# The Calendar Conundrum

Create a relationally sound calendaring system in FileMaker Pro 8 or 7. It's easier than you might think.

*By John Mark Osborne, Database Pros president and owner & FileMaker Advisor technical editor*

Ask any FileMaker Pro developer what it takes to create a calendar solution with event scheduling and monthly, weekly, and daily views. The first thing to pop to mind? It's most likely a ton of complicated calculation formulas and relationships. Opening Define Database for your average calendar solution presents a vast array of relationship lines and table occurrences, not to mention page-long calculation fields. One day, I decided to create a simple calendar solution with all the same features as commercial FileMaker Pro products. While I can't describe the entire solution in a single article, I can give you enough knowledge to create a variety of different views and event scheduling with just a couple relationships and simple calculations.

## The tables

I use a GUI table interface to display the monthly, weekly, and daily calendar views. The GUI table is called VIEW and contains mostly global and calculation fields. The VIEW table contains a single record. I introduce the fields and table occurrences inside the VIEW table throughout this article. For now, just think of it as the interface portion of the calendar solution.

The first data table is called DAYS and contains one record for every day of the year you want to display in your calendar. For instance, if you want to display calendars for the years 2005 and 2006, the DAYS table will contain approximately 730 records or 365 multiplied by two years (depending on whether there's a leap year or not). The DAYS table has very few fields so it doesn't balloon in size as you add more years to your solution. And with one record for every day, it's relationally sound, providing the flexibility to output any type of report or create any type of query. Most calendar solutions merely mimic the existence of records through complicated formulas and a lot of relationships, complicating input and output.

The second data table, EVENTS, contains one record for every time slot. You can choose to have a time slot for every 60 minutes, 30 minutes, or 15 minutes. I chose 15-minute increments to allow for the most flexibility in scheduling an event. In a 24-hour day, there are 96 15-minute increments, translating to approximately 35,040 event records for each year. That's a lot of records, but EVENTS minimizes the size of the table by limiting the amount of data on each record.

In a test with 365 days and the corresponding events, the size of the file is only one megabyte. Adding another year of data doubles the size of the file. So, 10 years worth of dates and events creates a 10-megabyte file. That's reasonable, but why not decrease the file size by only creating event records as they're needed? The advantage of creating a record for every 15 minutes in a day, regardless of a scheduled event, becomes clear when you see the flexibility and simplicity of displaying calendar information. While adding event records as needed lets you perform finds, print specialized reports, and navigate from a find result to a calendar view, you can't display the daily calendar as a graphical representation of your entire day, much like you'd see on a Palm device or a non-FileMaker Pro calendar product such as iCal. Therefore, I chose to create records for every time slot, regardless of whether there's a scheduled event. If you decide differently, there will be no difference when implementing this solution except in the appearance of the weekly and daily views.

## The data fields

I start with the fields where FileMaker Pro stores data. FileMaker Pro only stores data in the DAYS and EVENTS tables. As promised, there aren't very many fields. The only field you must have in the DAYS table is a date field, which I call "Date." The EVENTS table is a little more complicated and requires several fields.

Technical Editor John Mark Osborne is president and owner of Database Pros, offering the largest free FileMaker resource on the Internet. John Mark is internationally recognized as the author of Scriptology, a speaker at the FileMaker Developer Conference and Macworld conferences, and a trainer for the Professional Training series created by FileMaker, Inc. http://www.databasepros.com, jmo@filemaker.com

You must have a date field titled "Date" to relate to the DAYS table, a time field called "Time" to store the time of each 15-minute increment, and a text field called "Title" to store the title of a scheduled event. You create other fields in the VIEW table, but you don't need any other fields for the data tables.

## Populating the tables

Before you create the relationships to display the calendar views, you must populate the DAYS and EVENTS tables with some data. The following script and sub-script create day and event records for a single year at a time. It's a greatly simplified script compared to what I use in the commercial version of my calendar solution, but I wanted to distill the script to the basics. I figure most people will modify the script to suit their particular needs, so just the guts of the script are important. Here's the "Generate Days" script:

```
Show Custom Dialog ["Date"; "Enter the year you wish to
create"; VIEW::xStartYear]
Set Variable [$Date; Value:Date(1; 1; VIEW::xStartYear)]
Go to Layout ["DAYS" (DAYS)]
Freeze Window
Loop
    New Record/Request
    Set Field [Days::Date; $Date]
    Perform Script ["Generate Events"; Parameter: DAYS::
Date]
    Exit Loop If [Year(DAYS::Date + 1) = Year($Date) + 1]
    Set Variable [$Date; Value:$Date + 1]
End Loop
```

The Generate Days script asks the user to enter the year for which they want to create days. This step converts the year into a date corresponding to the first day of the year. You should place the date into a script variable instead of a global field because variables are much faster to increment in a looping script. In addition, script variables don't clutter up Define Database. If you have FileMaker Pro 7, you can substitute global fields for script variables and Set Field steps for Set Variable steps.

The script then selects a layout displaying records from the DAYS table and freezes the window to speed up the loop construct that follows. Record looping scripts are slow because the screen redraws each time FileMaker Pro displays a different record. The Freeze Window script step dramatically increases the speed of the looping script by eliminating screen refresh.

The loop is fairly basic, simply incrementing the $Date script variable by one day and creating a new record with the new date each time the loop repeats. FileMaker Pro stores dates as a number of days, so you only have to add a number to a date field. The result of the addition is still a date, but it's increased by the number of days

you added. There's no need to use the Date function if all you want to do is add a number of days.

The loop finally exits when the year in the Date field on the current record plus one day is equal to the year of the script variable plus one year. This works because the Set Field step that places the date on the new record is before the Exit Loop If step, and the Set Variable step that increments $Date is after the Exit Loop If step. In other words, adding one day to the date on the current record equals the year of the script variable plus one only on the last record for the year, causing the loop to exit. This creates one record for each day in the specified year. Here's this script, which I call "Generate Events":

```
Go to Layout ["EVENTS" (EVENTS)]
Set Variable [$Time; Value:Time(0; 0; 0)]
Loop
    New Record/Request
    Set Field [EVENTS::Date; Get(ScriptParameter)]
    Set Field [EVENTS::Time; $Time]
    Exit Loop If [$Time = Time(23; 45; 0)]
    Set Variable [$Time; Value:$Time + 900]
End Loop
Go to Layout [original layout]
```

The Generate Events script is called from the Generate Days script each time it loops. Every time you create a day record, FileMaker Pro creates corresponding time slot events for the day. Think of it as a loop within a loop.

The Generate Events script is basically the same as the Generate Days script in the sense that it creates a bunch of records with an incremented value. Each record gets the same date as the DAYS table via a script parameter. The time is incremented in 15-minute intervals by adding 900 seconds to the script variable. Time is stored in seconds, so all you have to do is add a number corresponding to the number of seconds to any time field to increase the value. The loop exits when the $Time script variable reaches the time of 11:45 pm. The result is 96 records for each 15-minute increment in the specified day.

## Month view

Displaying a month view requires you to create two global fields in the VIEW table. You create the xMonth and xYear global fields in the VIEW table, and you only have to populate them through simple navigational scripts that increment or decrement the month and year or drop-down lists. I prefer drop-down lists to navigational buttons simply because they allow more flexibility and speed in selecting a month and year. The only issue with drop-down lists is users usually like to see the month name, but the relationships I create in the following pages require a number representation of the month. Here's a formula that translates a month name into a month number:

```
Position("**JanFebMarAprMayJunJulAugSepOctNovDec";
Left(xMonth; 3); 1; 1) / 3
```

In general, the Position function returns the location of one text string within another text string. You can provide the text string by static text enclosed in quotes, a field, or a function that returns a text string result. With this formula, provide a string of the first three letters of each month as the string to search or first parameter. You only need to provide three letters to uniquely identify each month name. The left three letters of the contents of the xMonth global field determine the second parameter, or the search string. The third parameter tells the Position function to start searching at the beginning of the first parameter. The fourth parameter tells the Position function to find the first occurrence of the search string or second parameter.

If xMonth contains "February," the Position function locates "Feb" in the string of abbreviated month names at the starting position of six characters. Notice in the formula, the asterisks at the beginning of the string in the first parameter. These can be any characters and are used to pad the string so the positions of each month name abbreviation return a position divisible by three. In other words, "Feb" returns a position of 6 and is divided by three to result in a 2, which corresponds to the correct month number for February.

If you're really set on using buttons to navigate to the next and previous months, here are some simple scripts that modify the xMonth and xYear global fields. In this case, the xMonth global field is a number type because it holds month numbers and not month names. Here's the script to navigate to the next month:

```
Set Field [VIEW::xMonth; Case(VIEW::xMonth = 12; 1; VIEW::
xMonth + 1)]
Set Field [VIEW::xYear; VIEW::xYear + Case(VIEW::xMonth =
1; 1)]
```

The script simply adds 1 to the month. When the month reaches 1, the year is incremented by 1 as well. There's one exception for the Set Field step that increments the month. If the month reaches 12, the calculation has to return 1 rather than 13. You can easily handle this with a Case statement. To navigate to the previous year, you can duplicate the above script and change some of the values from 1 to 12 and some of the operators from plus to minus. Here's the modified script:

```
Set Field [VIEW::xMonth; Case(VIEW::xMonth = 1; 12; VIEW::
xMonth - 1)]
Set Field [VIEW::xYear; VIEW::xYear - Case(VIEW::xMonth =
12; 1)]
```

If you want to decrease the number of scripts, you can use a single script to handle next and previous months. To accomplish this improvement, you must employ a script parameter assigned at the button level to pass the word "Next" or "Previous" to the script, combine the Next and Previous scripts, and add an If statement checking for the script parameter, like so:

```
If [Get(ScriptParameter)  = "Next"]
    Set Field [VIEW::xMonth; Case(VIEW::xMonth = 12; 1;
VIEW::xMonth + 1)]
    Set Field [VIEW::xYear; VIEW::xYear + Case(VIEW::
xMonth = 1; 1)]
Else
    Set Field [VIEW::xMonth; Case(VIEW::xMonth = 1; 12;
VIEW::xMonth - 1)]
    Set Field [VIEW::xYear; VIEW::xYear - Case(VIEW::
xMonth = 12; 1)]
End If
```

The next part of the process is to create the calculations for the relationships. You create the calculations in the VIEW table, so there's no overhead in the data tables. The calculations determine the lowest and highest date for the month based on the contents of the cMonth calculation field (or xMonth field, if you used scripts to navigate) and xYear global field. Here's the formula for the MonthLow calculation:

```
Let(

FirstDay = Date(cMonth; 1; xYear);

FirstDay - DayOfWeek(FirstDay) + 1

)
```

The first part of the formula, where you transform the xMonth and xYear global fields into a date using the Date function, is pretty easy to understand. The result is the first day of the user-specified month and year, which is all you really need if you want a relationship to display all the records for a particular month and year. However, many monthly calendars show the entire first and last week, regardless of the current month. In other words, you see some of the days from the previous month at the beginning of the current month, and some of the days from the next month at the end of the current month.

The second part of the calculation aims to calculate the number of days in the week preceding the first day of the month. It does this by using a formula that determines the first day of the month and then subtracts the day of the week. Days of the week are numbered starting with 1 for Sunday and incrementing to 7 for Saturday. For example, February 1, 2006, falls on a Wednesday, or the fourth day of the week. Subtracting four days from the first day of the month results in January 28, 2006. Since the 29th is the Sunday of the first week, the calculation adds one to the result.

The MonthHigh formula is virtually the same except it adds days to the last day of the month. The last day of the month is determined by adding 1 to the month and subtracting 1 from the result. Subtracting one day from the date result has been omitted because the calculation adds days to the end anyhow. Here's the formula:

```
Date(cMonth + 1; 1; xYear) + 14
```

There's no need to determine the number of days in the last week of the year. In a standard calendar display of six weeks, there could be as many as two weeks blank at the end of a February calendar if the first day of February happens to be a Sunday. Therefore, the formula adds 14 days to every month. There's no need to determine the exact number of blank days at the end of a month, as you'll see when I reveal the trick for displaying the related records. The calendar simply doesn't display extra days if they don't fit into the six-week rows format.

Now take a look at the relationship for the month view (figures 1 and 2). The calculation fields from the

VIEW table relate to the Date field in the DAYS table using the less than or equal and the greater than or equal operators. This lets all the dates that fall between the calculated dates appear in a portal.

However, a single portal listing the results of the relationship doesn't look much like a monthly calendar. To get the calendar to display in six-week rows, you need to use 42 one-row portals that all display a different row of the relationship. When you fit all the portals together, they look just like a monthly calendar (figure 3).


***Figure 1:* Month view relationship** — The month view relationship relates the VIEW and DAYS tables using multiple predicates to display an entire month of dates.


***Figure 2:* Month view key fields and operators** — The MonthLow and MonthHigh calculation fields from the VIEW table relate to the Date field in the DAYS table using the greater than or equal and the less than or equal relationship operators.

## A few more details

Figure 3 reveals a few details I haven't yet explained. I'll start with the display of the day number because it's the easiest to describe. All you must do is place the Date field from the Month_DAYS table occurrence inside each of the one-row portals. This displays the date for each portal. Next, select the Date field and use the Date item from the Format menu to change how the date displays. You want to use a custom setting that only displays the day from the Date field (figure 4). Don't forget to empty the four entry fields to the right of the pop-up menus; they contain commas and spaces you don't need when displaying just a day number.

Highlighting the days from the current month is a little trickier. First, it requires you to create a Container


***Figure 3:* Finished monthly view** — Here's an example of what the month view might look like once all the one-row portals from the same relationship are arranged on a layout.


***Figure 4:* Date format dialog** — Displaying the day number inside each one-row portal is as easy as formatting the Date field to only display the day number.

field called "MonthHilite" in the VIEW table. This isn't a global Container field; it's a regular Container field. If you use a global field, the technique doesn't work. Draw a small white rectangle using the FileMaker Pro tool in layout mode. It doesn't have to be big, so use the Object Size palette to resize it to one pixel by one pixel. Cut the rectangle to the clipboard, enter browse mode, and paste it into the Container field. Again, you should only have one record in the VIEW table.

Next, create two new calculations in the VIEW table very similar to the calculations you created in the

previous relationships to display the one-row portals. Here's the formula for MonthLowHilite:

```
Date(cMonth; 1; xYear)
```

This calculation only figures out the first day of the month because you only want to highlight the portals displaying days from the current month. So the calculation for MonthHighHilite calculates the last day of the month:

```
Date(cMonth + 1; 1; xYear) - 1
```

FileMaker Pro uses these calculations in a relationship from the Month_DAYS table occurrence to the new Month_Hilite_VIEW table occurrence based on the VIEW table. The new relationship diagram is shown in figures 5 and 6. You must index the results from these calculations for the relationship, so you can't create them as calculation fields because they're based on global fields. You can't index calculations based on global fields, summary fields, related fields, and unstored calculations. The solution is to use the auto-enter calculation feature with the option to uncheck "do not replace existing value of field (if any)". Because the calculations reference the cMonth and xYear fields, any time the global or calculation field changes, the calculation returns a new result.

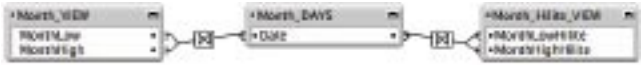

***Figure 5:* Highlighting portal rows with a relationship** — The new table occurrence for highlighting the days from the current month is based on the VIEW table relating to the DAYS table using multiple predicates.

The last step is to place the MonthHilite Container field in each one-row portal. You want to make the field as large as the portal so the color fills the entire portal. The largest you can make the field, and still have it display properly within the portal, is two pixels smaller in height and width than the portal. You also want to set the Container field to enlarge and uncheck the option to maintain original proportions via the Graphic item under the Format menu. This lets the one-pixel by one-pixel rectangle fill the entire Container field.

Highlighting the current day is very similar to the highlight for the current month. You need a Container field called DayHilite with a small swatch of color in it. FileMaker Pro displays the DayHilite Container field through a relationship from a new field called DateCurrent to the Date field in the Month_DAYS table occurrence (figure 7). This is a simple relationship based on the equals operator (=).

Because you must index the DateCurrent field, the best solution is to auto-enter this calculation:

```
Evaluate(Quote(Get(CurrentDate)); [xMonth; xYear])
```

*Continued*

***Figure 6:* Portal row highlight operators** — The MonthLowHilite and MonthHighHilite calculation fields from the VIEW table relate to the Date field in the DAYS table using the greater than or equal, and the less than or equal relationship operators.

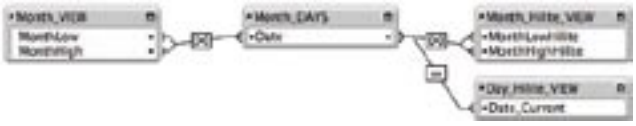

***Figure 7:* Highlighting the current day** — The Day_Hilite_VIEW relationship lets a highlight color from a Container field display on the current day.

The Evaluate function lets the xMonth and xYear fields trigger the update of the current date. This ensures the current date is updated whenever the calendar month or year changes.

All you have to do is place the DayHilite Container field in the one-row portals on the layer above the MonthHilite field. This lets the DayHilite cover the MonthHilite. It's also important to set the DayHilite field to a transparent fill so the field doesn't block the MonthHilite color except on the current day.

## Other views

Displaying a weekly and daily calendar view is fairly simple once you've mastered the monthly view. All you need are new sets of table occurrences, but with different key fields. For instance, the weekly view uses the xWeekStart and xWeekEnd global date fields to create a relationship from the VIEW table to the DAYS table using the same less than or equal, and greater than or equal relationship operators. The DAYS table is then related to the EVENTS table based on the Date field (figure 8).

***Figure 8:* Week view relationship** — These table occurrences create the relationships needed to display a particular week's events.

If you create a layout showing records from the Week_VIEW table occurrence, you can create a portal displaying records from the Week_EVENTS table occurrence. Much like the monthly view, you want to split the portal into seven portals, each displaying 96 rows. Simple navigational scripts can update the global fields to the next or previous week, or a drop-down list can list the week number with calculation fields transforming that week number into the beginning and end week dates.

At this point, you can start to see what I mean by a graphical representation of a day's events. When you look at the whole week, you can see all the events that occur at the same time in the same area on the layout. In other words, if you have a recurring event over the entire week, the events display in the same place on the five adjacent portals so it's easy to identify your week visually. If you display all 96 rows for each of the seven portals, scrolling the window scrolls all the portals together. If you just add your events as you need them, one day might display the recurring event at the top of the portal if the morning's events are light, and the next day might list the recurring event at the bottom of the portal if the morning contains a lot of events.

Creating a daily view is almost the same as a weekly calendar except you only need one date field. The xDate field from the VIEW table relates to the Date field in the DAYS table, which relates to the Date field in the EVENTS table. It's that simple. Displaying the daily view requires a single portal displaying 96 rows.

## Almost done

You can manually schedule events by entering them into the records on the weekly or daily views. But for more control, it's best to enter them with a script. You can also add a calculation to the monthly view to display a summary of the events for each day. You can even add a table occurrence from the EVENTS table onto the monthly table occurrence grouping to make each day in the month view a scrolling portal of events. With a little work, you can also make the calendar multi-user capable by modifying each relationship to include the account name. What I hope I have given you is a strong foundation for a truly relational calendar solution. It's up to you to add all the bells and whistles to make it your own calendar solution. ADVISOR

# FileMaker®

## Developer Conference 2005

# *Thank you...*

*ADVISOR MEDIA thanks these innovative companies for helping us bring this event to the FileMaker community.*

Developer Connection
www.apple.com

BUDDY SYSTEMS
www.buddysystems.com

excelisys
www.excelisys.com

PRODUCTIVE COMPUTING
www.productivecomputing.com

waves in motion
www.wmotion.com

---

.com Solutions Inc.
www.fmpromigrator.com

24U SOFTWARE
www.24usoftware.com

4 – Sight FAX 6
www.4sightfax.com

beezwax
get to know your data
www.beezwax.net

CHAPARRAL
Chaparral Software & Consulting Services
www.chapsoft.com

Cleveland Consulting, Inc.
www.clevelandconsulting.com

CNS
www.cnsplug-ins.com

FUSION PLUGINS
www.fusionplugins.com

FileBooks Link
www.filebookslink.com

FM Forums
www.fmforums.com
www.fmforums.com

FMPtraining.com
www.fmptraining.com

FMWebschool
www.fmwebschool.com

G-NEXT
www.gnext.co.jp

iSolutions
forward thinking, simple solutions
www.isolutions-inc.com

KEY planning
www.key-planning.com

Meta COMMUNICATIONS
www.meta-comm.com

www.mindfiresolutions.com

MoneyWorks
www.cognito.co.nz

the moyer group
www.moyergroup.com

my FMbutler
www.myfmbutler.com

New Millennium
www.newmillennium.com

Redstone
www.redstonesoftware.com

SOLIANT CONSULTING
www.soliantconsulting.com

troi automatisering
www.troi.com

ARIA
www.uams.edu

VTC
www.vtc.com

WorldSync
www.worldsync.com

FileMaker

FileMaker ADVISOR MAGAZINE

www.AdvisorEvents.com