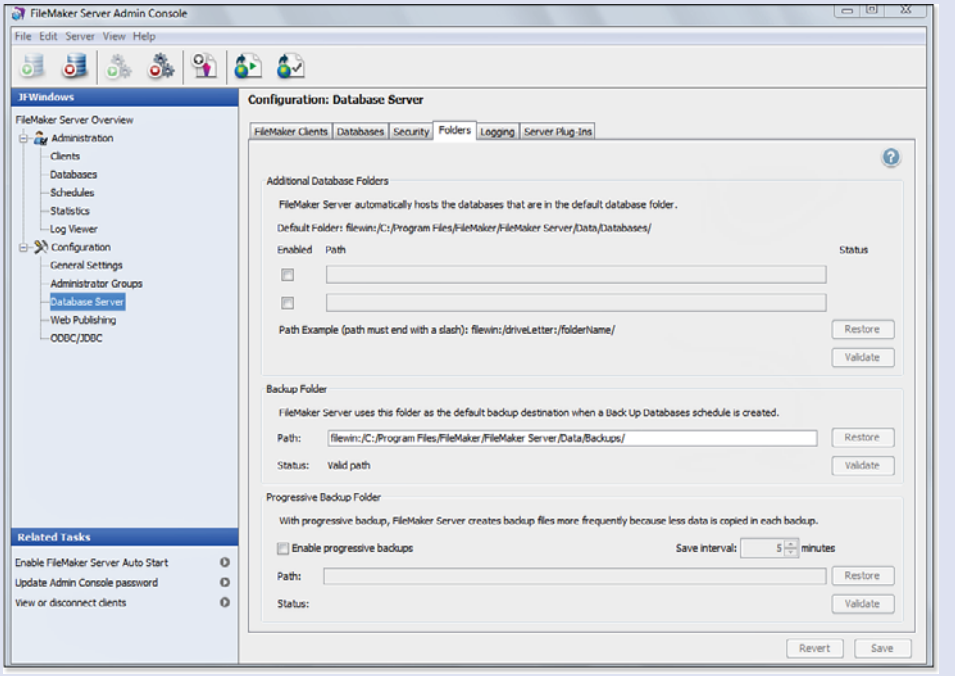


Figure 27.25
Use progressive backups.

When you turn on progressive backups, FileMaker Server makes a complete backup of all its files. This can take time: In many cases, instead of thinking about doing this overnight, think about doing it over a long weekend. If that isn't enough time, it can still continue to run while users are accessing the databases, so if you have a period of time in which the system is lightly used, that might be the time to do it.

You assign a path for the progressive backup as shown at the bottom of Figure 27.25. As changes are made, they are stored in a log. Periodically, at periods from 1 to 60 minutes (you set this at the lower right), the changes are merged into a new backup. This mechanism is much faster than periodic large-scale backups.

Despite this, many people suggest that you use both strategies instead of choosing one of them. After the initial setup is performed, progressive backups impose a very slight extra load on the system in most cases.

caution

Progressive backups rely on being able to manage the files in the progressive backup directory that you have provided. If you need to restore from a progressive backup, copy that file. Do not move files in the progressive backups directory, and do not open them directly with FileMaker Pro. Always copy them to another directory and then proceed.

Web Publishing

Web publishing sections are discussed in the relevant chapters.

*For more information on Instant Web Publishing, **see** Chapter 24, "Instant Web Publishing."*

*For more information on PHP and XML, **see** Chapter 25, "Custom Web Publishing with PHP and XML."*

ODBC/JDBC

The last item in the Configuration section lets you turn on ODBC/JDBC sharing so that other xDBC clients can connect to FileMaker databases.

FileMaker Extra: Best Practices Checklist

Much of the work of server maintenance and administration consists of diligently following a routine. For each server or service you maintain, there should be a checklist of necessary tasks. Some of these you have to do only once, when you set things up. Others are recurring tasks that you should attend to carefully. In this section, we present a series of considerations for setting up and maintaining a FileMaker Server installation.

If you're working with network staff or administrators who don't have previous experience with FileMaker, offer them this list as a handy overview of the essentials of maintaining a FileMaker Server.

Determine Network Infrastructure

You'll want to run FileMaker traffic over the fastest network possible. Before doing anything about a server machine proper, make sure that you have a handle on prevailing networking conditions. What's the topology of the network over which FileMaker will run? Is it fully switched or are hubs involved? What's the minimum speed of links within the network? With what other services will FileMaker traffic be competing? Knowing the answers to all these questions can help you make the right hardware choices and will give you a leg up on diagnosing any later problems that appear to be network related.

Purchase Hardware

We discussed ideal hardware characteristics earlier in the chapter. Simply put, buy the best machine you can afford. Get a machine with one or more fast processors, a healthy dose of RAM, fast disk storage, and consider a hardware RAID configuration), and a networking capability that matches the prevailing speed of your network. Expandability is also a good idea: Additional drive bays, external hard drive connectivity, and multiple slots (for additional or upgraded networking capability, for example) are all desirable.

Install Software

Use the latest version of an approved operating system, with all relevant patches and updates. Avoid enabling any other services on the machine except for those strictly necessary for system administration. In particular, avoid file sharing as much as possible. If it can't be avoided, make *sure* you do not enable file sharing for those areas that contain the hosted database files; otherwise, you run the risk of file corruption.

Install FileMaker Server and make sure that all appropriate updates are applied. Make sure that your version of FileMaker Server is compatible with both the operating system and, if applicable, the service pack level of the operating system. Make sure that all drivers are up to date, especially drivers for critical features such as disks. Make sure that the BIOS and firmware for the machine are up to date as well.

It's a good idea, if possible, to put the FileMaker Server data on its own volume, separate from the volume containing the applications and operating system. Here are a few more useful tips for operating system configuration:

- Disable any disk-indexing software.
- Configure any virus-scanning software so that it does not scan the FileMaker data files for viruses.
- On Windows, turn off Volume Shadow Copy.
- On Windows, set the network throughput in File and Print Sharing to Maximize Data Throughput for Network Applications.
- On Windows, set your virtual memory paging files to a specific size instead of allowing them to grow as needed.

Configure FileMaker Server

Configure FileMaker Server to a level appropriate for your expected usage (see the detailed notes earlier in the chapter). Bear in mind that it's worthwhile to try to use only those resource levels (for

example, maximum numbers of connected clients and hosted files) that you think you'll need. Here are some other quick rules of thumb:

- Set the cache to half the allowable maximum and increase it if the cache hit rate dips much below 90%.
- Set the cache flush interval to 1 minute. Fast, modern hard drives can flush most or all of even an 800MB cache in that period.

Deploy Databases and Schedule Backups

Decide on your database directory structure—that is, how you'll group databases into directories on the server. Decide whether to use an alternative database directory (but make sure that it's on a local hard drive, not on a networked volume!).

Regardless of your choice, establish backup schedules that provide you and your organization with an appropriate level of security. How much data can you afford to lose? Decide on the answer and back up accordingly. Remember that local backups by themselves are not sufficient security: You should make provisions to transfer this data to offline storage such as a tape backup. Finally, consider how many backups you want to keep in which place: how many on disk, and how many on offline storage such as your tape backup.

Monitor Usage Statistics

Keep a careful eye on usage statistics, especially early on when usage patterns are being established. Be alert for signs of inappropriate configuration, such as a low cache hit percentage or a high amount of unsaved data in the cache. Make sure that your network bandwidth continues to be adequate.

Recheck Performance

In addition to checking statistics and the logs, continually check the perceived performance of your databases. Users complaining that "FileMaker is slow" do not distinguish between unindexed searches and server configurations. Especially if you are using FileMaker Server Advanced, monitor performance and reconsider the deployment across multiple computers from time to time.

Stay on Top of Java

Because Admin Console uses Java, which is a shared resource, keep an eye out for Java errors when you launch it. If you can connect to Admin Console from one computer but not another, Java may be the issue. Clear Java cache files and revert as much as possible to a clean Java installation.

Monitor Log Viewer

Check the Log Viewer periodically to make sure that things are operating smoothly.

Keep Current with Software Updates

It should go without saying, but you'll want to keep current with all updates and patches to your operating system and to all software packages installed on the server, including, of course, FileMaker Server itself. You should monitor the updates that are available and probably not auto-install updates that might cause problems with FileMaker. In general, the FileMaker website (either TechTalk or the KnowledgeBase) will have information about major environmental updates soon after they are released (particularly if there is a compatibility problem).

INDEX

SYMBOLS

A

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z