vb123.com

Smart Access

Solutions for Microsoft® Access® Developers

Access Interfaces

User Configuration

Rebecca M. Riordan

This month, Rebecca Riordan discusses the importance of giving your users control over fundamental UI elements such as fonts and colors. It's simple to do, and your users will love you for it.

ONE of the fundamental principles of user interface design, and perhaps the most important, is "Put the user in control." Get it right, and users will be amazingly tolerant of the occasional infelicities in your design. Get it wrong, and it doesn't matter how beautiful your color schemes or elegant your graphic elements are—your system will be regarded as fundamentally broken.

Most of us work pretty hard to keep at least the illusion of having our users in control. We provide multiple paths through the application and multiple levels of undo. We're careful to prevent our applications from doing anything that might seem arbitrary or high-handed (you do *do* these things, right?). But developers hardly ever allow the user to control the most fundamental characteristics of the user interface: the fonts, colors, and graphic styles. And yet, as you'll see this month, doing so is (at least technically) a simple process.

The canonical method for providing user customization in Windows applications is, of course, the Windows Control Panel. Access provides partial support for Control Panel settings, but that support is problematic.

If you use the constants shown in the list that's included in the Download file, Access will update its UI colors when the user changes them in the Control Panel. This is great, as far as it goes. Unfortunately, that's not very far.

The first problem is that it's difficult to determine how to map the Control Panel components to the elements of a database application user interface. The vbWindowBackground is straightforward enough, but what color should the background of a textbox be? You can't simply pick something—vbInfoBackground, the background color of a ToolTip, for example—without

September 2004

Volume 12, Number 9

running the risk of surprising your application's users. (Remember that bit about not appearing arbitrary?)

The other problem, of course, is that Access only supports the colors specified in the Control Panel, not the fonts and (most importantly) the font size chosen by the user. There are Windows APIs that allow you to retrieve this information, but you must do so manually and, I'll warn you up front, it gets ugly.

In most situations, custom configuration is simple to implement and matches the needs of your users reasonably well. At any rate, while I've had clients complain about not having any way to change the color of a form, I've yet to have a customer complain that the application didn't automatically reflect the Control Panel settings.

Designing the infrastructure

Figure 1 shows a simple little form from this article's sample database. As you can see, it provides a way for the user to change the font used to display the controls' labels, the background color of textboxes, and the control style.

The first thing that may strike you about the sample is that it's a slightly odd collection of properties. Actually, I picked this set of properties intentionally to make the point that you need to consider what properties you should expose for user configuration. (Really, it was intentional; I wouldn't lie to you about something like that.) With this form, you're working with essentially three different types of settings—fonts, colors, and possibly graphics—but each type appears in a lot of different places.

So, for example, you might allow users to choose one font and size for labels and a different one for control text. Or separate fonts for labels, textboxes, and listboxes. Or separate fonts for active and inactive versions of each of these, or... You get the picture.

This is one of those situations where *more* is not necessarily *better*. If you were to allow users to individually set the visual characteristics of each and every form and control in your application, your good intentions would almost certainly backfire: Your users would be overwhelmed. It's better to select categories of settings and set those categories at a level of granularity

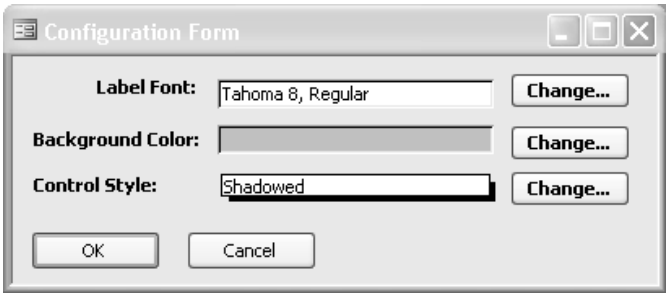

Figure 1. A simple configuration form from the sample database.

that suits the users. Then you need to implement those categories consistently. The only practical exception to this principle is to allow users to set the background color of individual forms. Some users find having each form set to a different color creates a useful sort of "roadmap": They know the pink form is for Customers and the yellow one is for Products, for example.

Once you've determined what characteristics you'll allow users to change, the next step is to determine a persistence strategy. You can persist a set of user selections in a table, in custom properties, or even in an XML file or the Registry (if you're brave). I tend to use tables, simply because I think in a table-oriented way, but you should choose the method that makes best sense in your environment.

You must also determine where the customizations will be persisted. In the canonical multi-user application that splits the front-end interface elements and back-end data, you generally want to store the customization in the front-end database. This works well if there's a more or less one-to-one relationship between users and workstations. This is typically the case—each user has a desk, and each desk has a computer. But if you need to support roaming users who might be working remotely from different workstations, or if a single workstation is shared among multiple users, then it's best to persist the customizations in the back-end database.

Assuming that you're storing the data in a table, you must decide what the structure of the table is to be. **Figure 2** shows a classic normalized version. If there's any possibility that the set of characteristics will change (and

Figure 2. A normalized structure for persisting user customizations.

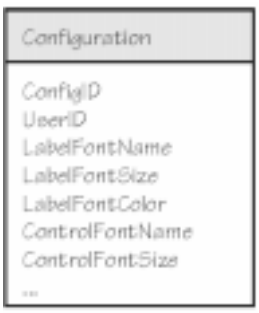

Figure 3. A flattened table structure for persisting user customizations.

it happens more often than you might expect), this is the structure to use.

In this table, each record contains a field for a String value (such as a font name), an Integer value (such as a color value), and a Logical value (as is used by the Bold and Italic settings). For any given row, two of these three fields will be empty, resulting in a very sparse table and some waste of disk space. However, since values of all three types must be stored, the only alternative would be to use a generic field type such as String and cast the values to the appropriate type at runtime, which would have much higher overhead.

Alternatively, you can flatten the table schema as shown in Figure 3. This makes the database analyst in me shudder, but in reality it's a very good solution for the majority of applications. Like any flat structure, it does require that you know in advance what properties you'll allow to be customized, but as you've seen, this will be determined during your initial analysis. Furthermore, if the set of customizable characteristics changes, you'll need to change both the structure of the table and the code that uses it.

Despite these problems, this flat structure has much to recommend it. It requires no joins, and only a single record need be returned from the database in client/server environments. Also, because the field names are known in advance, runtime processing is somewhat simpler. This is the structure I use most often in my own work, and the one demonstrated in the article's sample database.

Coding the common dialogs

Once you've designed your infrastructure, the next step is to build the configuration form for your application. The form in the sample database, shown in Figure 1, demonstrates the three most common techniques that you'll require for this purpose. Each of the buttons labeled "Change..." opens a subsidiary dialog that allows the user to choose the appropriate settings. The first two buttons use standard Windows dialogs, and the third opens a custom form.

The form uses the Microsoft Common Dialog ActiveX control to open the standard Windows Font and Color pickers. You should be aware that the Common Dialog control has some problems when used from Access, and many developers prefer to go directly to the Windows API. My requirements are straightforward, however, and the Common Dialog ActiveX is more than sufficient for my purposes.

The following code shows the Click event code for the single font property on the sample form. Using commonDialogs (the name of the Common Dialog control that I dragged onto the form), the code first sets the control's Flag property to cdlCFScreenFonts so that only fonts that can be displayed on the screen are shown (that is, no printer fonts will be displayed). The code then calls the ShowFont method to display the dialog modally. The result (with the font list on my system) is shown in Figure 4.

```
Private Sub showFonts_Click()
    With commonDialogs
        .Flags = cdlCFScreenFonts
        .ShowFont

        fntName = .FontName
        fntSize = .FontSize
        fntBold = .FontBold
        fntItalic = .FontItalic
    End With

    Me.labelFont = makeFontName

End Sub
```

The Common Dialog control exposes the user's selections as a set of properties that remain available after the font picker itself is closed. My code uses the properties—FontName, FontSize, and so on—to set local variables to the values that the user selected in the dialog. The last line in the procedure calls a utility function of mine, makeFontName, that concatenates the values stored in the local variables. The text of the font textbox is set to the value returned from the makeFontName function.

The next set of code shows the click event handler for the Change button associated with the control background option. It's a simpler version of the previous code. The ShowColor method of the commonDialogs control is called to display the standard color picker shown in Figure 5. Once the user closes the dialog and execution continues, the procedure sets the local variable ctrlColor to the selected color (returned by the Color property of the common dialog) and sets the background of the relevant control on the configuration form:

```
Private Sub showColors_Click()
    commonDialogs.ShowColor
```

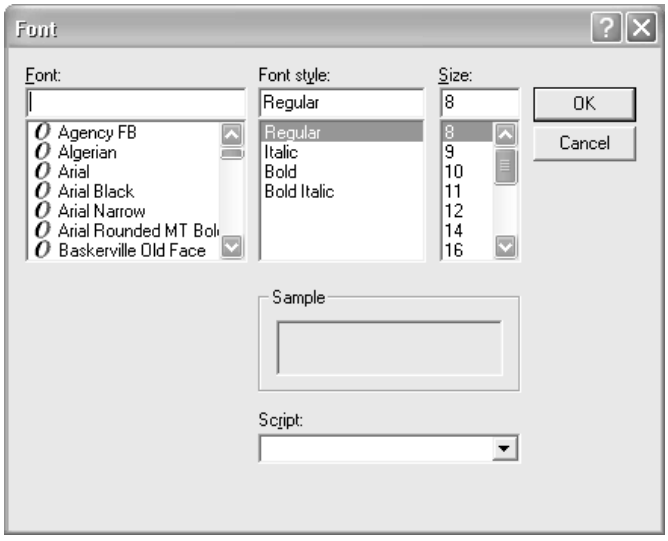

Figure 4. The standard Windows Font picker.

```
    ctrlColor = commonDialogs.Color
    Me.bkgdColor.BackColor = commonDialogs.Color
End Sub
```

Coding custom dialogs

As you can see, these are very simple procedures to implement. This approach also has the advantage of using dialogs that your users are likely to be familiar with. The problem is that these dialogs may provide more functionality than your application might need. The Font picker, for example, allows the user to select the font name, font size, and font style. But you might want to constrain the font size, for example, to avoid having the user select a font that's too big to be displayed in the textboxes on your forms.

In this case, of course, you'll need to create custom dialogs. The third option on the configuration form, Control Style, shows the general procedure. The custom form in the sample database is shown in **Figure 6**. The following code block shows the click event handler that opens it:

```
Private Sub showStyles_Click()
    DoCmd.OpenForm "ControlDisplay", acNormal, _
        , , , acDialog

    If Application.CurrentProject.AllForms_
       ("ControlDisplay").IsLoaded Then

        controlStyle = Forms!ControlDisplay.Tag
        DoCmd.Close acForm, "ControlDisplay"

    End If

    Me.ctrlStyle.SpecialEffect = controlStyle
    Me.ctrlStyle = makeStyleName
End Sub
```

The basic procedure here is identical to the other two procedures you've seen. The difference is in how the dialog itself behaves. The first line of the procedure opens the dialog modally. If the user selects Cancel, the form is closed. If, however, the user selects OK, the form is hidden so that its values remain available to the click procedure that called it.

As you can see, the click procedure uses the IsLoaded property to determine what action the user took—the form will only be loaded if the user clicked the OK button. The code then sets the local variable, controlStyle, and the style of the control.

All that remains is the mechanics of persisting and restoring the values the user has chosen. The following code from the sample database stores the values to the database. Remember that each of the click procedures stored the user's choices in local values. All this procedure needs to do, then, is update the appropriate record using standard data handling techniques:

```
Private Sub btnOK_Click()
    Dim rs As Recordset
    Set rs = CurrentDb.OpenRecordset("Settings")
    With rs
        .Edit
        !FontName = fntName
        !FontSize = fntSize
        !FontItalic = fntItalic
        !FontBold = fntBold
        !Color = controlColor
        !Style = controlStyle
        .Update
    End With

    rs.Close
End Sub
```

For efficiency's sake, the procedure updates all values, whether or not they've been updated by the user. This works perfectly well in the sample since, as you'll see, the local variables are initialized in the form's Load event. If you vary this technique, you'll need to check for Null values here.

A production application would require DAO code to find the appropriate record to update. Because the configuration table in the sample database contains only a single row, I've omitted it from the sample.

The following code loads the stored customizations. The first section of the procedure consists of standard data handling code to load the values into local variables. Again, I've omitted the code to find the appropriate record:

```
Private Sub Form_Load()
    Dim rs As Recordset
    Set rs = CurrentDb.OpenRecordset("Settings")
    rs.MoveFirst

    With rs
        fntName = !FontName
        fntSize = !FontSize
        fntItalic = !FontItalic
        fntBold = !FontBold
```

Continues on page 18

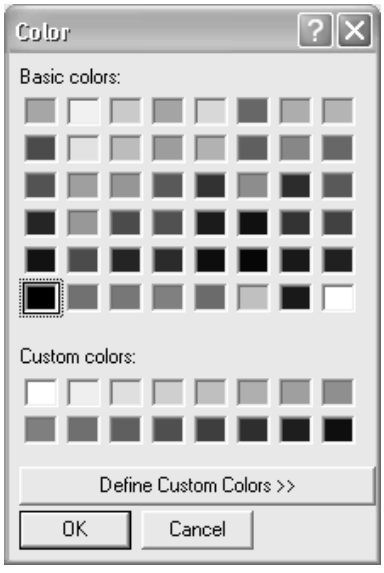

Figure 5. The standard Windows Color picker.

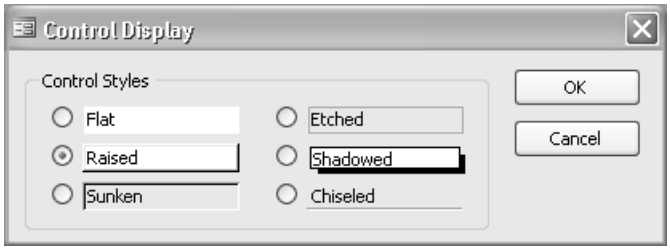

Figure 6. The custom Control Style dialog from the sample database.

FULL PAGE AD:
FMS

Sending E-mail with Access

Rick Dobson

2000 2002 2003 DOWNLOAD

In this article, Rick Dobson explores using the robust database management features of Access to develop e-mail solutions for managing business relationships.

ACCESS is the premiere development tool for small businesses and departments in medium-sized businesses. At the very heart of many businesses are the needs to keep track of and communicate with prospects, clients, vendors, and partners via e-mail. Access is great for keeping track of people, such as prospects and clients. However, Access isn't well known for its ability to send e-mail to people. I'm going to show you a couple of ways that you can leverage Access' outstanding database management development features when sending e-mail.

Of course, Outlook is the general-purpose Office tool for e-mail and related kinds of requirements. Access can create e-mail solutions with Outlook or work independently of Outlook. Depending on your type of e-mail task, you may prefer one approach over the other. For example, Outlook offers rich previewing and editing tools for e-mail. If this is a requirement for your e-mail task, it makes sense to take advantage of Outlook. On the other hand, Outlook has built-in security features that block or restrict the ability of rogue software (such as a virus) from sending e-mail to everyone in its Address Book. Unfortunately, this security measure makes Outlook unfriendly for servicing your legitimate need to send a message to everyone or even a substantial subset of your contacts. In this situation, you can use CDO (Collaboration Data Objects) programmed with VBA from Access to push out e-mail messages to many different recipients. The CDO library isn't known for its ease of use, but I'll show you a fast way to get started programming e-mail solutions with CDO from Access VBA projects.

Two Access e-mail solutions

I don't advocate using Access as a replacement for Outlook. Nevertheless, the database management features of Access make it an ideal development tool for selected types of e-mail applications. I'll highlight two common e-mail solutions that benefit from development with Access.

First, a business may need to send any of several routine e-mail messages to correspondents. The term *routine* applies to those messages that are the same for all recipients, except for a few minor changes (such as the name of the person you're sending it to, the name of a product, or the duration of a subscription or membership). These routine messages will typically thank clients and prospects for their interest in a firm, remind a subscription client that it's time to renew, or confirm with marketing prospects the date, time, and venue for a presentation. Many businesses already have a collection of informal templates for these kinds of e-mail. In addition, each message template will typically feature one or more fields that require editing whenever the message is sent.

You can use Access to store the information about each message as well as data about the fields that can change. Access forms can simplify modifying field values just before sending a message. When sending messages one at a time to individual correspondents, using Access and Outlook together makes a lot of sense. You could, of course, also use Microsoft Word's mail-merge facilities. However, an integrated approach built on Access and Outlook allows you to take advantage of Outlook's editing/previewing features along with Access' data management capabilities (both for tracking messages and the users who are to receive those messages).

A second kind of e-mail application requires sending the same message to all of your contacts or a substantial subset of them. Outlook distribution lists offer one approach to this task, but these lists don't update automatically like queries do. Access queries offer a more flexible means of specifying who can receive a message. You can program Outlook with VBA to repeatedly send the same message to a subset of members from an Outlook Address Book, but Outlook has no way to distinguish between your VBA program and a virus. Therefore, both your program and the virus encounter the well-known security prompt (see Figure 1). This prompt

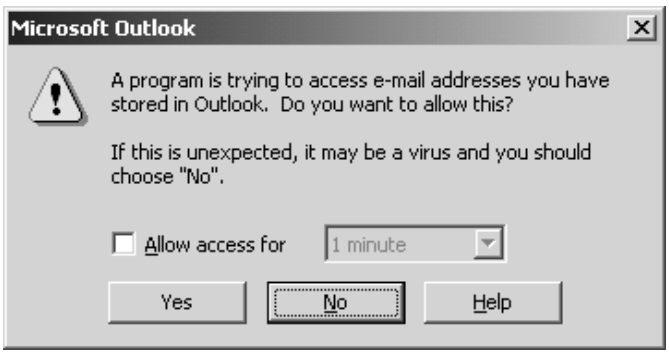

Figure 1. The Outlook security prompt that inhibits sending the same e-mail to a list of recipients.

requires a user to manually click a button, and the prompt will recur at regular intervals if you attempt to send a message to a long list of correspondents.

The security prompt problem has multiple solutions, and I'll show you one based on CDO (in my article in last month's issue, I discussed another solution: Express ClickYes). CDO offers a collection of classes that you can program with VBA. The NewMail objects simplify sending e-mail messages.

Sending one of several e-mails

To send one of several routine messages requires a collection of elements. First, you need to create a set of one or more messages that you save as files. Second, you need to create an Access database for tracking messages. The Access database also stores a list of e-mail correspondents in its Contacts table. The sample database for this article, SA0204.mdb, contains a Contacts table with several rows of data. The application uses the contact information to personalize your messages and to specify where to send an e-mail. Third, you need Outlook. After a user selects a message and performs any customization, the application opens Outlook and populates the fields of a new message to allow you to preview the message. This preview offers a second opportunity to customize a message before clicking the Send button in the Outlook message window to forward the message to the Outbox. Outlook automatically sends the message from the Outbox via a regular schedule or the next time a user clicks the Send / Receive button.

Since the application programmatically manipulates Outlook, you need a reference to the Outlook object library. I developed the sample applications for this article with Access 2002 and Outlook 2002. Therefore, the sample Access database file has a reference to the Outlook 10.0 object library. If you use another version of Office, such as Office 2000 or Office 2003, you need a reference to either the Outlook 9.0 or Outlook 11.0 object library.

You can compose the items in your message collection with any program that edits and saves plain text or HTML. I used Word as the e-mail editor for Outlook with Plain Text selected for the message format in order to generate a message with plain text formatting. This format doesn't permit formatting such as active hyperlinks or even bold text, but you can see what your message will look like in an e-mail reader set to read only plain text (the HTML and Rich Text message format settings in Outlook provide more formatting options). Since Outlook and CDO both support an HTML property for a message or mail object, I selected HTML as the alternative to plain text in the sample application. You can use either Outlook / Word with an HTML output setting or any HTML editor to compose messages in HTML format. I chose FrontPage 2002 as my HTML editor.

The three messages for this sample application have names ClubRenewal.htm, ClubWelcome.txt, and WidgetEnquiry.htm. They reside in the Articles folders of the C: drive on my test computer. You can store your messages wherever you choose, but you must synchronize the paths and file names to the messages with the table that specifies information about the messages.

Within each message, you can use one or more keywords to designate fields that users can update at runtime. You're free to designate any keyword that you prefer. For example, the message in the ClubWelcome.txt file uses the keyword NewMemberName to designate the name of the recipient. On the other hand, the message in the ClubRenewal.htm file specifies the name of the recipient using ClubMemberName as a keyword. The application has built-in mechanisms for tracking keywords in messages. In addition, users can edit the values that replace keywords in messages from Access forms. Two tables—Messages and MessageFields—track the messages for an application along with the message keywords and their last value.

A value is a string that replaces the keyword in sent messages. For example, the keyword NewMemberName can have a last value of Rick Dobson. If Virginia Dobson is the next member to join the club, then the last value of the NewMemberName keyword will change from Rick Dobson to Virginia Dobson.

The Messages table contains a row for each message along with six columns to distinguish each message. The MessageID column is an Autonumber field that provides a unique value as the primary key. The MessageName column contains a short descriptive text field for the message. The Subject column contains the default Subject field value for a message. Users can override this field value from Outlook before clicking the Send button in the Outlook message window. The HTML column has a Yes / No data type. A value of Yes corresponds to an HTML format, and a value of No specifies a plain text format. The Path and Filename columns are two text fields that designate the path to the message file.

The MessageFields table has four columns. The initial MessageID column has a Long Integer value that points back at one of the rows in the Messages table. The FieldID column also has a Long Integer value, which specifies one of the keywords in the message file. Since the message in the ClubWelcome.txt file has four keywords, its FieldID values range from 1 to 4. The FieldName and LastFieldValue columns both have a text type. These columns designate the names of keywords and their last value.

The Contacts table has four columns. The first column is an Autonumber column. The second and third columns are for the first and last name of a contact. The fourth column contains the e-mail address of a contact.

Forms and procedures

Figure 2 shows frmMessages, which is a main / subform that displays Messages table column values in its

main form and matching column values from the MessageFields table values in its subform. The top five textboxes in frmMessages with a checkbox between the third and fourth textboxes are bound to the columns of the Messages table.

The three textboxes immediately above the subform are unbound. These three controls allow you to display contact information. The textbox with a label of "To criteria" lets a user enter any part of the values for the first and last names, such as Rick (for Rick Dobson). Clicking the Find button recovers the first and last names as well as the e-mail address of the first row in the Contacts table that matches the criterion. The application puts the combination of the first and last name in the next-to-last textbox on frmMessages, and it puts the e-mail address in the form's last textbox. In addition, the Click event procedure for the button updates the last column value in the first row of the subform. Therefore, you should always reserve this message field value for the full name of recipients.

The Click event procedure for the form's Send button populates a new Outlook mail item with selected values from frmMessages, such as the recipient's e-mail address and the message's subject, as well as an edited version of the message. Keywords in a saved message are updated with values from the LastFieldValue column in the message to be sent.

The following code is from the Click event procedure for the Find button on frmMessages. The code begins by creating a SQL Where clause to be used against the Contacts table. The constant strDQ holds four double quotation marks, which are required to insert a single double quote into a string. I've used the Like operator to match any sequence of characters in the first and last names that have a space between them. Three DLookup functions populate the next-to-last textbox on the form (txtSenderName) and the last textbox on the form (txtE-mailAddress). The code closes by populating the last column value in the first row of the subform (ctl1.Form.Controls(6)):

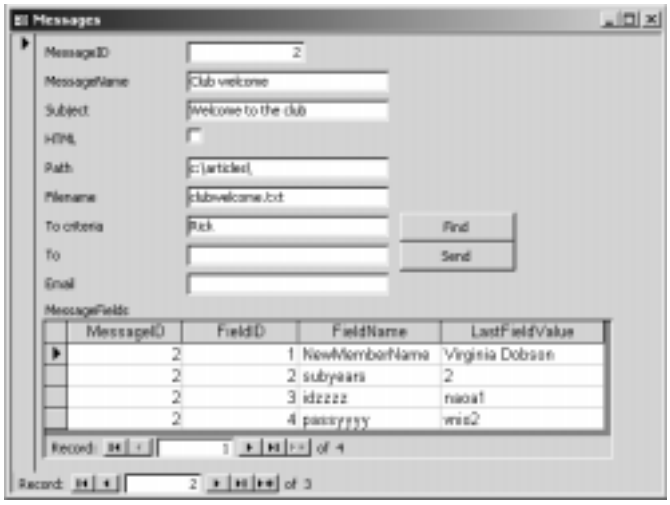

Figure 2. The frmMessages form lets a user look up contacts and update keywords in saved messages before opening a message in Outlook.

```
strSQL = "(Fname & " & "Lname) " & _
    "Like " & strDQ & "*" & _
    txtSenderCriterion & "*" & strDQ

strFirst = DLookup("Fname", "Contacts", strSQL)
strLast = DLookup("Lname", "Contacts", strSQL)
txtSenderName = strFirst & " " & strLast
txtE-mailAddress = _
    DLookup("E-mailAddress", "Contacts", strSQL)

Set ctl1 = ctlMessageFields
ctl1.Form.Recordset.MoveFirst
ctl1.Form.Controls(6) = strFirst & " " & strLast
ctl1.Form.Refresh
```

The next set of code is from the Send button Click event procedure. The code begins by using the FileSystemObject to open the saved message specified in the current row in the frmMessages form. Your VBA project needs a reference to the Microsoft Scripting Runtime for this to work. Next, the code edits keywords in the message based on the values in the LastFieldValue column of the frmMessageFields subform. The code finishes by passing the e-mail address, formatting style for the message, subject field, and a string with the edited message to the SendToSubjBody procedure:

```
str1 = txtPath & txtFilename
Set tst1 = fso1.OpenTextFile(str1, ForReading)
str2 = tst1.ReadAll

Set ctl1 = ctlMessageFields
ctl1.Form.Refresh
ctl1.Form.Recordset.MoveFirst
For int1 = 1 To ctl1.Form.Recordset.RecordCount
    For Each ctl2 In ctl1.Form.Controls
        If ctl2.Name = "txtFieldName" Then
            strFind = ctl2.Value
        End If
        If ctl2.Name = "txtLastFieldValue" Then
            strReplace = ctl2.Value
        End If
    Next ctl2
    str2 = Replace(str2, strFind, strReplace)
    ctl1.Form.Recordset.MoveNext
Next int1

SendToSubjBody txtE-mailAddress, _
    chkHTML, txtSubject, str2
```

The last bit of code from this application that I want to show you is from the SendToSubjBody procedure. This procedure begins by instantiating a variable that points at an Outlook session. Then, the code uses the session to create a new mail item. Before displaying the mail item in Outlook, the procedure populates three properties for the mail item. The To property accepts the e-mail address. The Subject property takes the String value for the message's subject passed to the procedure. The procedure populates one of two properties depending on the formatting for the message. When the message has an HTML format, the procedure assigns the string with the message to the HTMLBody property. Otherwise, the procedure assigns the message string to the mail item's Body property. The final statement in the routine displays Outlook with the

newly composed message. From this point, the user can either click the Send button or perform any final editing prior to clicking the button:

```
Set ola1 = New Outlook.Application

Set mai1 = ola1.CreateItem(olMailItem)
mai1.To = strTo
mai1.Subject = strSubj
If bolHTML = True Then
    mai1.HTMLBody = strBody
Else
    mai1.Body = strBody
End If
mai1.Display
```

Sending the same message to multiple recipients

Sending the same message to multiple recipients is a different kind of e-mail application than the preceding one, which enables users to send any one of several different e-mail messages to one e-mail address. Sending the same message to multiple recipients is suitable for many business contexts. I use this kind of capability to notify registered visitors about new content at ProgrammingMSAccess.com. If you want to get a first-hand feel for what this type of application can do, register for the site's messages at www.programmingmsaccess.com/mygb.htm. Consulting firms can use the capability to inform clients and prospects about new services, staff additions, and white papers or presentations authored by staff persons.

As I mentioned previously, the built-in security prompt in Outlook makes it difficult to perform this kind of task. In addition, Outlook automatically saves a copy of each message e-mailed in its Sent folder. If you're e-mailing the same message to several thousand, or more, recipients, it's unlikely that you wish to keep a copy of each sent message. Finally, when sending the same message (perhaps differentiated by recipient name), you may want to filter your contact list to just a subset of contacts who should get a copy of the message. Outlook does offer filtering, but it doesn't have the same rich query features provided by Access. Using Access for filtering a list of contacts will be substantially more flexible than using Outlook (and more familiar to Access developers like you).

The following sample demonstrates how to use the CDO object model as an alternative to Outlook for sending the same message to multiple recipients with Access. When you program the CDO object model with VBA, you can avoid the Outlook security prompt. In addition, you bypass the whole Outlook user interface, which means you don't automatically retain copies of messages in Outlook folders. You also gain the flexibility to readily specify recipient e-mail addresses from any source, such as an Access query.

The CDO object model used to be available as a separate redistributable, but it now ships exclusively as part of several packages, including Office 2000 through Office 2003. However, CDO doesn't install by default with Office unless you perform a complete installation. The CDO library is available from the cdo.dll file, which typically resides in the local resources folder (1033 for the US) of the \Program Files\Common Files\System\Mapi\ path. You can add a reference to the CDO library by selecting Microsoft CDO 1.21 Library from the Available References list of the References dialog box from the Tools | References menu.

The sample for mailing one message to multiple recipients consists of three elements. First, you need a list of contacts. The sample demonstration uses a query (qryContacts) based on the Contacts table. Second, you need a message. The sample message is in HTML format. The application is set up to accommodate HTML instead of plain text. HTML format offers the ability to include active hyperlinks in a document, which isn't available from messages in plain text format. Many recipients prefer active hyperlinks in messages because the recipient can click the links to view the hyperlinked Web page instead of having to copy a URL into the Address box of their Web browser. The third element is a form, frmBatchMail, which includes textboxes for specifying the parameters of a typical batch mail job and a button that has an event procedure for launching the mail job. The form's Load event procedure automatically populates the textboxes for the sample batch job that I describe here. You can change the settings of the Load event procedure to work with any message and query for

contacts that you prefer.

The qryContacts query simply returns all the columns for all the rows in the Contacts table. Since the sample Contacts table contains just four rows, there's no need to restrict which rows are retrieved. However, a substantially larger Contacts table might contain more columns to support filtering on various criteria. In addition, Access makes it simple to filter on substrings for the Fname, Lname, and E-mailAddress columns. For any application where you're mailing to e-mail addresses, you shouldn't use the Access hyperlink data type to represent e-mail addresses. Instead, use text data types that contain e-mail addresses.

I've saved the sample message with the name ProgrammingMSAccess_Feb_2004.htm in the Articles folder. You can use any other name and folder you choose as long as you update the frmBatchMail Load event procedure or override the default settings in the form when it opens. In addition, the sample message initially designates the recipient's full name (first name, space, last name) with the FullName keyword. You can also override this keyword for initially representing a recipient's full name either from the frmBatchMail form or its Load event procedure.

Figure 3 shows frmBatchMail with the default settings. The Subject textbox is the Subject for the messages that the application pushes out to recipients in the source of contacts, which is specified in the last textbox on the form. The From textbox designates the Display name of the recipient and, optionally, the e-mail address of the sender (that would be you). If you elect to designate a sender e-mail address, place the address in angle brackets after the Display name.

The code behind the form takes advantage of the NewMail class to drastically simplify the use of the CDO library. The NewMail class resides in the CDONTS library. This library is a subset of the CDO library that's available from cdo.dll. NewMail class instances feature a collection of read-only properties that let you set, but not read, the properties for a class instance. A NewMail class instance corresponds to a single e-mail message. After using the instance once, you should discard the instance and make a new one for another message. NewMail class instances permit you to set most common e-mail fields with String or Long values.

Figure 3. The frmBatchMail form with the default settings specified by its Load event procedure.

The following code shows an excerpt from the Click event procedure for the button on frmBatchMail. The listing begins by reading the message file into a String variable (str2). Next, the code opens a record source for contacts. After these two preliminary steps, the code loops through all the rows in the contacts record source. The code within the Do loop begins by creating a NewMail class instance. Then, the code assigns the Subject and From properties of the class instance. The assignments to the BodyFormat and MailFormat properties are necessary for a message in HTML format. See the NewMail class online documentation at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cdo/html/_denali_newmail_object_cdonts_library_.asp for other possible settings of these two properties. Next, the code replaces the initial keyword or the last person's full name from the string with the message before assigning the message to the Body property of the NewMail object. Just before invoking the Send method to route the message to a recipient, the code assigns the E-mailAddress field from the query to the To property of the NewMail object. The loop closes by setting the NewMail object to Nothing and moving to the next row in the contacts query source:

```
str1 = txtPath & txtFilename
Set tst1 = fso1.OpenTextFile(str1, ForReading)
str2 = tst1.ReadAll

Set rst1 = New ADODB.Recordset
strSource = txtNamesQry
rst1.Open strSource, CurrentProject.Connection

strFind = txtFullName
Do Until rst1.EOF
    Set nmo1 = CreateObject("CDONTS.NewMail")
    nmo1.Subject = txtSubject
    nmo1.from = txtFrom
    nmo1.BodyFormat = 0
    nmo1.MailFormat = 0
    strReplace = rst1("Fname") & " " & rst1("Lname")
    str2 = Replace(str2, strFind, strReplace)
    strFind = strReplace
    nmo1.Body = str2
    nmo1.To = rst1("E-mailAddress")
    nmo1.Send
    Set nmo1 = Nothing
    rst1.MoveNext
Loop
```

All of my examples are from the area called CRM: Customer Relationship Management. You can position yourself to serve whole new areas of business needs that are both common and growing in importance—incorporating e-mail into your applications to manage business relationships. ▲

DOWNLOAD **409DOBSON.ZIP at www.pinnaclepublishing.com**

Rick Dobson is an author/trainer/Webmaster. His practice, CAB, Inc., sponsors national and Web-based seminar tours. His most recent book is *Programming Microsoft Office Access 2003*, and one of his most recent DVDs is "Programming Visual Basic .NET and ADO.NET with SQL Server and Access." Rick is an MCP for Visual Basic .NET. You can learn more about his practice, books, DVDs, and seminars at www.programmingmsaccess.com. rickd@cabinc.net.

Access Traps for the Naïve Developer

Garry Robinson

95 97 2000 2002 2003 DOWNLOAD

We all love Access, but our favorite tool has many "features" that lead the naïve developer into error. You may not appreciate the cost of these less-than-helpful additions but, should you upgrade to an enterprise database, you'll regret every one of them. Garry Robinson outlines those errors and how to avoid them (along with some code to find the errors).

RECENTLY I was asked to start preparing one of the Access databases that my company provides support for so that it was ready to upgrade to SQL Server 2000 or another enterprise database. The database was initially designed by a techno-savvy person, who, to his credit, came up with a database design that has stood the test of time and the critique of many of his peers. Unfortunately, Access can be a little too accommodating when an enthusiast designs a database, and this can allow design flaws to creep in—errors that a database professional may have been wise enough to avoid.

I'm going to discuss some of the subtleties that you'll need to address in your database tables in terms of upsizing your tables to an enterprise or open source database. It's better to make your database as perfect as you can before you try to convert your data. Once you're in an environment where you have Access as a front end and some other database as your back end, things get a lot more complicated. Or, if you're like me, improving your database model and reducing the size of your database is just a good thing to do.

Index gotchas

If Access is going to update a back-end database through ODBC, Access requires that a table must have at least one unique index. This means that just about every table in the database will need to have a primary key. A primary key isn't absolutely essential since any unique index on a table will do. In fact, tools like the SQL Server upsizer wizard will simply make the first unique index in the table the primary key.

Once you decide that you need to add a primary key to a table, you may not be allowed to add the key because you have duplicate values in the fields that you want to use in your primary key. There are two solutions. The first is to add one of those ugly AutoNumber fields to your table and make this your primary key. This is certainly quick and, if you resolve to review the key again later, you're really no worse off than you were before you started.

The better way to solve the duplicate items issues is to use the Find Duplicates query wizard (just click on the new query button in the database container to get to the wizard). This query will identify your duplicate values so that you can eliminate them. After having revised your index, don't forget to renew your database's relationship diagram if there are any other related tables.

If you don't add a primary index to a table straight away, Access goes out of its way to offer to add a unique index for you. Some naïve developers accept this offer and never give it another thought. Unfortunately, the default name for this primary key is "ID" and the naïve developer often accepts this. Once Access has played this card, a subsequent trap for the unwary occurs when the lookup table wizard is used. If the wizard is invoked, Access will often add an auto number field to a main table that matches the auto number field in the lookup table. This has the effect of storing a number in your table and a number in your lookup table. In addition, the name (probably "ID") is also duplicated into the main table, leading to more confusion. This repetition of names makes it difficult to figure out what tables are related to each other.

Another problem with these "ID" fields is that the field will generally be accompanied by a unique index that's also called ID. This unfortunate naming convention will cause problems with the transfer where it may be that indexes and fields might not be allowed to share names (or where "ID" is a reserved word). The solution is to search through and eliminate all ID field and relationship names and replace them with meaningful names.

Another Access "feature" whose results you can run into during conversions is the AutoIndex option. This little "nuisance" is located in the Table/Queries tab in Options (see **Figure 1**). Even though I try to clear this option as soon as I start working on a database, many developers are unaware of this option. The result is that there could be many tables in the database with indexes that weren't planned for. If you think that this is unlikely, try this little exercise:

1. Make sure that the option "Auto Index on Import/

Create" has the value "ID".
2. Open a new table in design mode and add a field with any name.
3. Add "ID" to the end of the field name.
4. Save the table.
5. When prompted, choose Yes to create a primary key.
6. Now open the table in design view and choose View | Indexes from the menu.

You'll now find that you have two indexes in the table: your primary key and the key automatically generated on the field ending with "ID".

Imagine that particular "feature" applied in a database with 100 tables (or more) and you'll start to see the challenges that can beset a database developed by an enthusiastic developer with the assistance of an enthusiastic Access wizard interface. While some of these indexes might actually speed data retrieval, keeping all of these indexes up-to-date is slowing down your database. And, when you upsize, they'll slow down the database server for everybody.

Another great gotcha is finding a relationship between two tables that have different sized fields. I don't seem to fall for this one very often, probably because, when I have a field in one table that I want to duplicate in another table, I copy and paste the common field. Maintaining these relationships is inefficient and, in SQL Server, forbidden. The error message that you'll get if you attempt to upsize a mismatched relationship to SQL Server looks like this:

```
[Microsoft][ODBC SQL Server Driver][SQL Server]
Column 'myTable.sampleNumber' is not the same length
as referencing column 'mySecondTable.SampleNumber'
in foreign key 'MyTable_FK00'.
```

To fix the relationship, head to Access' relationship window, right-click on the join between the two tables, and delete the relationship. Next, right-click on one of the tables and switch into design mode. Now change the field size to match the size in the other table, save the table, and *voilà!*—you'll be back in Access' relationship window. To complete the exercise, re-create the relationship between the tables.

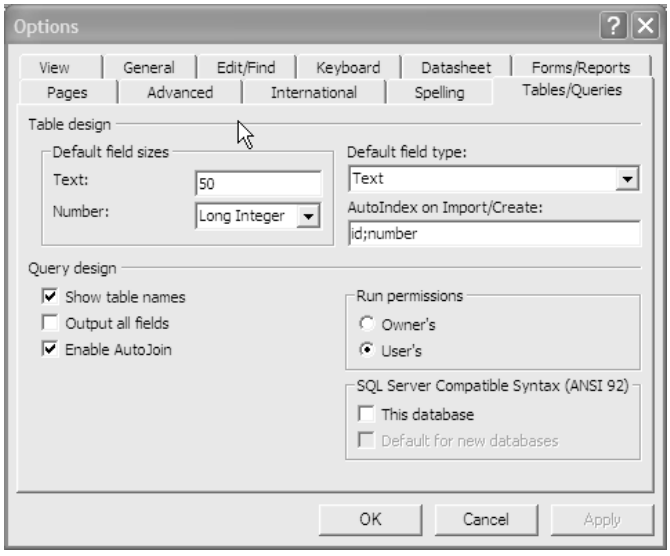

Figure 1. The AutoIndex on creation option.

Automated detection

At this stage, you may be wondering if I can show you some code that will identify these issues rather than making you wade through your databases to find these problems manually. I certainly can, and I'll start with some VBA that loops through all the tables. For each table, I call two functions of mine: one that checks for the existence of a primary key and another that verifies that the fields used in a relationship are the same size in both tables:

```
Dim i As Integer
Dim strTable As String
Dim varMsg As Variant

  For i = 1 To CurrentData.AllTables.Count
    strTable = CurrentData.AllTables(i - 1).Name
    If Left(strTable, 4) <> "msys" Then
      varMsg = checkPrimaryKey(strTable)
      If Not IsNull(varMsg) Then
        MsgBox varMsg & strTable
      End If

      varMsg = chkFKeyLength(strTable)
      If Not IsNull(varMsg) Then
        MsgBox varMsg & strTable
      End If

    End If
  Next i
```

Both of the functions that I wrote use the good old DAO library to retrieve information about the tables. I've recently become more upbeat about using DAO in my applications, as it's become obvious that ADO is never going to replace DAO for managing Access databases (this was confirmed for me when DAO reappeared in the Access Help files in Access 2003).

Here's the function that reviews all the indexes for every table to see if any of them have the Primary property value set to True. Passed a table name, the code retrieves the definition of the table from the TableDefs collection, and then loops through the table's Indexes collection looking for a key flagged as the Primary key:

```
Function checkPrimaryKey(tableReq As String) _
  As Variant
Dim dbData As DAO.Database
Dim tdf As DAO.TableDef
Dim idxLoop As DAO.Index

  checkPrimaryKey = Null
  Set dbData = CurrentDb
  dbData.TableDefs.Refresh

On Error Resume Next

tdf = dbData.TableDefs(tableReq)
For Each idxLoop In tdfLoop.Indexes
   If idxLoop.Primary = True Then
    GoTo checkPrimaryKey_exit
```

```
    Exit Function
   End If
Next idxLoop
checkPrimaryKey = "No Primary key for table "

checkPrimaryKey_exit:
   Set dbData = Nothing

End Function
```

My next piece of code is the function that checks the fields on both sides of a relationship to see if the field size is the same. It does this by working through the relationship objects in the database and verifying the field size on both sides of the relationship. If a discrepancy is found, the function returns a descriptive error:

```
Function chkFKeyLength(tableReq As String) _
 As Variant

Dim dbData As DAO.Database
Dim relLoop As DAO.Relation

Set dbData = CurrentDb

chkFKeyLength = Null

On Error Resume Next
dbData.Relations.Refresh

For Each relLoop In dbData.Relations
  With relLoop
    If tableReq = .Table Then
        If dbData.TableDefs(.Table) _
             (.Fields(0).Name).Size <> _
                dbData.TableDefs(.ForeignTable) _
                  (.Fields(0).ForeignName).Size Then
          chkFKeyLength = "Different foreign key " & _
                "lengths between " & .Table & _
                " and foreign table " & .ForeignTable
      End If
    End If
  End With
Next relLoop

dbData.Close
Set dbData = Nothing

End Function
```

With those examples, you can see how you might write code to test for upsizing issues that you commonly encounter in your databases. But wait! There's more that you should check for.

Table gotchas

No matter what Access will *let* you do, all of your tables should be named without any fancy characters or spaces between parts of the name. Moving tables with these kinds of names to any other database (including

Upsizing Issues

While my article focuses on poor practices (and highlights how those practices create problems when upsizing your database), in this sidebar I'm going to look at a variety of issues that occur *only* during upsizing.

Before you wade into a SQL Server conversion project, you really do need to sit down with a good book on the topic. In fact, you'll probably need to sit down with a *few* good books. As an introduction to upgrading, I like Russell Sinclair's book *From Access to SQL Server*. I also like the book *SQL: Access to Access SQL*, by Susan Harkins and Mike Reid, because it offers some good insights into setting up SQL Server. The book also offers lots of detail on Access and SQL Server query design, which is good reading whether you're converting or not. The most comprehensive and up-to-date book is *The Developer's Guide to SQL Server*, by Andy Baron and Mary Chipman. This book should probably be on your shelf. For those of you who have the Enterprise Edition of *Access Developer's Handbook*, don't forget to thumb through the book, as it has enough information to get you off and running through those troubling early stages of getting to know a new technology.

You should consider using a tool to help with your conversion. At the time that I was writing this article, I was putting together an Access wizard that will be available as part of the tools sold from my Web site. For a more comprehensive solution, you may want to try SSW Upsizing PRO!, which Adam Cogan's company sells at www.ssw.com.au. This will give you a very detailed list of all the issues that you will face. I recommend Adam's tool to anyone who's seriously considering a larger upsizing project.

Tools do present their own special problems. For instance, the SQL Server upsizing wizard can miss hidden tables when doing a conversion. Finding out that you missed a whole bunch of hidden tables late in a conversion project can be a little embarrassing. Unhide tables before running any automated tools.

Even if you use a tool, it would be surprising if your conversion went right the first time. Make a copy of your back-end database and run the conversion wizard to give you a detailed list of all the issues.

When your conversion does succeed, you're into a new world. One of the key components of making a conversion to an enterprise server work is to be sure that the correct skills are onsite for when the conversion succeeds. There's no doubt that Access databases are easier to manage than SQL Server (for instance, I can ask my clients to e-mail me a compressed copy of an Access database for me to make enhancements to). SQL Server databases require a more qualified technician with administration access to the server to assist in maintaining your new database server.

Microsoft has recently announced SQL Server Express, a "lite" version of SQL Server. I recommend this package for anyone contemplating this particular database engine.

Microsoft's own SQL Server) is going to make your conversion more difficult. Even in Access, dealing with table and field names with embedded spaces is awkward, requiring you to enclose the name in square brackets.

From time to time, all Access developers will have used a reserved word as a field name in a table or a query. Once again, Access isn't too harsh on the developer and will frequently forgive these errors. But, as I stated before, now is the time to sort out these anomalies before you upsize to a more restrictive database. You should avoid not only reserved words from the Access environment but also reserved words from the server environment. You even have to consider the reserved words used by the ODBC environment if you intend to use links to the server database tables. For more on Access reserved words, head to www.utteraccess.com/forums/access/access242075.html. For more on SQL Server and ODBC reserved words, see http://msdn.microsoft.com/library/en-us/tsqlref/ts_ra-rz_9oj7.asp?frame=true.

For one project that I worked on, there were more than 50 tables that suffered from issues such as reserved words or field names that didn't follow safer naming conventions. I considered using an Access renaming tool like Speed Ferret or FindAndReplace but, in the end, I took a simpler approach. I remembered that the name of the table in the server/back-end database must follow the correct naming convention. However, my method prevents the names from appearing to change in the front-end database so my code doesn't need to change:

1. Open the back-end database.
2. From the Tools | Options menu, make sure all of the Name Autocorrect options are turned off.
3. Rename the table from its current name to a (slightly) different name that conforms to your stricter naming conventions.
4. Fix up any issues with the field names in the renamed table.

Continues on page 19

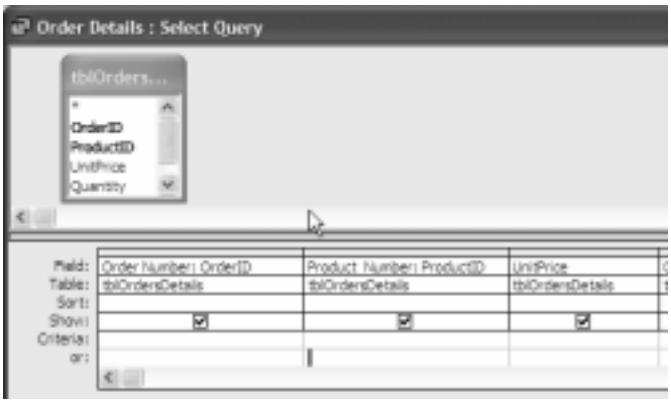

Figure 2. Setting up a query with field aliases to mimic the old table nomenclature.

Working All Day

Doug Steele

This month, Doug Steele 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 two days per week, it should be simple arithmetic, shouldn't it? You'd expect to use code like this:

```
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 return the number of weeks between two dates—sort of.

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 includes DateFrom in the count, 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 09 Sep, 2004 to 21 Sep, 2004. Ignoring DateFrom but including DateTo, there's 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 values for First Day of the Week. Those various numbers 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 values calculated by both DateDiff("d", DateFrom, DateTo, vbxxx) and DateDiff("w", DateFrom, DateTo, vbxxx) don't depend on the value of vbxxx.

Figure 1. 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 weekdays. As I mentioned before, whether or not the start or end date is on a weekend is important. With my calculation, if the end date is a Saturday the calculation will be too large by a day, unless the start date is also on a Saturday. If the end date isn't a Saturday but the start date is, the calculation will come up short by a day. All other combinations are fine.

Armed with this information, I can create the following function that calculates the weekdays correctly:

```
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
```

In this code 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 can 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 statement with the IIf function may be a little confusing, so I'll explain it in a bit more detail. The syntax for the IIf function 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 using the IIf function 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 isn't 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 SQL query. If you do that, you must use the numbers, as I did here, 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're going to use a function and there's a chance that some of the dates you pass to the function might be Null, then you should make the function's 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 (for example, January 1), some holidays fall on specific days (such as First Monday in September), and some have weird and wonderful formulae for their calculation (for instance, 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

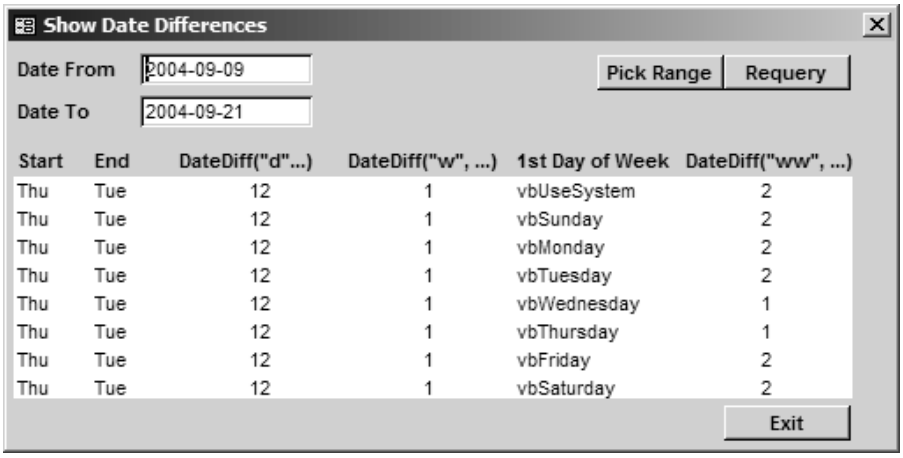

Figure 2. DateDiff calculations for Date Range 09 Sep, 2004 to 21 Sep, 2004.

(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 simple 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 with this code:

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

To do the same thing in SQL, you'd use this query:

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

Incorporating this into my WorkDayDiff function would look like this:

```
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, like this:

```
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 Holidays 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: only ignoring weekends. First, if the given start date falls on a weekend, then you want to start from the previous Friday. 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 from the date (otherwise, 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 two days every week, you can actually add a week to your start date for every five working days you're trying to add. Fortunately, 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 1, so the code actually looks like this:

```
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 that you're trying to add isn't an exact multiple of five, 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 one day and it won't be a weekend, but if you need to add two days, you'll end up with a Saturday, so you actually need to add four days to take you to Monday. Hopefully Table 1 illustrates the adjustment that's required.

In Table 1, the column Max Days represents the maximum numbers of days you can add to the day without hitting a weekend day. The column

Table 1. Adjustments required for remaining days.

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

Weekday(..., vbSaturday) shows the value returned by the Weekday function when you use Saturday as the FirstDayOfWeek. The formula 7 - Weekday(Date, vbSaturday) gives you the same value as Max Days. That means you need something like this code:

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

Recognizing that the remaining days can be calculated as NumberOfDays Mod 5, the 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 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 two or three 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, like this:

```
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 before. 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 of being understandable! Here's the code:

```
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
```

▲

DOWNLOAD 409STEELE.ZIP at www.pinnaclepublishing.com

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.Am/DougSteele for some Access information, as well as Access-related links. He enjoys hearing from readers who have ideas for future columns, though personal replies aren't guaranteed. AccessHelp@rogers.com.

User Configuration...

Continued from page 4

```
        controlColor = !Color
        controlStyle = !Style
    End With
    rs.Close

    Me.labelFont = makeFontName
    Me.bkgdColor.BackColor = controlColor

    Me.ctrlStyle.SpecialEffect = controlStyle
    If controlStyle = 0 Then
        Me.ctrlStyle.BorderStyle = 0
    End If
    Me.ctrlStyle = makeStyleName
End Sub
```

After the code that closes the recordset it simply sets the control properties as required. The only tricky bit here is the SpecialEffect property. A bug in Access sets the BorderStyle property to Heavy when you set this property to 0 (Flat). The If statement in my code handles this bug by resetting the property to 0 for a fine border.

Because the sample form is only using each property

in the table to set a single property of a single control, the sample procedure does this explicitly:

```
Me.bkgdColor.BackColor = controlColor
```

Alternatively, you could loop through the Controls collection of the Form to set the appropriate property for all controls of the correct type, but the basic procedure would remain the same. You can use a control's ControlType property to determine what kind of control you've retrieved from the Controls collection.

None of the procedures that I've shown you this month are particularly difficult technically—opening dialog boxes, loading and restoring values from a table, and setting control properties at runtime are all standard stuff. The real cost of providing this functionality is first in the analysis: deciding what characteristics ought to be configurable by the user and how they can best be stored. That cost is fairly minimal, so consider providing this functionality in your next system. It doesn't always make sense, but it's always worth considering. ▲

DOWNLOAD 409RIORDAN.ZIP at www.pinnaclepublishing.com

Rebecca M. Riordan is an author, systems architect, Microsoft MVP, and a darn good cook. Her current projects include an Access VBA, a book on designing user interfaces for database applications implemented with .NET, to be published by Addison-Wesley, and a book on low-carbohydrate baking that's currently looking for a publisher. rebeccamarye@msn.com.

Know a clever shortcut? Have an idea for an article for *Smart Access*?
Visit www.pinnaclepublishing.com and click on "Write For Us" to submit your ideas.

Access Traps...

Continued from page 14

5. Open the front-end database and delete the link to the old table.
6. Create a new link to the renamed table.
7. Create a new query that has exactly the same name as the original table.
8. Add the renamed table to the query.
9. Add all of the fields from the table to the query.
10. Where a field has been renamed, create an alias for the field that matches the old field name.

In **Figure 2** (on page 14), I demonstrate how I've set up a field alias for a couple of fields in a query so that the query now mimics the old naming conventions. I'm not

suggesting that you shouldn't fix these unfortunate names. But the good thing about my approach is that it quickly resolves the issues in your back-end database by isolating those issues in your front end. You can more easily fix and test these issues in your front-end database—and do it after the hurly-burly of the back-end conversion has been completed.

Access tries so hard to be helpful, it seems almost cruel to criticize the results. However, if you accept the results of the Access "helpers" without thought, you won't be following the "best practices" for a professional database design. While Access may let you get away with these problems, it's only a matter of time until these deficiencies rise up and bite you—and converting to an enterprise database is just one of those times. ▲

409ROBINSON.ZIP at www.pinnaclepublishing.com

Garry Robinson runs GR-FX Pty Limited, a company based in Sydney, Australia. If you want to keep up-to-date with his latest postings on Access issues, visit his company's popular Web site at www.vb123.com or sign up for his Access e-mail newsletter by sending a blank e-mail to tips@vb123.com. The Web site features many Access resources and software that are used by more than 10,000 readers per month. To find out about Garry's book, *Real World Microsoft Access Database Protection and Security*, point your browser to www.vb123.com/map. You can find Garry's contact details at www.gr-fx.com.

September 2004 Downloads

- 409RIORDAN.ZIP—The sample database that Rebecca Riordan has provided demonstrates how easy it is to set up your Access application so that your users can customize the user interface. Rebecca's example builds from the setting available in the Windows Control Panel to allow for additional customization.
- 409DOBSON.ZIP—Rick Dobson's sample database shows how to use Access as a center for mailing out notices to your customers and clients. Methods that you can use include integrating with Access or using the CDO objects that come with Office. Rick's code also demonstrates how to customize the content of your e-mail message using keywords.
- 409STEELE.ZIP—In this month's "Access Answers" column, Doug Steele demonstrates, first, how a simple date-related problem (workdays in a period) can have hidden traps—and then how a smart solution makes it simple again.

For access to current and archive content and source code, log in at www.pinnaclepublishing.com.

Editor: Peter Vogel (peter.vogel@phvis.com)
Contributing Editors: Mike Gunderloy, Danny J. Lesandrini, Garry Robinson, Russell Sinclair
CEO & Publisher: Mark Ragan
Group Publisher: Michael King
Executive Editor: Farion Grove

Questions?

Customer Service:
Phone: 800-493-4867 x.4209 or 312-960-4100
Fax: 312-960-4106
Email: PinPub@Ragan.com

Advertising: RogerS@Ragan.com

Editorial: FarionG@Ragan.com

Pinnacle Web Site: www.pinnaclepublishing.com

Subscription rates

United States: One year (12 issues): $149; two years (24 issues): $258
Other:* One year: $179; two years: $318

Single issue rate:
$20 ($25 outside United States)*

* Funds must be in U.S. currency.

Smart Access (ISSN 1066-7911)
is published monthly (12 times per year) by:

Pinnacle Publishing
A Division of Lawrence Ragan Communications, Inc.
316 N. Michigan Ave., Suite 300
Chicago, IL 60601

POSTMASTER: Send address changes to Lawrence Ragan Communications, Inc., 316 N. Michigan Ave., Suite 300, Chicago, IL 60601.