

Access Answers: Working all day

Doug Steele

Doug tries to address commonly asked questions from Access developers. This month, he looks at calculating working days.

DateDiff computes how many days there are between two dates, but I want to only consider working days. Is there an easy way to ignore Saturdays and Sundays?

There are a number of ways to calculate this, some more efficient than others.

For instance, you can easily write a function that loops through each day in the range, only counting those that aren't Saturday or Sunday.

```
Function CrudeWorkDayDiff( _  
    DateFrom As Date , _  
    DateTo As Date _  
    ) As Long  
Dim dtmCurr As Date  
Dim intWeekday As Integer  
Dim lngCount As Long  
  
    dtmCurr = DateFrom  
    Do While dtmCurr < DateTo  
        intWeekday = Weekday(dtmCurr)  
        If intWeekday <> vbSunday _  
            And intWeekday <> vbSaturday Then  
            lngCount = lngCount + 1  
        End If  
        dtmCurr = DateAdd("d", 1, dtmCurr)  
    Loop  
  
End Function
```

The problem with code like this, though, is that it's inefficient. It really isn't necessary to look at every single date in the range. It's easy to determine how many calendar days there are between two dates using the DateDiff function. You can also use the DateDiff function to determine how many weeks there are between the two dates. Since you want to eliminate 2 days per week, it should be simple arithmetic, shouldn't it?

```
DateDiff("d", DateFrom, DateTo) - _  
    2 * DateDiff("ww", DateFrom, DateTo)
```

Unfortunately, it isn't that simple. There will be problems with that calculation if the start or end date is on a weekend.

Before I give the proper equation, I think it's important to understand how DateDiff works to calculate weeks between two dates. There are actually two different arguments you can pass to DateDiff that will calculate weeks ("w", for weekday and "ww", for week). They work differently, but they both sort of return the number of weeks between two dates.

If you use "w" as the interval parameter, DateDiff looks at what weekday DateFrom is, and counts the number of times that day occurs between DateFrom and DateTo. It counts DateFrom, but not DateTo.

If you use "ww" as the interval parameter, DateDiff counts calendar weeks by counting how many times the first day of the week occurs between the two dates. Note that there's an optional fourth parameter that can be provided to the DateDiff call to specify the first day of the week. If you don't specify, VBA assumes Sunday is the first day. If you specify the same first day of the week to the DateDiff function as the DateFrom's day of the week, the two calculations will be identical. In other words, DateDiff("w", DateFrom, DateTo) will always be the same as DateDiff("ww", DateFrom, DateTo, Weekday(DateFrom)).

I think a visual aid may be in order. Take a look at Figure 1, which shows a calendar indicating what the days of the week are for the date range of 9 Sep, 2004 to 21 Sep, 2004. Ignoring DateFrom but including

DateTo, there is one Thursday in the range, two each of Friday, Saturday, Sunday, Monday and Tuesday, and one Wednesday.

Now look at Figure 2, which shows the differences calculated by DateDiff for each of the possible value for First Day of the Week. Those counts above correspond to what DateDiff("ww", DateFrom, DateTo, vbxxx) returned (where xxx is replaced with the weekday name, such as vbSunday, vbMonday and so on). Note, however, that the value calculated by both DateDiff("d", DateFrom, DateTo, vbxxx) and DateDiff("w", DateFrom, DateTo, vbxxx) do not depend on the value of vbxxx.

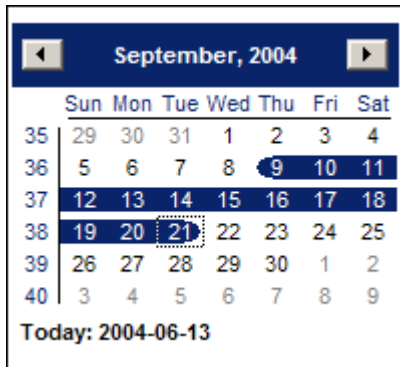


Figure 1: Date Range 09 Sep, 2004 to 21 Sep, 2004

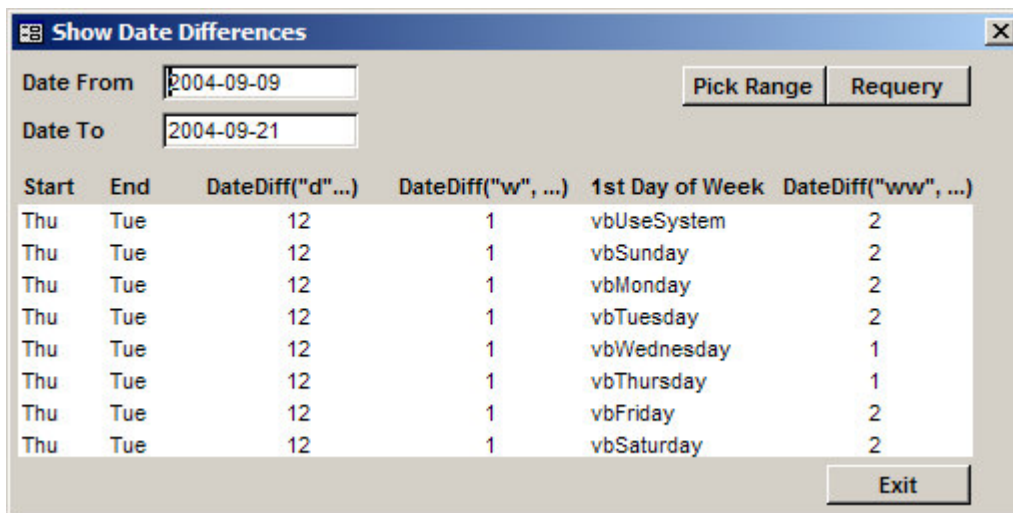


Figure 2: DateDiff calculations for Date Range 09 Sep, 2004 to 21 Sep, 2004

Since I want to eliminate Saturdays and Sundays from my calculations, I'm going to use DateDiff with the "ww" interval, with vbSunday as the first day of the week. DateDiff("ww", DateFrom, DateTo, vbSunday) counts how many times Sunday falls between the two dates, and I can work with that. (vbSunday is the default, but I'm going to be explicit, just in case the user's preferences override the default.)

But knowing how many weekends occur in the range isn't enough to be able to accurately calculate the difference. As I mentioned above, whether or not the start or end date is on a weekend is important. It turns out that if the end date is a Saturday, then the calculation will be too large by a day, unless the start date is also on a Saturday. If the end date is not a Saturday but the start date is, then the calculation will come up short by a day. All other combinations are fine.

Armed with this information, we can create the following function:

```
Function WorkDayDiff( _
    DateFrom As Date , _
    DateTo As Date _
) As Long
```

```

WorkDayDiff = DateDiff("d", DateFrom, DateTo) - _
DateDiff("ww", DateFrom, DateTo, 1) * 2 - _
IIf(Weekday(DateTo, 1) = 7, _
    IIf(Weekday(DateFrom, 1) = 7, 0, 1), _
    IIf(Weekday(DateFrom, 1) = 7, -1, 0))

```

End Function

Note that I'm using the actual values of the constants vbSaturday and vbSunday (7 and 1, respectively) in the function to save space (I'm limited as to how many characters of code I'm allowed to show on a single line in this column!). What I'm really saying is DateDiff("ww", DateFrom, DateTo, vbSunday), and all of the Weekday comparisons are of the form Weekday(DateValue, vbSunday) = vbSaturday.

The IIf statement may be a little confusing, so I'll explain it in a bit more detail. Remember that the syntax for the IIf statement is IIf(expr, truepart, falsepart), where truepart is the value or expression returned if expr is True, and falsepart is the value or expression returned if expr is False. I'm trying to arrive at an adjustment number that I'm going to subtract from my other calculations. The first check is whether DateTo is a Saturday. If it is, I check whether DateFrom is also a Saturday. If it is, no adjustment is necessary. If it isn't, I need to subtract one from the rest of the calculation. If DateTo is not a Saturday, again I check whether DateFrom is a Saturday. If it is, I need to add one to the rest of the calculation. If it isn't, no adjustment is necessary.

Some final comments. You don't actually need to create a function: you can put that ugly expression in-line in your query. If you do that, then you must use the numbers, as I did above, since the Jet engine doesn't know anything about the constants vbSunday and vbSaturday. Putting the expression in the query itself has the advantage that you can then use the query from outside of Access. For example, if you have a Visual Basic program, you can have the query defined in your MDB, but use the query from your VB program. (If the query has a user-defined function in it, such as my WorkDayDiff function, you can't do that)

If you are going to use a function, if there's a chance that some of the dates you pass to the function might be Null, make the function arguments Variants, not Dates. (This is especially likely if you're going to use the function in a query, and some of the date fields in the table might be Null)

Now I know how to eliminate weekends from my dates. What about holidays?

Unfortunately, holidays can't be excluded in such a straightforward manner. First of all, there's the fact that holidays differ from country to country (or even state to state or province to province). Then, some holidays fall on specific dates (eg. January 1st), some holidays fall on specific days (eg. First Monday in September) and some have weird and wonderful formulae for their calculation (eg. Easter is the Sunday after the Paschal Full Moon, where the Paschal Full Moon may occur from March 21 through April 18, inclusive, so that Easter is somewhere between March 22 through April 25, inclusive.)

I've always found that the easiest way to handle holidays is to create a table of holidays in your database. (yes, this means you need to include some means of adding new holidays to your application). Once you have such a table, deducting the holidays from your calculation is as simply as determining how many holiday days occur within the range.

In other words, if you have a table Holidays, with primary key HolidayDate, you can figure out how many holiday days occur in your date range as:

```

DCount("*", "Holidays", _
    "HolidayDate Between " & _
    Format$(DateFrom, "\#mm\dd\yyyy\#") & _
    " And " & _
    Format$(DateTo, "\#mm\dd\yyyy\#"))

```

or in SQL as

```

SELECT Count(*)
FROM Holidays
WHERE HolidayDate Between DateFrom And DateTo

```

That means our WorkDayDiff function above could be rewritten as

```
Function HolidayWorkDayDiff( _  
    DateFrom As Date , _  
    DateTo As Date _  
) As Long  
  
    WorkDayDiff = DateDiff("d", DateFrom, DateTo) - _  
        DateDiff("ww", DateFrom, DateTo, 1) * 2 - _  
        IIf(Weekday(DateTo, 1) = 7, _  
            IIf(Weekday(DateFrom, 1) = 7, 0, 1), _  
            IIf(Weekday(DateFrom, 1) = 7, -1, 0)) - _  
        DCount("*", "Holidays", _  
            "HolidayDate Between " & _  
            Format$(DateFrom, "\#mm\dd\yyyy\#") & _  
            " And " & _  
            Format$(DateTo, "\#mm\dd\yyyy\#")  
  
End Function
```

Remember that if any dates in the Holidays table fall on a Saturday or Sunday, they're going to get double-counted. You'd need to alter your DCount statement accordingly if that's a concern:

```
DCount("*", "Holidays", _  
    "HolidayDate Between " & _  
    Format$(DateFrom, "\#mm\dd\yyyy\#") & _  
    " And " & _  
    Format$(DateTo, "\#mm\dd\yyyy\#") & _  
    " And Weekday(HolidayDate, 1) <> 1 " & _  
    " And Weekday(HolidayDate, 1) <> 7 ")
```

(If you look in the accompanying database, you'll see I've actually added another field to the Holiday table, Location, so that you can represent different holidays for different locations.)

Okay, I now have a replacement for DateDiff to compensate for weekends and holidays. What about an equivalent to DateAdd?

That's a little bit more difficult, but it's not impossible.

As before, I'll start with the simpler case of only ignoring weekends. First, if the given start date falls on a weekend, you want to start from the previous Friday instead. A little trick I use is to use the Weekday function, but with a FirstDayOfWeek value of Saturday. That means that Saturdays will return a value of 1, and Sundays will return a value of 2. If the Weekday is less than 3, you need to subtract the weekday value to the date, otherwise you add nothing to the date. The DateAdd function, combined with an IIf statement lets you do this:

```
DateAdd("d", _  
    IIf(Weekday(DateFrom, vbSaturday) < 3, _  
        0 - Weekday(DateFrom, vbSaturday), 0), _  
    DateFrom)
```

Then, since you're trying to ignore 2 days every week, you can actually add a week to your start date for every 5 working days you're trying to add. Remember that you can figure out the integer number of fives in a number using the \ operator:

```
DateAdd("ww", _  
    NumberOfDays \ 5,  
    DateFrom)
```

Of course, you want to use the adjusted DateFrom from step one, so it's actually:

```
DateAdd("ww", _  
    NumberOfDays \ 5,  
    DateAdd("d", _  
        IIf(Weekday(DateFrom, vbSaturday) < 3, _  
            0 - Weekday(DateFrom, vbSaturday), 0), _  
        DateFrom))
```

Now, assuming that the number of working days you're trying to add isn't an exact multiple of 5, there's a remainder of days left to add. The "trick" is to determine whether adding those days will cross a weekend day. If, for example, you're trying to add days to a Thursday, you know you can add 1 day and it won't be a weekend, but if you need to add 2 days, you'll end up with a Saturday, so you actually need to add 4 days to take you until Monday. Hopefully Table 1 illustrates the adjustment that's required.

Weekday	Weekday Days to Add				Max Days	Weekday(..., vbSaturday)
	1	2	3	4		
Monday	1	2	3	4	4	3
Tuesday	1	2	3	6	3	4
Wednesday	1	2	5	6	2	5
Thursday	1	4	5	6	1	6
Friday	3	4	5	6	0	7

Table 1: Adjustments required for remaining days

The column Max Days represents the maximum numbers of days you can add to the day without hitting a weekend day. The column Weekday(..., vbSaturday) shows the value returned by the Weekday function when you use Saturday as the FirstDayOfWeek. Note that $7 - \text{Weekday}(\text{Date}, \text{vbSaturday})$ gives you the same value as Max Days. That means you need something like

```
DateAdd("d", _
    Days + _
    IIf((Days + Weekday(Date, vbSaturday)) < 7, _
    0, 2), Date)
```

Recognizing that the remaining days can be calculated as $\text{NumberOfDays} \bmod 5$, our final formula is

```
DateAdd("d", _
    NumberOfDays Mod 5 + _
    IIf((NumberOfDays + Weekday(Date, 7)) < 7, _
    0, 2), _
    DateAdd("ww", _
    NumberOfDays \ 5,
    DateAdd("d", _
    IIf(Weekday(DateFrom, 7) < 3, _
    0 - Weekday(DateFrom, 7), 0), _
    DateFrom)))
```

(As before, I used 7 rather than vbSaturday in the code above in the interest of space.)

To be honest, I haven't found an easy way to include holidays in the calculation. Just as you need to adjust your starting day if it falls on a Saturday or Sunday, so too do you need to adjust it if the starting day falls on a holiday. Unfortunately, since you have no way of knowing how many days you need to adjust by (you could have 2 or 3 consecutive holiday days), that means that you can't simply use DateAdd in your adjustment: you actually need to create a loop and continue subtracting days until you're finally on a non-Weekend, non-Holiday day.

```
Do While _
    Weekday(DateFrom, 1) = vbSaturday OR _
    Weekday(DateFrom, 1) = vbSunday OR _
    DCount("*", "Holidays", _
    "HolidayDate = " & _
    Format$(DateFrom, "\#mm\dd\yyyy\#")) = 1
    DateFrom = DateAdd("d", -1, DateFrom)
Loop
```

Once you've arrived at a proper starting date, you can go through the same calculation as above. Once you've arrived at a tentative date to, you need to check whether any holidays fall between one day after the original start date (you've already compensated if the start date itself was a holiday) and your tentative date to. If there are any holidays within that range, you need to adjust the date to accordingly. Unfortunately, you could be including yet another holiday when you do that, so you need to loop until no more holidays are found.

As you can see, that's a rather difficult algorithm to explain, and it's messy to implement.

As a result, I've opted to use a variation of the crude approach I showed originally. Yes, it means you have to loop through all of the days, so the time it takes increases as the value for NumberOfDays increases, but it has the advantage that it's understandable!

```
Function CrudeWorkDayAddHoliday ( _  
    NumberOfDays As Long , _  
    DateFrom As Date _  
    ) As Date  
  
Dim dtmCurr As Date  
Dim lngCount As Long  
  
    lngCount = 0  
    dtmCurr = DateFrom  
    Do While lngCount < NumberOfDays  
  
        Do  
            dtmCurr = DateAdd("d", 1, dtmCurr)  
        Loop Until Weekday(dtmCurr, 7) >= 3 And _  
            DCount("*", "Holidays", "HolidayDate = " & _  
            Format$(dtmCurr, "\#mm\dd\yyyy\#")) = 0  
  
        lngCount = lngCount + 1  
    Loop  
  
End Function
```

*Doug Steele has worked with databases, both mainframe and PC, for many years. Microsoft has recognized him as an Access MVP for his contributions to the Microsoft-sponsored newsgroups. Check <http://I.AmDougSteele> for some Access information, as well as Access-related links. You can reach him at AccessHelp@rogers.com, but note that personal replies are not guaranteed. However, **please** don't hesitate to send ideas for future columns or, even better, complete columns!*