The Calendar Conundrum
by John Mark Osborne
www.databasepros.com

Creating a FileMaker based calendar solution with event scheduling and monthly, weekly and daily views has always been programmed with complicated calculation formulas and lots of relationships. Opening Define Database for some of the calendar solutions I have seen 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 the commercial FileMaker products with which I have seen. 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 of relationships and very 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. Thew VIEW table contains a single record. You will be introduced to the fields and table occurrences based on 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 2005 and 2006 years, the DAYS table will contain approximately 730 records or 365 days times 2 years (depending on whether there is a leap year or not). The DAYS table will have very few fields so it won't balloon in size as you add more years to your solution. And, with one record for every day, it is relationally sound, allowing you 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 lots of relationships.

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

In a test with 365 days and the corresponding events, the size of the file was only 1 megabyte. Adding another year of data doubled the size of the file. So, 10 years worth of dates and events would create a 10 megabyte file. That's reasonable but why not make the file size even smaller by only creating event records as they are 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 allows you to perform finds, print

specialized reports and navigate from a find result to a calendar view, you won't be able to display the daily calendar as a graphical representation of your entire day much like you would see on a Palm device or a non-FileMaker calendar product like iCal. Therefore, I have chosen to create records for every time slot regardless of a scheduled event. If you decide differently, there will be no difference when implementing this solution except when viewing the weekly and daily views.

The Data Fields

Let's start with the fields where data will be stored. Data is only stored in the DAYS and EVENTS tables. As promised, there aren't very many fields. The only field you will need in the DAYS table is a date field which I am calling "Date". The EVENTS table is a little more complicated and requires several fields. You will need 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. Other fields will be created in the VIEW table but no other fields are required for the data tables.

Populating the Tables

Before we create the relationships to display the calendar views, you will need to populate the DAYS and EVENTS tables with some data. It's going to be a long day if you manually enter the days and events. The following script and sub-script create day and event records for a single year at a time. It is a greatly simplified script compared to what I use in the commercial version of my calendar solution but I wanted to distill the script down to the basics. I figure most people are going to modify the script to suit their particular needs so just the guts of the script are important.

Generate Days

```
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. The following step converts the year into a date corresponding to the first day of the year. The date is placed into a script variable rather than 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 7, global fields can be substituted 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 a different record is displayed. The Freeze Window script step dramatically increases the speed of 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. Dates in FileMaker are stored as a number of days so all you have to do is add a number to a date field. The result of the addition is still a date but increased by the number of days you added. There is 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 will equal the year of the script variable plus one only on the last record for the year, causing the loop to exit. The result is the creation of one record for each day in the specified year.

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 a day record is created, corresponding time slot events are created 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 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 two global fields to be created in the VIEW table. The xMonth and xYear global fields are created in the VIEW table and can be populated through simple navigational scripts that increment or decrement the month and year or drop-down lists. I prefer drop-down lists to a navigational button simply because it allows for more flexibility and speed in selecting a month and a year. The only issue with drop-down lists is users usually like to see the month name but the relationships we will create in the following pages require a number reperesentation of the month. Here is a formula that will translate 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. The text string can be provided by static text enclosed in quotes, a field or a function that returns a text string result. In this formula, a string of the first three letters of each month is provided as the string to search or first parameter. Three letters is all that is needed to unqiuely identify each month name. The second parameter, or the search string, is provided by the left three letters of the contents of the xMonth global field. The third parameter tells the Position function to start searching at the beginning of the first parameter and 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 will locate “Feb” in the string of abbreviated month names at the starting position of 6 characters from the beginning. Notice the asterisks at the beginning of the string in the first parameter. These could be any character and are used to pad the string so the positions of each month name abbreviation return a position divisible by 3. In other words, “Feb” returns a position of 6 and is divided by 3 to result in a 2, which corresponds to the correct month number for February.

If you are really set on using buttons to navigate to the next and previous months, here are some simple scripts that handle the modification of the xMonth and xYear global fields. In this case, the xMonth global field is a number type since it will hold month numbers and not month names. Here is 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 one to the month. When the month reaches one, the year is incremented by one as well. There is one exception for the Set Field step that increments the month. If the month reaches 12, the calculation needs to return 1 rather than 13. This is easily handled with a Case statement. In order to navigate to the

previous year, you can duplicate the script above and change some of the values from one to twelve and some of the operators from plus to minus. Here is 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 need to 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.

```
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. The calculations are created in the VIEW table so there is 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 is the formula for the MonthLow calculation.

```
Let(

FirstDay = Date(cMonth; 1; xYear);

FirstDay - DayOfWeek(FirstDay) + 1

)
```

The first part of the formula, where the xMonth and xYear global fields are transformed 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. That seems like all you need if you are going to have a relationship 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 will 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.

What the second part of the calculation aims to do is calculate the number of days in the week preceding the first day of the month. This is done using a formula that first determines the first day of the month and subtracts the day of the week. Days of the

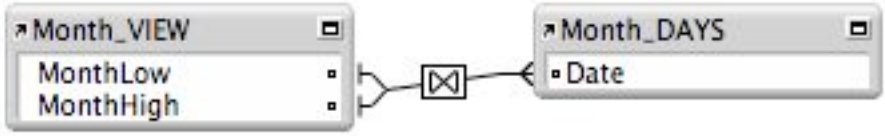
week are numbered starting with 1 for Sunday and incrementing to 7 for Saturday. For example, February 1st, 2006 falls on a Wednesday or the fourth day of the week. Subtracting 4 days from the first day of the month results January 28th, 2006. Since the 29th is the Sunday of the first week, 1 is added to the result.

The MonthHigh formula is virtually the same except that it adds days to the last day of the month. The last day of the month is determined by adding 1 to the month. Subtracting 1 from the result gives you the last day of the month. Subtracting one day from the date result has been omitted since days are added to the end anyhow. Here is the formula.

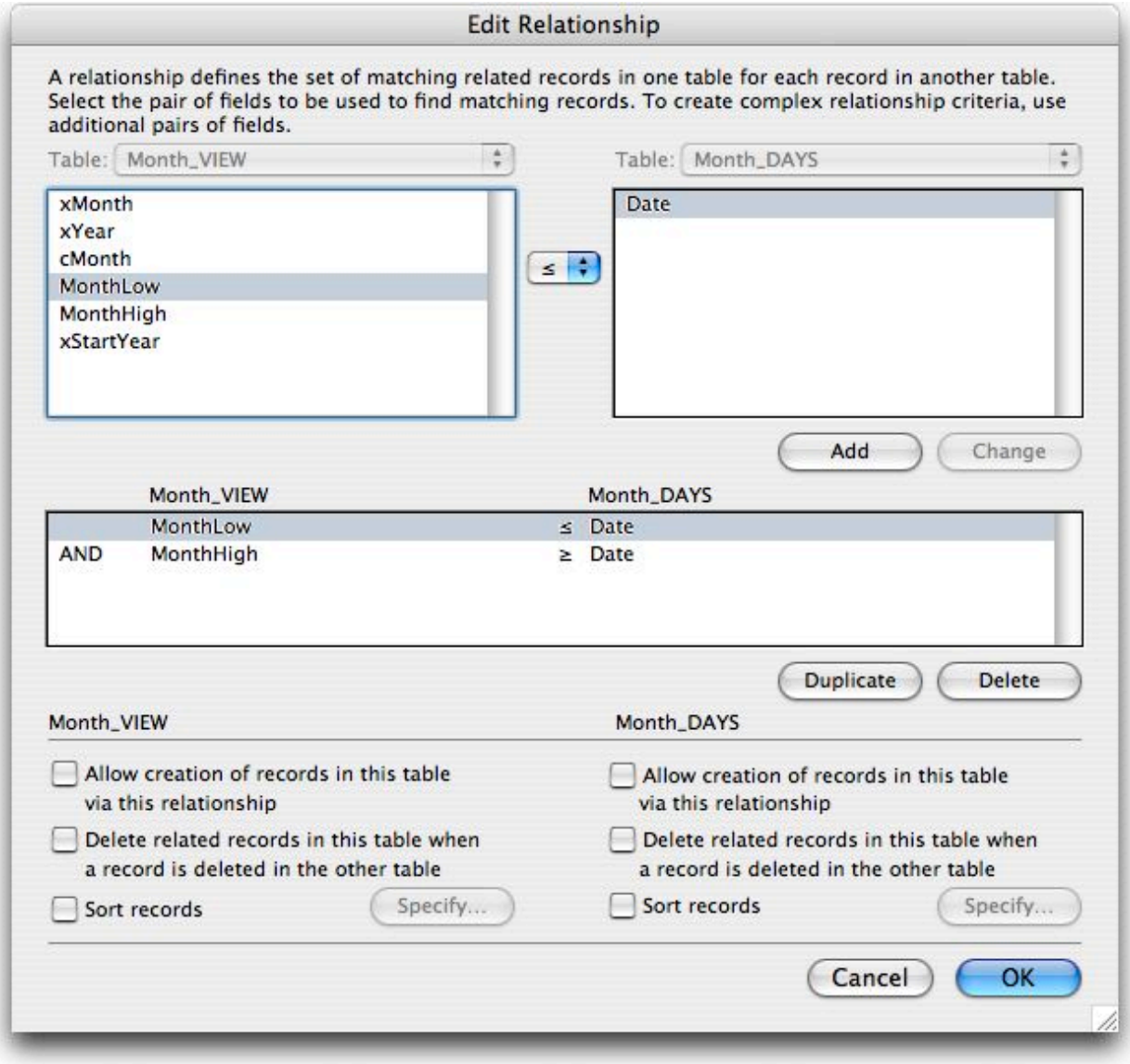
Date(cMonth + 1; 1; xYear) + 14

There is no need to determine the number of days in the last week of the year. In a standard calendar display of 6 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 is no need to determine the exact number of blank days at the end of a month as you will see when the trick for displaying the related records is revealed. Extra days not needed will simply not be displayed in the month view.

Let's show the relationship for the month view (see Figure 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 allows all the dates that fall between the calculated dates to appear in a portal.

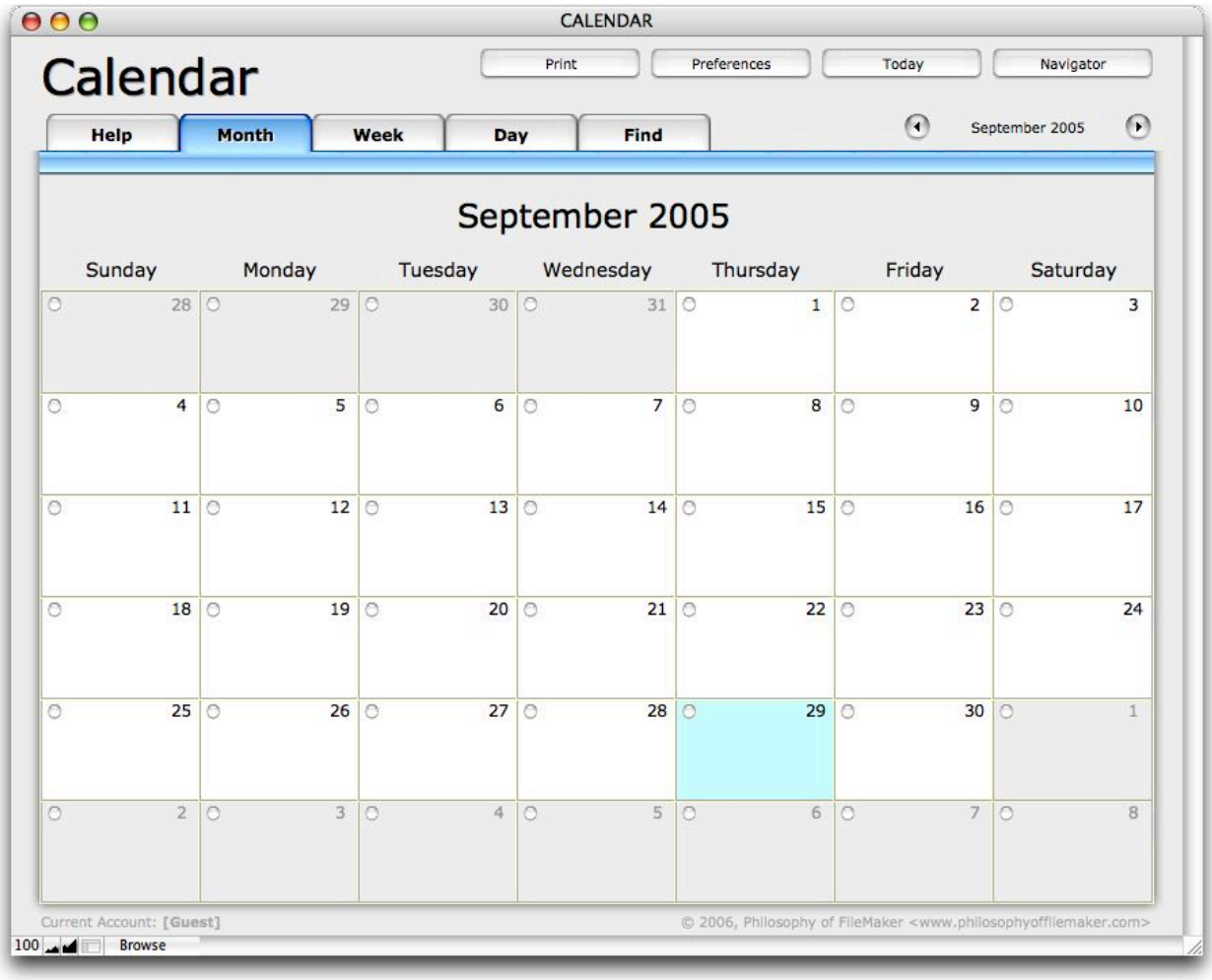


<FIGURE 1> The month view relationship relates the VIEW and DAYS table using multiple predicates to display an entire month of dates.



<FIGURE 2> 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.

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

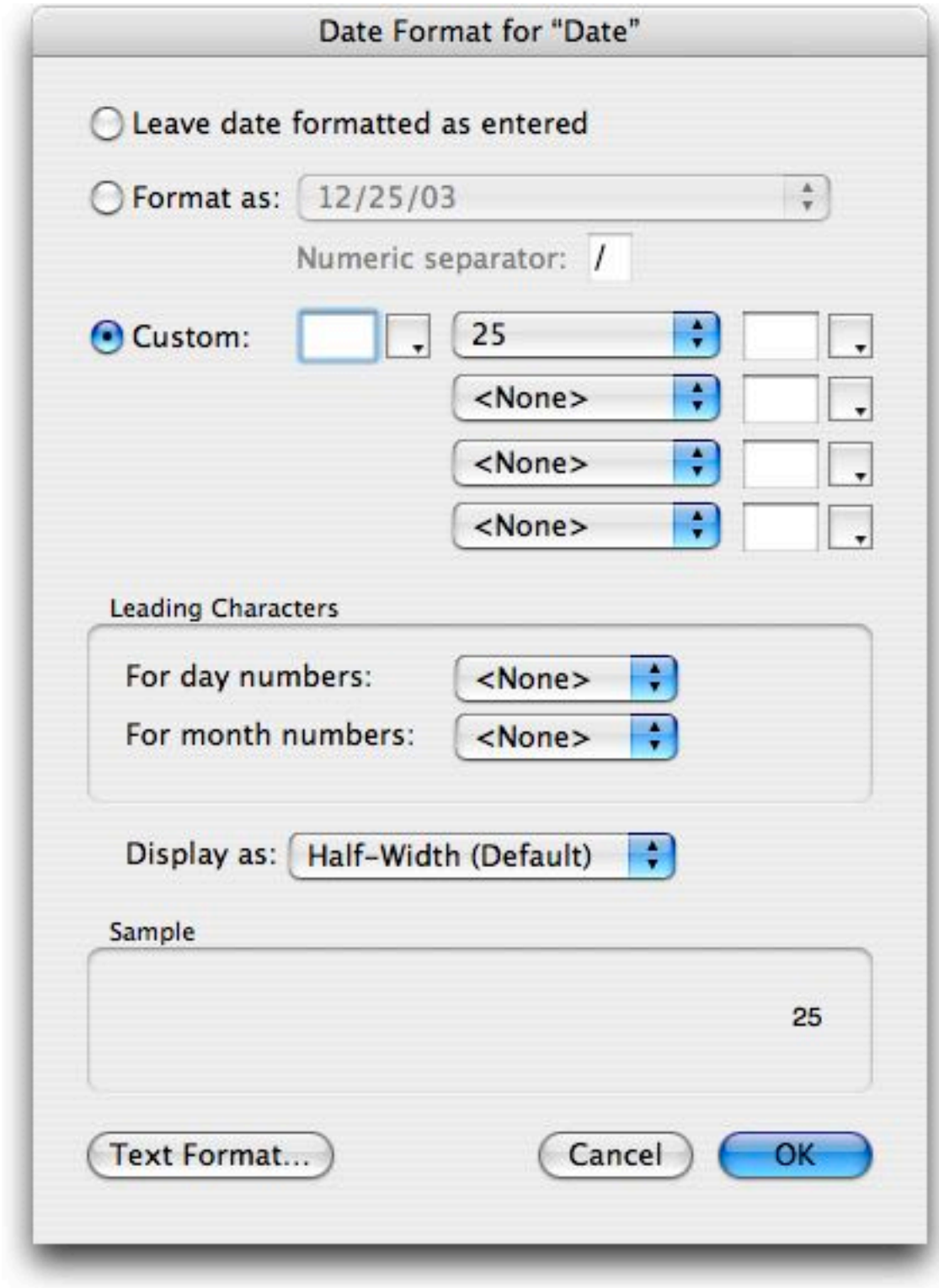


<FIGURE 3> An example of what the month view might look like once all the one row portals from the same relationship are arranged on a layout.

A Few More Details

There are a few details in the example screen shot in Figure 3 that haven't been explained such as how to display the day number, the color highlight for the days from the current month and the color highlight for the current day.

Let's start with the display of the day number since it is the easiest to describe. All you need to do is place the Date field from the Month_DAYS table occurrence inside each of the one row portals. This will display 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'll want to use a custom setting that only displays the day from the Date field (see Figure 4). Don't forget to empty the four entry fields to the right of the popup menus as they contain commas and spaces.



<FIGURE 4> Displaying the day number inside each one row portal is as easy as formatting the Date field to only display the day number.

Highlighting the days from the current month is a little trickier. First, it requires the creation of a Container field called MonthHilite in the VIEW table. This will not be a global Container field but a regular Container field. If you use a global field, the technique will not work. Draw a small white rectangle using the FileMaker tool in layout mode. It doesn't have to be big so use the Object Size palette to resize it to 1 pixel by 1 pixel. Cut the rectangle to the clipboard, enter browse mode and paste it into the Container field. Again, there should be only one record in the VIEW table.

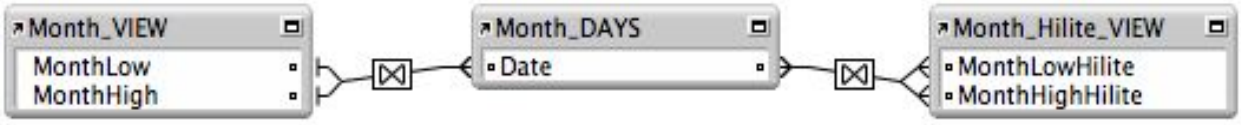
Next, create two new calculations in the VIEW table very similar to the calculations used for the previous relationships used to display the one row portals. Here is the formula for MonthLowHilite.

Date(cMonth; 1; xYear)

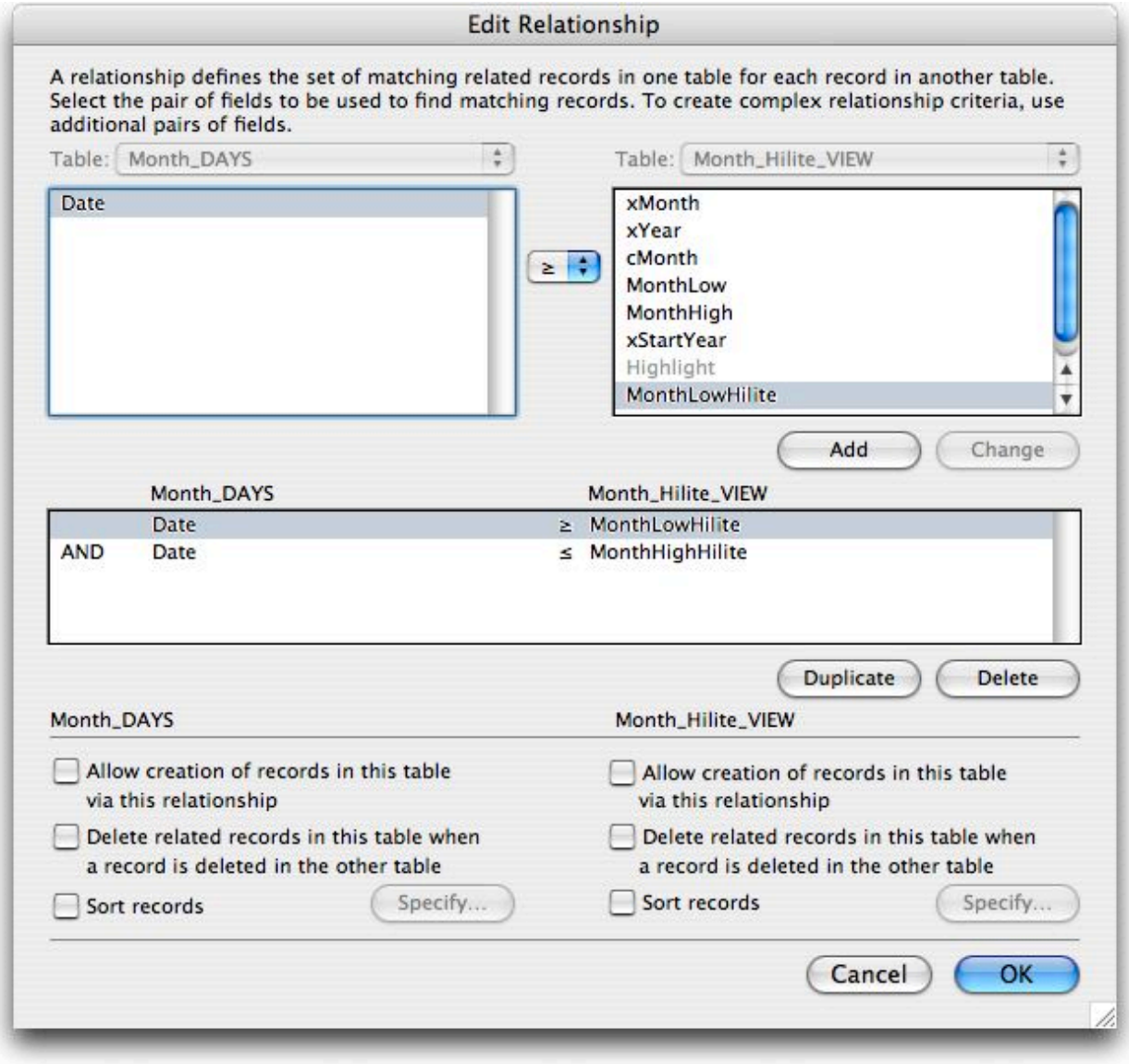
All this calculation does is figure out the first day of the month because you only want to highlight the portals displaying days from the current month. So, the calculation for MonthHighHilite calculates the last day of the month.

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

These calculations are used 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 Figure 5 and 6. Since the results from these calculations need to be indexed for the relationship, they can't be created as calculation fields because they are based on global fields. Calculations based on global fields, summary fields, related fields and unstored calculations can't be indexed. The solution is to use the auto-enter calculation feature with the option to "do not replace existing value of field (if any)" unchecked. Since the calculations reference the cMonth and xYear fields, any time the global fields change, the calculation will return a new result.



<FIGURE 5> 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.

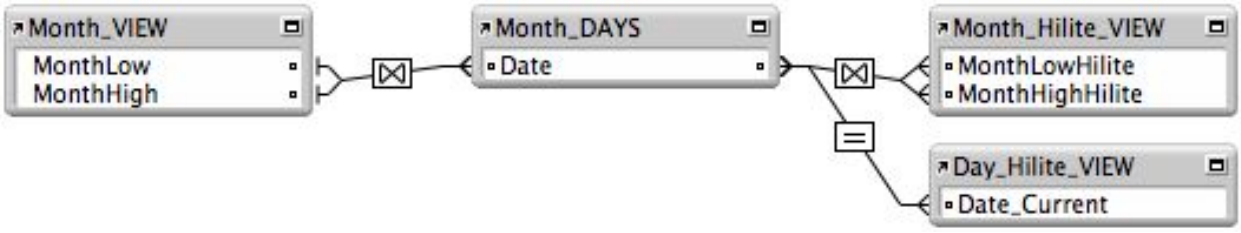


<FIGURE 6> The MonthLowHilite and MonthHighHilite calculation fields from the VIEW table relate to the Date field in the DAYS table using the greater than or equal and the less than or equal relationship operators.

The last step is to place the MonthHilite Container field in each one row portal. You'll 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 2 pixels smaller in height and width than the portal. You'll 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 will allow the 1 pixel by 1 pixel rectangle to fill the entire Container field.

Highlighting the current day is very similar to the highlight for the current month. You'll need a Container field called DayHilite with a small swatch of color in it. The DayHilite

Container field will be displayed through a relationship from a new field called DateCurrent to the Date field in the Month_DAYS table occurrence as seen in Figure 7. This is a simple relationship based on the equals operator (=).



<FIGURE 7> The Day_Hilite_VIEW relationship allows a highlight color from a Container field to display on the current day.

Since the DateCurrent field has to be indexed, the best solution is to auto-enter a calculation.

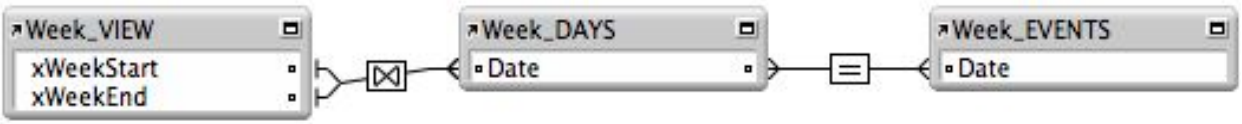
Evaluate(Quote(Get(CurrentDate)); [xMonth; xYear])

The Evaluate function allows the xMonth and xYear fields to trigger the update of the current date. This ensures the current date is updated whenver 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 allows the DayHilite to cover the MonthHilite. It is also important that the DayHilite field be set to a transparent fill so the fill color doesn't block the MonthHilite color.

Other Views

Displaying a weekly and daily calendar view is fairly simple once you have 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 (See Figure 8).



<FIGURE 8> These table occurrences create the relationships necessary to display the events for a particular week.

If you create a layout showing records from the Week_VIEW table occurrence, you can create a portal displaying records from Week_EVENTS table occurrence. Much like the monthly view, you will want to split up the portal into 7 portals each displaying 96 rows. Simple navigational scripts can update the global fields to the next or previous week or a drop-down list could 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 will display in the same place on the 5 adjacent portals so it is easy to identify your week visually. If you display all 96 rows for each of the 7 portals, scrolling the window will scroll all the portals together. If you just add your events as you need them, one day could have the recurring event at the top of the portal because the mornings events are light and the next day could have the recurring event at the bottom of the portal because the morning contains a lot of events.

Creating a daily view is almost the same as a weekly calendar except that 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

Scheduling events can be done manually 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 could also add a calculation to the monthly view to display a summary of the events for each day. You could 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 could 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.