CHAPTER 3

Defining and Working with Fields

In this chapter

Working Under the Hood

Fields are the heart of any database. By storing information in properly categorized fields, you impart both function and meaning to what would otherwise be an incomprehensible pile of raw data.

We'll spend much of this chapter describing what kinds of fields exist in FileMaker Pro, how they store information, and how to ensure proper data integrity in your database solutions.

If you're new to development in FileMaker Pro, this chapter is a good place to start. No doubt some of the topics we cover will lack a certain context, but establishing a solid foundation in field definition is a vital part of becoming a practiced developer.

If you have built a few FileMaker Pro databases, you may need only to skim this chapter. Of the topics we cover here, indexing is likely the most advanced; our discussion explores some subtle differences from prior versions of FileMaker Pro.

New Databases Begin with Field Definitions

To create a new database, simply launch FileMaker Pro and then choose File, New Database. You'll be presented with the option to start with a template or to create a new, empty file. To create a file of your own, select the Create a New Empty File option and click OK.

After you've stepped through these first tasks, you'll be taken to the Define Fields dialog.

Working with Templates
We recommend that you go back at some point and work with the templates that ship with FileMaker. They're a good learning tool, and you will be able to see how fields are defined in these finished solutions. There are dozens of templates, they're not all that complicated, and they'll give you some good ideas for designing your own solutions. From them you can learn about simple user interfaces and calculation functions and can see some basic scripts in action as well.

Using the Define Database Dialog

When you choose to start on a new, empty database, FileMaker Pro creates a file for you and automatically opens the Define Database dialog (shown in Figure 3.1). As a developer, you'll spend a good bit of time in the three tabs in this dialog. FileMaker Pro's Define Database dialog allows you to create the fields, tables, and relationships you need in order 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. It is these elements that compose a database's structure or *schema*. It is here that you form your database behind the scenes.

NOTE

Notice the active table in Figure 3.1. The fields you define are associated with this selected table.

Figure 3.1
The three tabs allow you to switch among defining tables, fields, and relationships.

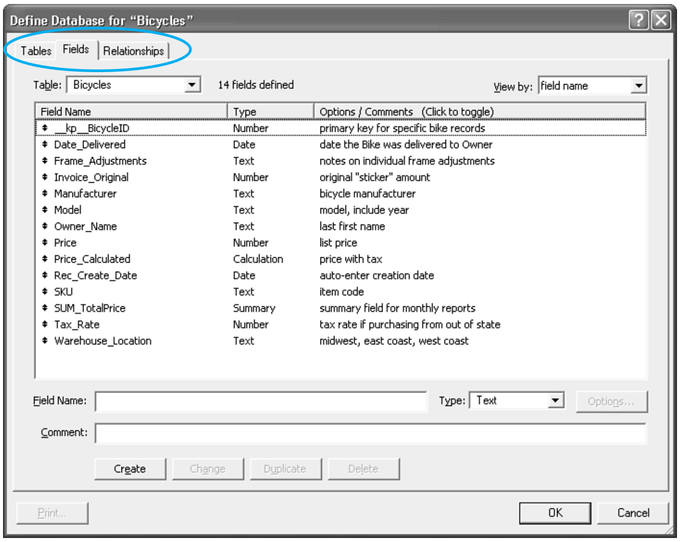


Table: Bicycles   14 fields defined   View by: field name

Field Name	Type	Options / Comments (Click to toggle)
__kp__BicycleID	Number	primary key for specific bike records
Date_Delivered	Date	date the Bike was delivered to Owner
Frame_Adjustments	Text	notes on individual frame adjustments
Invoice_Original	Number	original "sticker" amount
Manufacturer	Text	bicycle manufacturer
Model	Text	model, include year
Owner_Name	Text	last first name
Price	Number	list price
Price_Calculated	Calculation	price with tax
Rec_Create_Date	Date	auto-enter creation date
SKU	Text	item code
SUM_TotalPrice	Summary	summary field for monthly reports
Tax_Rate	Number	tax rate if purchasing from out of state
Warehouse_Location	Text	midwest, east coast, west coast

Field Name: Type: Text

Comment:

FileMaker Pro will have already created a default table for you, named the same as the file itself. Notice the Table menu selection on the Fields tab of the dialog in Figure 3.1. Any fields you create will be created in that table.

→ For some basic information on tables, **see** "Understanding Tables," **p. 31**.

→ For a detailed discussion of multiple-table solutions, **see** Chapter 6, "Working with Multiple Tables," **p. 157**.

Notice the third tab in the Define Database dialog: Relationships. We won't be covering multitable relational databases in this chapter, but it is on that tab that you'd create the relational associations among tables in your solution.

→ For information on relational data modeling, **see** Chapter 5, "Relational Database Design," **p. 129**.

TIP

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 something will save time later in trying to figure out what you were thinking at the time.

To view comments, toggle between options and comments at the top of your field list.

WORKING WITH FIELDS

Every table in any database—FileMaker Pro or otherwise—is a collection of information stored in fields (or columns, if you're familiar with that terminology). It is by storing information in appropriate fields that a database is given meaning.

For example, by entering "124 Main Street" in a field called Address, we've identified what "124 Main Street" is. In the case of a street address, it's fairly easy to identify without a field definition, but what about "912.5" all on its own? That could be a price, a number of units, a chapter heading, or a thousand other things. When you place that number into a named column or field, your data becomes meaningful. If "912.5" sat in a field named Temperature, you'd likely conclude that it is pretty darn hot. Conversely, if it sat in a Kilobytes Available field, you'd look to be buying a new hard drive. Keep this ultimate goal of imparting meaning in mind as you create and name fields, and assign appropriate data types to them.

Field Naming Conventions

One of the nice things about FileMaker Pro development is the freedom developers have in naming fields. (FileMaker Pro is not unique in allowing developers field naming freedom, by the way.) That freedom also, unfortunately, gives rise to confusion and arbitrary naming conventions. Name_xTJm2 may mean something to someone, or Name might also, but both examples—the overly specific and the overly general—require a strong familiarity with a given system. If you ever return to a database months after building it, odds are you will have forgotten your clever abbreviations.

➔ For a complete presentation of Soliant Consulting's coding standards, **see** Chapter 27, "Documenting Your FileMaker Solutions," **p. 841**.

We encourage you to take advantage of FileMaker's allowance for long field names of up to 100 characters. Use full text names (like Street Address line1), avoid abbreviations (or if you use them, be sure to provide an obvious key!), and try to group things logically. Note that although FileMaker allows for spaces and special characters, we recommend using underscores, letters, and numerals only.

Here's an example of what we'd consider a fairly reasonable approach to naming fields:

- Address_City
- Address_Postal_Code
- Address_State
- Address_Street_line1
- Address_Street_line2
- Person_Name_First
- Person_Name_Last
- Phone_Home
- Phone_Work

These fields are quite simple to identify and are neatly grouped together when sorted alphabetically. This isn't such a big deal for small databases, but if you ever work on a large database, with multiple developers, a well-established naming convention is vital. We encourage you to adopt good programming habits right from day one.

Another common approach many developers use includes abbreviations for data types. Often it's handy to know the data type of a given field when working with it without having to refer to the Define Database dialog. Here we've used "t" for text, "n" for number, and "c" for calculation:

- ProductName_t
- Price_n
- TaxRate_n
- Tax_c

We'll cover indexing later in the chapter, but 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, these fields would probably be left behind. Here we have two abbreviations: "k" for key (or match field), and "z" (so that it sorts to the bottom of the list) 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," **p. 129**.

- __kp_primary_AlbumID
- _kf_foreign_ArtistID
- AlbumName
- Date
- z_SelectedPortalRow
- z_UserColor_Preference
- z_UserGenre_Preference

Finally, here's a real example from a database we recently were hired to modify (used with permission and good humor!):

- Bike Type
- Wheel Dm
- Bike Name
- Model
- Type
- Temp
- Date
- Bike
- Bike2
- Sp.99 Meas
- Tire Dm
- Bikeid
- Sku
- 2002 Tire Diam
- 1999 Tire Diam
- BikeMODEL
- SUMMARY
- zTempzzz
- Phils field (no lie!)

We're sure that we've belabored the point, but this database was difficult to modify not because it was complex, but because it was hard to interpret. As in all things, a little planning goes a long way.

If you're planning on using FileMaker Pro as a web back end, refer to "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. 648**, as well as Chapter 23, "Custom Web Publishing," **p. 699**.

→ For more thoughts on documenting and commenting in your database, **see** Chapter 27, "Documenting Your FileMaker Solutions," **p. 841**.

There are some restrictions on field naming in FileMaker Pro: A field name must be unique within its table, and must be less than 100 characters in length.

You can opt to use special characters, numbers, spaces, even the names of functions, but we recommend against using them. If you use , (comma), +, -, *, /, ^, &, =, ≠, >, <, (,), ", ; (semicolon), : (colon), or :: (double colon relationship indicator), you need to enclose such special characters within a `$()` in calculation formulas to have them interpreted as field names. For example, the calculation `$(Tax,special)` returns the value of a field named `Tax,special`.

We recommend strongly that you name fields without using special characters, names of functions, or operators (`AND`, `OR`, `NOT`, `XOR`, `TRUE`, `FALSE`).

The same is true for fields that begin with a space, a period, or a number: You'll have to contort your calculations to deal with them. Don't use them. Begin each field with a standard alphabetical letter or an underscore.

Adding Field Comments

Notice also that you can add comments to your field definitions. We don't mean to be pedantic, but we want to drive home that establishing good programming habits will serve you well for the rest of your life as a developer. Use the field comments feature. Explain to yourself a year from now why a field exists, any dependencies or assumptions you made, and possibly how you intend to use it.

Creating New Fields

To create fields in FileMaker Pro, you need to enter some text in the Field Name area of the Define 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 it ever comes time 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 from 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. 136**.

Working with Field Types

One of the most important aspects of understanding FileMaker Pro is understanding field types, how they're different from one another, and how to use them effectively.

Simply stated, field types identify what kind of information each field of your database is expected to hold. A person's name is text, the dollar 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.

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. It's the combination of a proper identifying field name and a data type definition that gives a database its context and meaning.

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'll 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: A number field can be expressed as a Boolean. A Boolean value is either true or false, and is often used to test the condition of something. A zero or null value in a number field is treated as false in the Boolean sense; any other data is treated as true. You will often run across number fields being used to store Boolean values.

The primary distinction between a number field and a text field lies in how they're sorted: 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 Gregorian calendar dates only. FileMaker Pro honors whatever date formatting your country follows by taking the standard your operating system uses at the time a new file is created. Date formats—the order of year, month, and day—are common for a given file. Although it's possible to change the way dates are displayed, it is this basic ordering that is fixed at the time of file creation.

Dates in FileMaker Pro are internally stored as the number of days since 01/01/0001. January 1, 2004, for instance, is 731581. 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 seven.

Date fields can store values from January 1, 0001, to December 31, 4000.

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

Time

Time fields hold HH:MM:SS.ddd information. Notice that a decimal may be added to the end. Also useful: If a user enters `25:00`, FileMaker Pro rightly interprets this as 1:00 a.m.

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.

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 with date formats, your time format is established during the creation of the file, based on system operating system settings.

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

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: 1/1/2004 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.

TIP

To extract just the date from timestamp data, simply use the `GetAsDate()` function. Likewise, use `GetAsTime()` to extract just the time.

Container

Container fields are different from the five 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.

Container fields also support displaying/playing three native types of media: pictures, QuickTime movies, and sounds. Refer to the FileMaker help system for supported formats, but most common image formats are included…as well as some you won't expect. For example, by using QuickTime, it's possible to display and play a Macromedia Flash 5 `.swf` file.

Last, on Windows, a wide range of OLE objects are supported, including Microsoft Excel documents, PDF, and more.

There's one important thing to remember about using container fields: Either you can store the file or media in FileMaker itself—requiring disk space—or you can simply store a path reference to the file instead. If you choose to store just a reference to the file, FileMaker

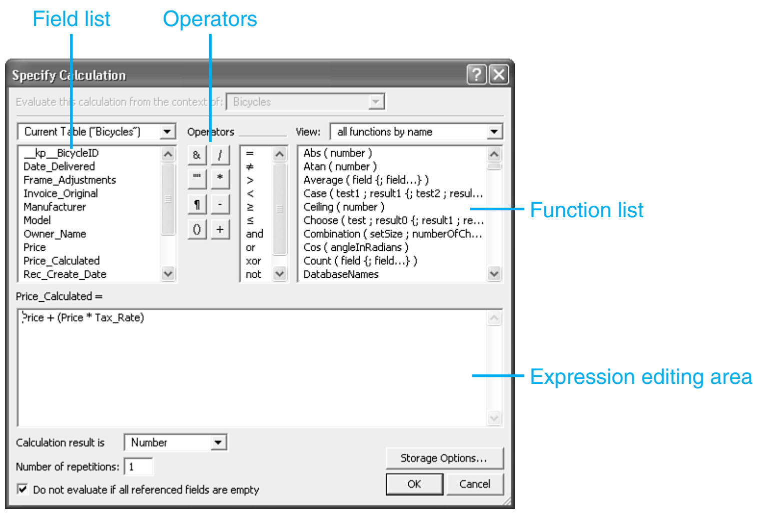
Pro, somewhat like a web browser, displays the image or file icon as necessary, but does not hold the actual document itself. A nice feature of storing references is that you can then double-click documents in your container fields to launch them in your operating system.

CAUTION

Keep in mind that if you move the source document, the FileMaker Pro reference remains but is no longer valid.

Calculation

Calculation fields evaluate formulas and display the requisite results. When you create a calculation field, the Specify Calculation dialog, shown in Figure 3.2, opens.



Evaluate this calculation from the context of: Bicycles

Current Table ("Bicycles")

__kp__BicycleID
Date_Delivered
Frame_Adjustments
Invoice_Original
Manufacturer
Model
Owner_Name
Price
Price_Calculated
Rec_Create_Date

Operators

& / "" * ¶ - () +

= ≠ > < ≥ ≤ and or xor not

View: all functions by name

Abs (number)
Atan (number)
Average (field {; field...})
Case (test1 ; result1 {; test2 ; resul...
Ceiling (number)
Choose (test ; result0 {; result1 ; re...
Combination (setSize ; numberOfCh...
Cos (angleInRadians)
Count (field {; field...})
DatabaseNames

Price_Calculated =

```
Price + (Price * Tax_Rate)
```

Calculation result is Number

Number of repetitions: 1

☑ Do not evaluate if all referenced fields are empty

Figure 3.2 Calculations form an essential pillar of FileMaker Pro development.

→ Be sure to refer to FileMaker Pro's online help or our companion book, *FileMaker 8 Scripts and Functions Desk Reference*, for a complete list of functions. The book presents every function in FileMaker along with discussion and examples.

The features of the Specify Calculation dialog box 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.
- **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 where you assemble your actual formula or expression. This is a simple text entry area: If you want, work in a text editor and paste 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.

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.
- `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 zero or one (Boolean) depending on whether MyField has a value in it, including zero. If a zero 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've just tested for the number of days past 1/1/0001.)

Calculations are fundamental to FileMaker programming, and it's worth your while to master them fully.

→ For more detail on calculations, **see** Chapter 8, "Getting Started with Calculations," **p. 217**, and Chapter 14, "Advanced Calculation Techniques," **p. 391**.

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

Summary

Summary fields allow 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.

For example, say you have a table called Transaction, which contains Transaction_Date and Transaction_Amount fields. You can then define and place on a layout a summary field to total the Transaction_Amount field. The summary field adds the values of the Transaction_Amount field for whatever set of records is currently active. If you perform a find, by date, on 10/1/2006–10/31/2006, 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.

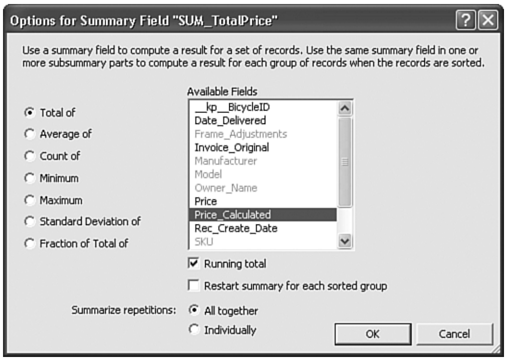
Table 3.1 contains a list of summary field functions.

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 may also enable the option to display a running total for your recordset. This then 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 valid data in the specified field. For example, if 18 of the 20 current found records have data, your summary field will display `18`. A running count functions similarly to a running total: It displays the incremented count of each record in your found 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. 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 are 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 Field dialog opens, prompting you to choose the function you want to use and the field for which you want a summary (see Figure 3.3).

It's generally a good idea to place summary fields on their own layouts so that a user deliberately chooses to have them evaluate a found set.



Options for Summary Field "SUM_TotalPrice"

Use a summary field to compute a result for a set of records. Use the same summary field in one or more subsummary parts to compute a result for each group of records when the records are sorted.

[✓ Total of] Average of Count of Minimum Maximum Standard Deviation of Fraction of Total of

Available Fields:
- __kp__BicycleID
- Date_Delivered
- Frame_Adjustments
- Invoice_Original
- Manufacturer
- Model
- Owner_Name
- Price
- [✓ Price_Calculated]
- Rec_Create_Date
- SKU

[✓ Running total]

Restart summary for each sorted group

Summarize repetitions: [✓ All together] Individually

Figure 3.3
Summary fields are useful for performing functions across sets of records, but use them with care. They can increase the time it takes to load any given layout.

In Browse mode a summary field will evaluate your found set and display 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 will display information only when the user scrolls to that portion of the window.

Summary fields will evaluate a found set for a given layout whenever you enter Preview mode.

Assigning Field Options

In addition to establishing fields and assigning data types, you may assign various options to your fields as well. These 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 have named a field and chosen its type on the Fields tab of the Define Database dialog box, click Create to save it to your database. You may 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

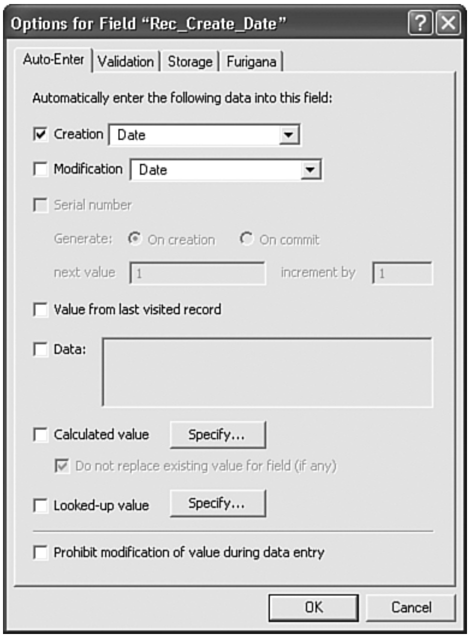
When defining noncalculation fields in FileMaker Pro, you can choose to have data automatically entered into a field as records are created and/or modified. The applications for this can range from assigning default values to fields, to automatically reformatting data, or inserting values from other 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 3.4).

Auto-entry data is inserted into a field based on some trigger event. The most common event is record creation: When a user clicks New Record, data can be prepopulated into the record and then be accessible for changes to be made. 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 will cover both cases in the sections that follow.

Figure 3.4
FileMaker's auto-entry options allow you to define rules for automatically populating data into fields in your database.



Automatically enter the following data into this field:

- [✓] Creation: Date
- [ ] Modification: Date
- [ ] Serial number
  - Generate: [✓ On creation] On commit
  - next value: 1 increment by: 1
- [ ] Value from last visited record
- [ ] Data:
- [ ] Calculated value
  - [✓] Do not replace existing value for field (if any)
- [ ] Looked-up value
- [ ] Prohibit modification of value during data entry

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), or current account name (the one entered by the user when logging in to the database).

CAUTION

The name is something users can modify as they want via FileMaker's Preferences dialog, so you generally shouldn't depend on it for anything vital. If you want to depend on knowing who has created or modified a given record using auto-entry functions, always use the Account Name option.

NOTE

If you do not change any of the account settings of a new file, FileMaker will have established two default accounts for you: Guest and Admin. Both begin with full access to the database.

Serial Number

Using this option allows you to auto-enter a number that increments every time a new record is added to the table. Often this is used to uniquely identify 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 then 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 regardless of how complex or simple your plans are, we encourage you to adopt some best practices.

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

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 will guard against confusion or even possible loss of data integrity.

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 thing is to make certain your keys are unique and unmodifiable.)
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 then override, you're risking the chance that they'll introduce duplicate IDs.

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. 162**.

Value from Last Visited Record

Used most often as a way to speed data entry when information is often repeated for groups of records, this 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 a field and activating it, it is the data from the first, edited record from which a new record obtains its value.

Data

Here you may 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. Being 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 its value will always be determined by its defined formula, it is possible to insert the results 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 results of the calculation formula will be entered into the field, overriding any existing value, anytime a field referenced by the calculation changes.

Put differently, any field referenced in your calculation statement acts as a trigger—anytime that referenced field is updated, the calculation will be retriggered, and its result put back into the auto-entry field. In fact, the auto-entry field itself can act as such a trigger, if it's referenced in the auto-entry calculation. This enables you to dynamically reformat data as it is entered. One great example of this is a phone number field. You may always want phone numbers formatted as "(123) 456-7890" regardless of how a user entered the data. By using a calculated auto-entry function, you can reformat the phone number anytime it's modified.

For an example of this technique, refer to Figure 3.5.

The actual calculation for this auto-entry option looks like this (returned as text):

```
Let ( [
    //define variables:

rawNumber = Filter (Phone_Number; "0123456789") ;
length = Length (rawNumber);
red = RGB (160;0;0);

    //set error flag for a phone number that's too short
error = If ( length < 10 ; TextColor ("error: " & Phone_Number; red); "")

];
    // now apply the phone formatting and return results
```

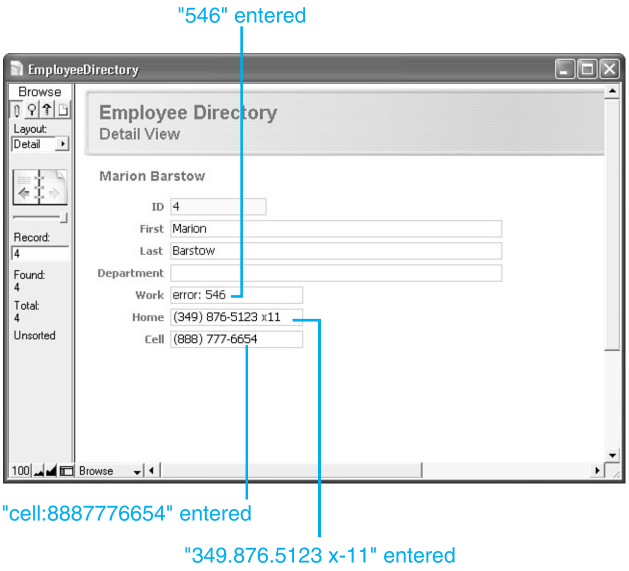
```
        If ( error ≠ ""; error;

            "(" & Left (rawNumber; 3) & ") " & Middle (rawNumber; 4; 3) &
            "-" & Middle (rawNumber; 7; 4) &

            // this condition tests for extra digits
            that we'll treat as an extension
            If ( length > 10; " x" & Middle (rawNumber; 11; length - 10); "")

        )
    )
```

Figure 3.5
By using a self-referencing calculation, FileMaker Pro is able to replace and correct data as it is entered by the user.



Employee Directory
Detail View

Marion Barstow

ID	4
First	Marion
Last	Barstow
Department	
Work	error: 546
Home	(349) 876-5123 x11
Cell	(888) 777-6654

Here the calculation is written as though it were being applied to a field called Phone_Number. To get the results shown in Figure 3.5, you would need to either redefine the calculation for each of the three fields shown, or (better) define a single custom function that contains the bulk of the logic in the preceding listing, and call that function from the auto-entry calculation for each field you want to filter in this way.

→ To learn more about advanced calculation functions, including custom functions, **see** Chapter 14, “Advanced Calculation Techniques,” **p. 391**.

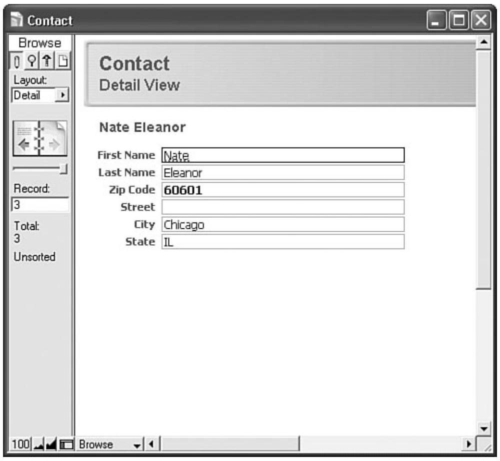
→ To see a version of this function that works more flexibly and for international formats, see our companion book, *FileMaker 8 Functions and Scripts Desk Reference*.

Looked-Up Value

This auto-entry option copies a value from a record in a related table into a field in the current table. Anytime 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 ZIP code into a given record, it’s possible you could have another table and then auto-populate your city and state fields with the appropriate information.

When a user enters a ZIP code in the record in Figure 3.6, the City and State fields below are triggered to pull values from the ZipCodes table. An important fact to keep in mind is that FileMaker has *copied* 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.



Contact
Detail View

Nate Eleanor

First Name: Nate
Last Name: Eleanor
Zip Code: 60601
Street:
City: Chicago
State: IL

Figure 3.6
Lookup functions work somewhat like relational data, but instead of displaying values from a related record, their information is copied and stored when a trigger event occurs.

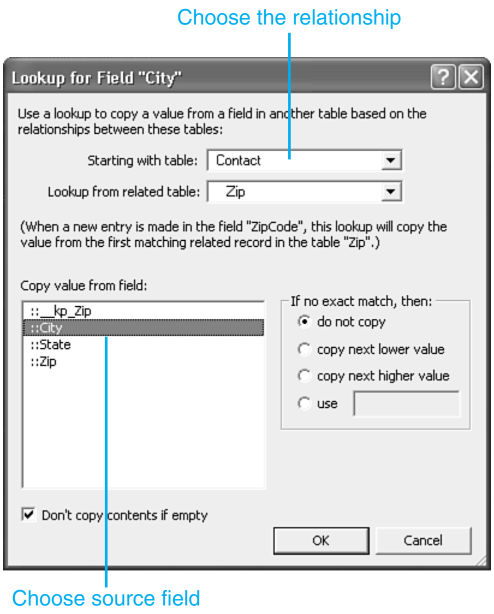
Take special note that lookup auto-entry functions work just as all auto-entry functions do: They copy or insert information into a field. You are not displaying related information, nor are you controlling content by calculation. Thus, lookup values are not live links to related data. If you were to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

This is an important distinction to understand, 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 tracked the prices of products, you'd want to store the price of each Order line item or product within the order itself. That way if your prices ever change, your historical orders will preserve their original prices.

To see how to create a lookup field, refer to Figure 3.7.

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

When you're performing a lookup, it is possible to work with near matches, in addition to exact matches. In the case of the ZIP codes example, obviously you'd 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.



Lookup for Field "City"

Use a lookup to copy a value from a field in another table based on the relationships between these tables:

Starting with table: Contact

Lookup from related table: Zip

(When a new entry is made in the field "ZipCode", this lookup will copy the value from the first matching related record in the table "Zip".)

Copy value from field:

- ::__kp_Zip
- ::City
- ::State
- ::Zip

If no exact match, then:

- [✓ do not copy]
- copy next lower value
- copy next higher value
- use

[✓] Don't copy contents if empty

OK Cancel

Figure 3.7
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.

How you set up your trigger values is important here. It's easy to compare numbers and come up with the next closest value. If your trigger 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," **p. 157**.

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 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.

TIP

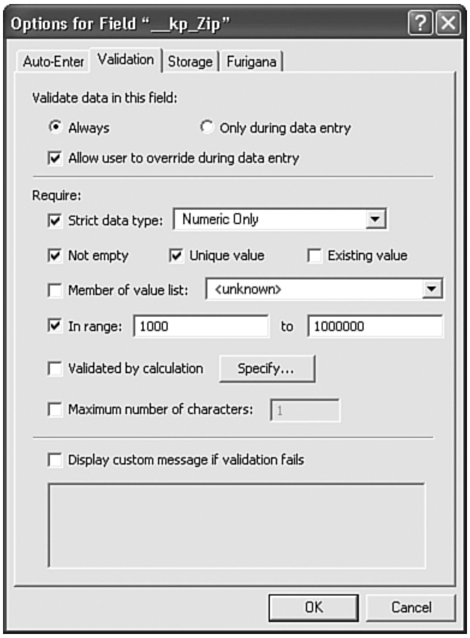
Using FileMaker 8's capability to import tables allows you to create a boilerplate new table complete with a primary key serial ID, four housekeeping fields, and whatever other standard fields you want defined. Then whenever you need to add a table to your database, import from the boilerplate rather than having to re-create these standard fields.

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, none is perfect. We all make mistakes.

As data is entered into FileMaker Pro, you may opt to apply one or more validation checks to test that certain conditions are met before allowing users to commit the record to your system. This 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 3.8.



Options for Field "__kp_Zip"

Validate data in this field:

- [✓] Always
- [ ] Only during data entry
- [✓] Allow user to override during data entry

Require:

- [✓] Strict data type: Numeric Only
- [✓] Not empty
- [✓] Unique value
- [ ] Existing value
- [ ] Member of value list: <unknown>
- [✓] In range: 1000 to 1000000
- [ ] Validated by calculation
- [ ] Maximum number of characters: 1
- [ ] Display custom message if validation fails

Figure 3.8
These are common validation settings for a numeric key value meant to always remain unique and unmodified by users. Even if your users can't access a given field on any layout, it's still a good policy to validate.

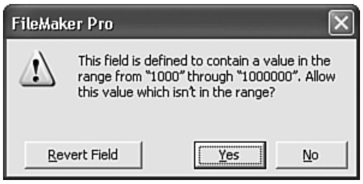
This example demonstrates a common approach to ensuring that your primary keys are properly maintained. This may be overkill if you've enabled the Prohibit Modification of Value During Data Entry option on the Auto-Enter tab, but on the chance that a developer turns that option off for some reason, or 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 19, "Importing Data into FileMaker Pro," **p. 567**.

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, the user is not interrupted or prompted for action.

Figure 3.9 shows an example of what your users might see when validation fails.



FileMaker Pro

This field is defined to contain a value in the range from "1000" through "1000000". Allow this value which isn't in the range?

Figure 3.9
The Yes 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 enter data manually into the field being validated; some validation will 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()`.

Validation isn't triggered simply by clicking or tabbing into a field; a change needs to be attempted. And 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 will be performed only when data in that specific field is changed.

At the top of the Validation tab of the Options dialog (refer to Figure 3.8), notice the Always and Only During Data Entry choices. The latter tests for validation conditions only when users modify the field in question. When the Always option is enabled, 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 your data is not imported. You will see a note in the Import Records Summary dialog listing how many errors were encountered.

In the case in which the Only During Data Entry option is used, that improper data would be inserted into your database.

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

STORAGE AND INDEXING

Field storage and indexing options are found on the Storage tab in your Field Options dialog; these options control how FileMaker Pro indexes each field in order 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. Commonly fields with this option are simply 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.

Global fields are often used by developers 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 is saved across sessions. In other words, the next time you open your database, whatever value you last entered into a global will have remained. 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 then be specific only to a given user's session. Other users continue to see the default values, and after the database session is closed it 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.

→ For an example of using a global to drive portal behaviors, **see** Chapter 16, "Advanced Portal Techniques," **p. 471**.

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.

NEW Note that FileMaker 8 offers a new feature in the form of variables that are defined within scripts (and by using the `Let()` function, within calculations). These variables exist in memory only 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; in FileMaker 8, the need for global fields has dropped considerably.

→ To learn more about variables in FileMaker, **see** Chapter 15, "Advanced Scripting Techniques," **p. 435**.

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 intuitively makes common sense, 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, a repeating field cannot be manipulated as a whole. You can reference only one specific repetition at a time.

NEW FileMaker 8 extended the usefulness of repeating fields somewhat in 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. Imagine a spreadsheet. Even though an entire row may be blank, the cells are there, ready and waiting for input. If your users are familiar with Microsoft Excel or have been using a paper form for years, it may make sense for you to duplicate the look-and-feel in question, using repeating fields.

In addition to facilitating data entry, you could simulate a related child table with repeating fields.

→ For a detailed discussion of multiple-table solutions, **see** Chapter 6, "Working with Multiple Tables," **p. 157**.

Knowing when to use repeating fields can be somewhat tricky. If you ever have multiple bits of information that belong together—say, a product name and a price—it's always best to create a table and define fields for those items. They are attributes of the same item—a product. You might be able to grasp that Price[3] corresponded to Product[3], but only anecdotally. If another developer followed behind you in your work, that developer wouldn't necessarily make that same assumption.

The Excel spreadsheet example we gave previously could actually be exactly the wrong way to go: A spreadsheet assumes that a single row relates to all the information within that row. Repeating fields set next to each other on a layout do not share that programmatic logic.

We tend to use only repeating fields that genuinely need multiple instances of the same field: a graphics library, for example, or a simple place to store default values.

INDEXING

Databases store data by definition, 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.

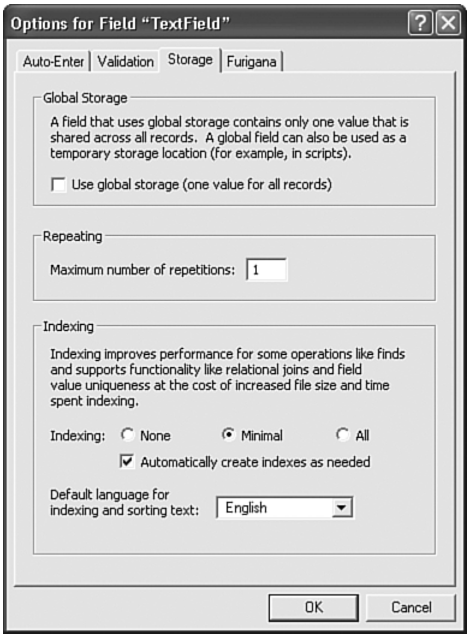
An *index* is somewhat like a database within a database. FileMaker Pro can store, along with a specific value in a given field, a list of all the records in which that exact data is used. This then enables FileMaker to recall those records quickly without having to resort to a linear scan of your file. 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, take a 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 are presented with a dialog box showing all the discrete values indexed for a given field. Just as with selecting from a value list, you may opt to choose from this list rather than type.

Allowing a user to select from an index is only one of the reasons indexes are used in FileMaker. Indexes enable FileMaker Pro to 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, 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 3.10) 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.



Options for Field "TextField"

Auto-Enter | Validation | Storage | Furigana

Global Storage

A field that uses global storage contains only one value that is shared across all records. A global field can also be used as a temporary storage location (for example, in scripts).

☐ Use global storage (one value for all records)

Repeating

Maximum number of repetitions: 1

Indexing

Indexing improves performance for some operations like finds and supports functionality like relational joins and field value uniqueness at the cost of increased file size and time spent indexing.

Indexing: ○ None [✓ Minimal] ○ All

☑ Automatically create indexes as needed

Default language for indexing and sorting text: English

Figure 3.10
FileMaker creates either one type of index or both, depending on how a field is defined and used by users.

Value indexes are established by a database's schema definition—as a developer defines fields and builds relationships—and allow for relationship matches and value lists. If a developer creates a serial ID and joins a relationship via such a field, a value index is created for the serial ID field.

Unless a developer explicitly sets a field to generate an index, word indexes are created as users are interacting with and using a given database. They are utilized in text fields for find requests, or 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 may 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 indices. Value indices are possible for all field types; word indices apply only to text fields. The Minimal setting will be an available option only for text fields, and when you see it marked, this indicates that at least one of the two indices exists for the field. There's no straightforward way of determining which. By explicitly setting the field to minimal, FileMaker will create, on demand, either of the two indices based on how the field is used: When a user creates a find request including that field, a word index will be created. If a developer uses the field in a relationship, a value index will be created.

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

→ To explore the vagaries of storage and indexing considerations for calculation fields, **see** "Other Options," **p. 224**.

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, they cannot be used 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 has four alphabets. One is based on glyphs from Chinese, known as Kanji, two are based on phonetic syllables known as Hiragana and Katakana, and the last is our own Roman alphabet, adopted in the nineteenth century for foreign words. When you're working in Japanese, it is possible to render the phonetic equivalents to a Kanji-based block of text. Quite useful when one doesn't know how to read one of the 20,000-plus characters in Kanji.

Suffice it to say that unless you're a student of Japanese (native or otherwise), this tab will likely not attract much of your attention.

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 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 "12/25/2003."

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 21 and 23 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. In the case that you need to interoperate with them, you might need to have your field names 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 when you have completely tested the field in question.

FileMaker Extra: Indexing in FileMaker

One of the more significant changes in FileMaker 7 and 8 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 is one of the reasons why 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.

In FileMaker 8, words can be indexed up to approximately 100 characters. Text fields can be indexed to a total of 800 characters, and numbers can be indexed up to 400 digits. The limits to indexing have been effectively removed.

What this means to developers is that we can now use far more complex concatenated key combinations (ironically there will be less of that in FileMaker 8, given that data can be related across multiple tables), use longer alphanumeric keys, or, as we suggested earlier, introduce a descriptive elements to keys.

In the past, FileMaker Pro would identify "Special_Edition_Using_FileMaker_8" (32 characters) as identical to "Special_Edition_Using_MS_Access"—clearly a terrible mistake to make. It's now possible to match against paragraphs of text or very large numbers. Determining matches will be more exact and finds and sorts more robust.