User Interface Standards—Reports

Dennis Schumaker

Many of what you think of as reports probably aren't reports to users—which makes life difficult for both of you. Dennis Schumaker shows how to create reports that make sense to your users, along with the code to implement those reports within a set of standards.

As Access programmers, we're all familiar with the need to create reports in our applications. However, to a user, the concept of a report is somewhat different. A user thinks in terms of an invoice, a purchase order, a check, or some other type of document. These items are all reports to a programmer, but, to the user, they're distinctly different things. Many programmers tend to think of all of these items as reports and, therefore, group these items together in an application in a reporting area—similar to what's shown in **Figure 1**. This isn't necessarily the best location for all reports within an application.

Several years ago, our firm implemented QuickBooks in our accounting department. Writing checks in QuickBooks wasn't the most intuitive process. In the accounts payable portion of the application, the user made checkmarks by individual line items to identify items to be paid. After that activity was completed, though, the user had to exit that portion of the application and enter the "reporting" area to print the checks. That wasn't exactly intuitive to me, but, to the programmer who wrote the code, a check *is* a report. As a user, I'd expected to be able to print the checks from the same location in the application where I'd identified items to be paid. In my mind, identifying items to pay and then paying those items are all part of the same business process. Why should I have to go to several different places in an application to get that one job done?

By critically looking at applications that we'd developed over the years, we recognized that there were three elements of reports that needed to be addressed as a part of our user interface design standards:

- *Location:* Where do you put the reporting functions within your application? The simple approach is to locate all the reporting in some type of report module, but that may not be the best option from a user's perspective.
- *Format:* What standards are you going to implement to manage the appearance of your reports? For instance, will all reports have "page X of XX" appear in the same location, a date printed in the same location, the titles in the same font, and so forth? How will you keep your documents' appearance consistent?
- *Selection:* How will the user select the specific information to be contained in the report? In most cases, the user wants a report to show information just for certain records, sorted in a specific way, to varying levels of detail, and so on.

At my company, we determined it was necessary to develop some standard user interface design concepts for addressing each of the issues identified here. The CD collection sample database that came with Access (which is included in the Download file available at www.smartaccessnewsletter.com) was reprogrammed to demonstrate the standards and act as our gold standard. This article will both discuss these standards and show the code that implemented them, using this simple database as an example.

You may disagree with the standards that we developed, and the only useful part of this article for you may be the code and programming techniques that I've included. However, the issues that we addressed are issues that your users will have also. Even if you don't intend to use our standards, you'll need to decide

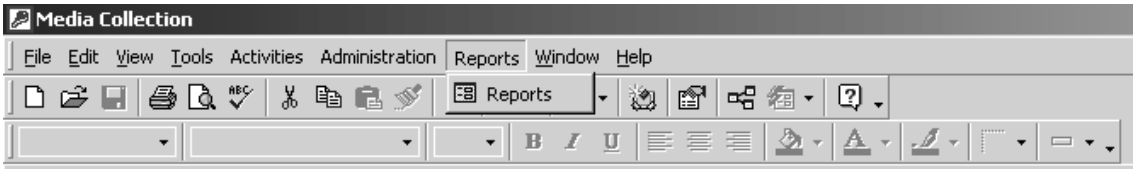

Figure 1. Reports bar.

how you intend to tackle these problems.

Location

Our overall design concept to address where reporting takes place is to put reporting within the area of the application where the user expects to use it. In many cases, reports will be located throughout the application and not just in some segregated reports area. As a result, most of our business process and application maintenance forms (as I discussed in previous articles) have reporting built into them. As an example, the main business process form for the CD collection database is shown in **Figure 2**.

The application maintenance form for the CD collection database is shown in **Figure 3**.

Each of these forms gives the user the ability to print relevant reports from the form without having to go to a separate area of the application—the dreaded "reports area." At almost any time, except in the middle of performing an operation, the user can press the Print button and be presented with a Report Preview of the information contained on the form.

In some cases, we've found that the user prefers to look up information in a report by being able to page through the report on screen rather than scroll down to the record on the form. Providing this printing feature has proven to be a feature that all users have begun to appreciate, particularly because it avoids doing a File |

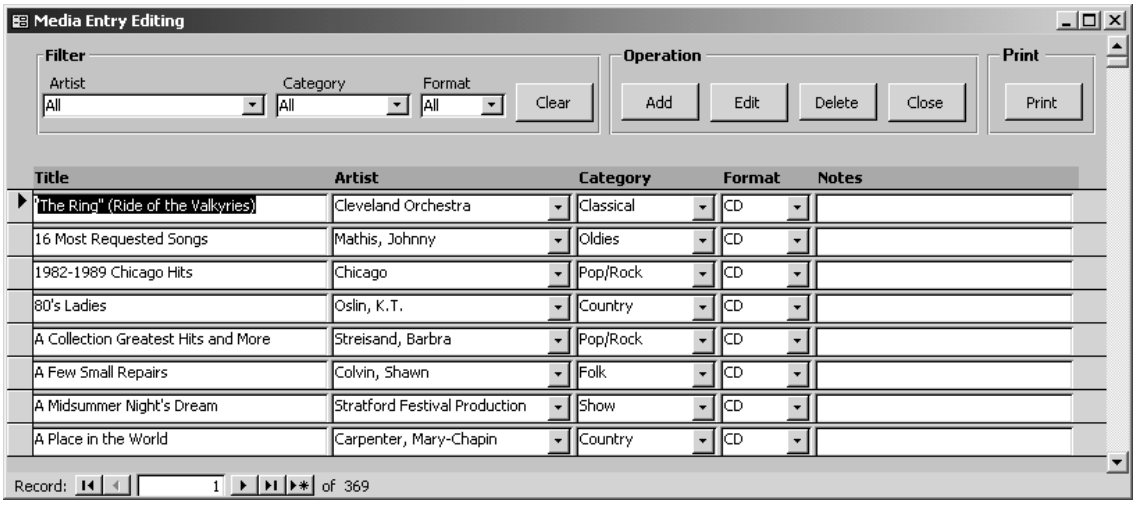

Figure 2. CD collection business process form.

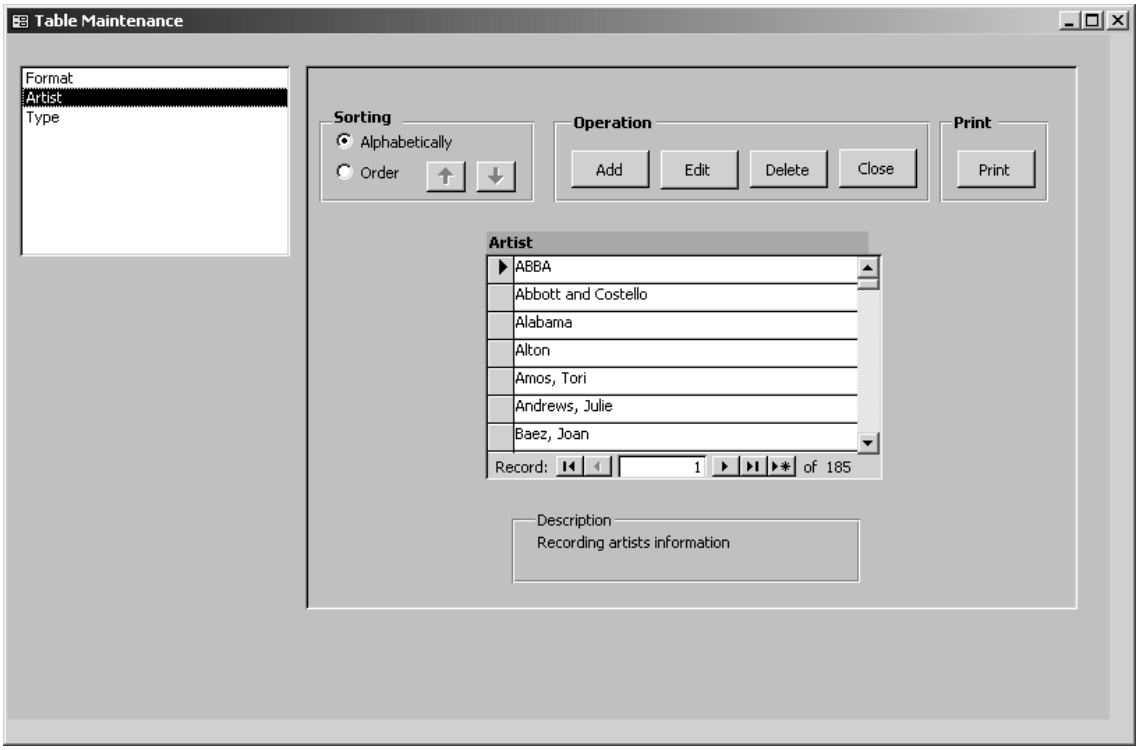

Figure 3. CD collection application maintenance form.

Print that prints the information in a form layout vs. the report layout that we provide. The Print button is also integrated with the form's filter settings; in the code behind the Print button, the form's filter settings are used to apply a filter to the record source of the report (this was discussed in my article on Form design in the November 2001 issue of *Smart Access*). Extending these concepts to other types of applications, you might want to enable your users to print an invoice from the same location in the application where they enter, edit, or update the invoice.

Some applications are developed primarily to maintain and report that information in a variety of ways. For those applications, locating the reporting in a separate reports area probably makes more sense than distributing the reports through the application. However, the principle of putting the report where the user expects it still applies: In a reporting application, the user is more likely to expect that there would be some type of reports area. In a reporting application, the user typically needs the ability to sort, select, or choose various reporting formats (different reports) for the information. Building the capability within one area of the application might not only be more cost-effective from a programming standpoint but would also make more sense from the user's perspective. Our approach to providing this reports area is discussed later in this article.

Format

It's difficult enough to make all your reports conform to format standards when you're the only programmer. However, standardization becomes even more difficult when multiple programmers are involved.

A standard report from the CD collection database is shown in **Figure 4**. Various areas of the report are identified with letters where specific standards have been developed that all programmers need to learn and implement when creating reports. The specific standards are briefly presented in **Table 1** (on page 14).

Logo

The client's, or customer's, logo is contained within a subreport that's located on each report. This logo is actually a compressed graphic in a subreport. That subreport is then placed in each individual report. If, instead of using the subreport, you decide to place the logo on each report, you may experience an attack of "database bloat." Graphics tend to take up a significant amount of space, so limiting the logo to one subreport that's contained in each report saves you disk space.

Fonts

Tahoma is the standard typeface for all of our reporting. Different font sizes and the use of bold, heavy, italic, and other features are based on the location of the information in the report. The choice of

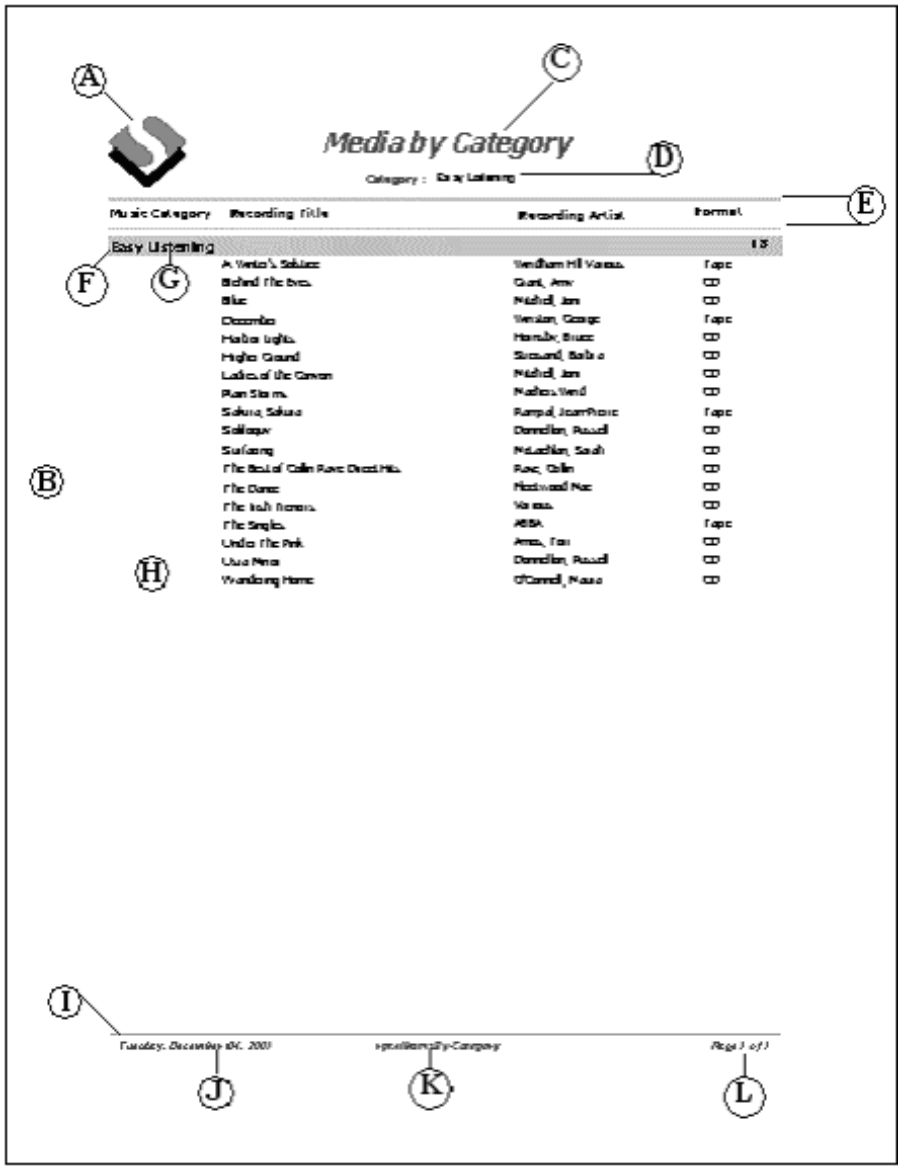

Figure 4. Standard portrait report.

the fonts isn't as important as being consistent.

The size of the font varies depending on:

- Whether the text is in a heading or a detail section.
- The amount of information that we're trying to show in the report.

Generally, font sizes of 8-point, 10-point, and 12-point are the sizes we use most. We typically use 10-point in detail sections (going down to 8-point if necessary to fit the information). Headings are usually in a 10-point to 12-point font.

We also use color. Headings are generally in some color other than black with the detailed information usually in black, as discussed later. You may be asking why we use color in our reports when 99 percent of the time they're printed on black and white printers. The reason is because if at some point we do start printing more of our documents in color, we won't have to re-create our reports to take advantage of color printing.

Report Title

Each report generally has a title that's centered on the report. It's in a larger font than the body text (usually 12-point through 24-point), in the color 16711680 (blue), with a font size of 18, font weight heavy, text alignment centered, forecolor 16711680 (blue), among other miscellaneous settings.

Criteria

Below the Report Title is the information in the report on the type of selection, filtering, and / or sorting criteria that was applied before any summarization was done. This section displays any reporting criteria that the user might have chosen. We used Tahoma, forecolor 16711680 (blue), and a smaller font size than the Report Title for this information.

Divider lines

Divider lines separate different sections of the report. These lines are used to separate information that's contained in what might be considered the footer of the report from the rest of the report. Likewise, divider lines are also used to separate the Report Title, Criteria, and Logo from the rest of the report. Other divider lines are used to separate the heading information. The page column headings are made to stand out by using both a top and bottom line.

Grouping sections

Grouping sections are usually formatted using either divider lines or shading. Group headers are formatted using shading. The text in a group header is usually slightly larger than the text in the detail section. The text in a group header is also formatted slightly larger than the detail section.

Table 1. Report standards.

Report	Report area	Description
A	Logo	The client's, or customer's, logo is contained within a subreport that's located on each report.
B	Fonts	Tahoma is the standard font used for all of our reporting. Different font size, bold, heavy, italic, and other features are used based on the location on the report.
C	Report Title	Tahoma font, with a font size of 18, font weight heavy, text alignment centered, forecolor 16711680 (blue) is used.
D	Criteria	This identifies any reporting criteria that the user might have chosen. It's Tahoma, forecolor 16711680 (blue), and a smaller font size than the Report Title.
E	Divider Lines	The page column headings are made to stand out using both a top and bottom line.
F	Group Header Format	Group headers are formatted using shading.
G	Group Header Text	The text in a group header is bolded.
H	Detail Section Format	The detail section is formatted in the smallest font size and no color.
I	Page Footer Line	The footer of the report is separated from the main parts of the report by a thin line. This line is border color 9868950 (light blue) and border width 1 point.
J	Print Date	This is the date the report was printed. This is also in a smaller Tahoma font in the same blue color as the rest of the footer.
K	Report Name	This is the name of the report in the Access database. A function is used to print this name.
L	Page Numbering	Functions are used to print the page numbering in the "page X of XX" format.

Detail section

The nature of the data being presented dictates how the information in the detail section is presented. The text in a detail section is usually the smallest font on the report (8-point or 10-point). We usually don't use any font colors in the detail section.

Report footer

The footer of the report is separated from the main parts of the report by a thin line. This line is border color 9868950 (light blue) with a border width of 1 point. The report footer contains the following elements:

- *Print Date*—The date the report was printed. This is also in a smaller Tahoma font in the same blue color.
- *Report Name*—This is the name of the report in the Access database. The built-in function NAME() is used to print this name.
- *Page Numbering*—Functions are used to print the page numbering in the "page X of XX" format.

Selection

Many of our reports are located within the application in an obvious or intuitive location. But, in many of our applications, the user also expects to find the report in some type of reporting area, which most of our applications contain. The reporting area is one location within the application from which several reports can be print-previewed and/or printed. There's at least one significant difference in functionality between a report in the application area and a report in the reporting area.

In most of our reporting from within a reporting area, we need to permit the user to customize the specific report. This is different from reports that are built into other areas of the application where customization is minimal. For instance, if you've created the ability to print a purchase order from a purchase order entry form, all that the user needs is the ability to print the purchase order that he or she is creating or editing. Normally, a user wouldn't expect to be able to customize the output to print all purchase orders between various dates, groups of vendors, and so on. However, working within a reporting area, a user would expect options like those to be offered.

Consequently, reporting contained within a reporting area usually contains the ability to choose additional features, known as report criteria. We implemented this capability with a form that supports the application of various sorting, selecting, and filtering capabilities based on the information contained within the selected report.

An example of the type of forms that we create to support this functionality is shown in **Figure 5**. This form uses a table in the database for populating the list box and determining which controls are to be turned on or off for the various reports. Using this design, it's easy to add more reports without having to add much code at all; instead, records in a table need to be added.

Report criteria

The Report form consists of three key areas. The list box on the left side of the form lists by name each report that's available. The report names are contained in the table in the database. This table is shown in **Figure 6**. The description field also comes from the working table.

Individual controls are turned on/off based on values contained in the table. An ADO recordset is created to look up the values in the table (twrkReport) and pass that information to code that displays the

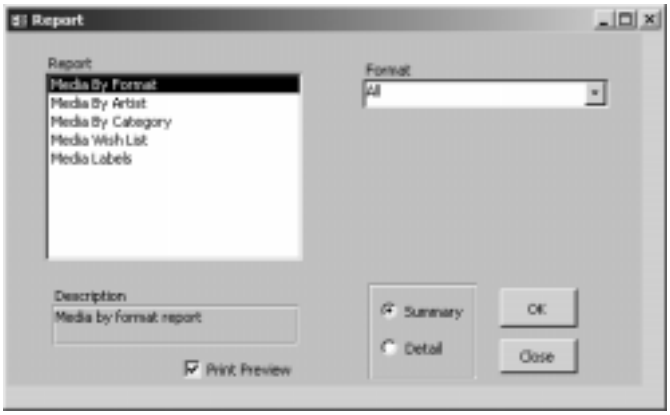

Figure 5. Report criteria form.

twrkReport : Table

ID	ReportName	ReportLabel	bArtist	bCategory	bFormat	bRecordingTitle	bSortBy	bSummary	bRequestNumberRange
1	rptAlbumsbyFormat	Media By Format	☐	☐	☑	☐	☐	☑	☐
2	rptAlbumsByArtist	Media By Artist	☑	☐	☐	☐	☐	☑	☐
3	rptAlbumsByCategory	Media By Category	☐	☑	☐	☐	☐	☑	☐
4	rptWishList	Media Wish List	☐	☐	☐	☐	☐	☐	☐
5	rptInfoLabel	Media Labels	☐	☐	☐	☐	☐	☐	☑
0			☐	☐	☐	☐	☐	☐	☐

Figure 6. Report working table.

proper controls. Here's the code for looking up the values in the table:

```
Private Sub lstReport_AfterUpdate()
    Dim rs As New ADODB.Recordset
    Dim bCategory As Boolean
    Dim bArtist As Boolean
    Dim bFormat As Boolean
    Dim bRecording As Boolean
    Dim bSummary As Boolean
    Dim bUpLabels As Boolean

   rs.Open "Select * From twrkReport " & _
               "Where ID = " & Me.lstReport, _
               CurrentProject.Connection, adOpenKeyset, _
               adLockOptimistic

    If Not rs.EOF Then
        mstrReportName = rs!ReportName
        bCategory = rs!bCategory
        bArtist = rs!bArtist
        bFormat = rs!bFormat
        bRecording = rs!bRecordingTitle
        bSummary = rs!bSummary
        bUpLabels = rs!bUpLabels
        Me!txtDesc = rs!Description
    End If

    Call TurnOnOff(bCategory, bArtist, bFormat, _
       bRecording, bSummary, bUpLabels)

End Sub
```

Once the values have been pulled from the table, the TurnOnOff function actually changes the controls. The individual controls are turned on and off based on the values contained within the table (twrkReport) for that individual report:

```
Private Sub TurnOnOff(bCategory As Boolean, _
        bArtist As Boolean, bFormat As Boolean, _
        bRecording As Boolean, bSummary As Boolean, _
        bUpLabels As Boolean)
With Me
    If bCategory Then
        !cboCategoryLU.Visible = True
        !cboCategoryLU = -1
    Else
        !cboCategoryLU.Visible = False
        !cboCategoryLU = -1
    End If
    If bArtist Then
        !cboArtistLU.Visible = True
        !cboArtistLU = -1
    Else
        !cboArtistLU.Visible = False
        !cboArtistLU = -1
    End If
    If bFormat Then
        !cboFormatLU.Visible = True
        !cboFormatLU = -1
    Else
        !cboFormatLU.Visible = False
        !cboFormatLU = -1
    End If
    If bSummary Then
        !fraReport.Visible = True
    Else
        !fraReport.Visible = False
    End If
    If bUpLabels Then
        txtFirstNumber = _
               NToZ(DMin("RecordingID", "tblRecordings"))
        txtLastNumber = _
               NToZ(DMax("RecordingID", "tblRecordings"))
        !lblReqNumRange.Visible = True
        !lblFirstNumber.Visible = True
        !txtFirstNumber.Visible = True
        !lblLastNumber.Visible = True
        !txtLastNumber.Visible = True
        !lblThreeUp.Visible = True
        !chkThreeUp.Visible = True
        !chkThreeUp = False
    Else
        !lblReqNumRange.Visible = False
        !lblFirstNumber.Visible = False
        !txtFirstNumber.Visible = False
        !lblLastNumber.Visible = False
        !txtLastNumber.Visible = False
        !lblThreeUp.Visible = False
        !chkThreeUp.Visible = False
        !chkThreeUp = False
    End If
End With
End Sub
```

The code behind the OK button then uses the value(s) in the combo or text boxes on the form to apply a filter to the report. The form contains two different types of reports:

- Some reports use a combination of values to filter the report.
- The second report (which prints labels) actually filters the information by the RecordingID.

The code first questions which type of report is being requested and runs the appropriate ReportFilter code based on the selections for filter criteria that have been entered into the combo or text boxes on the form. The code for printing labels looks like this:

```
Private Sub CmdOK_Click()
   If mstrNameOfReport = "r pointInfoLabel" Or _
        mstrNameOfReport = "r pointInfoLabel3Up" Then
       If IsNull(Me!txtFirstNumber) Then
           MsgBox "Please enter a First Number", , _
                       "No First Number entered!"
           Exit Sub
       ElseIf IsNull(Me!txtLastNumber) Then
           MsgBox "Please enter a Last Number", , _
                       "No Last Number entered!"
           Exit Sub
       End If
       If Me.chkPreview = True Then
         DoCmd.OpenReport mstrNameOfReport, _
               acViewPreview, , _
               "tblRecordings.RecordingID " & _
               " Between " & _
               NToZ(Me!txtFirstNumber) & _
               " And " & NToZ(Me!txtLastNumber)
       Else
         DoCmd.OpenReport mstrNameOfReport, _
               acViewNormal, , _
               "tblRecordings.RecordingID " & _
               " Between " & _
               NToZ(Me!txtFirstNumber) & _
               " And " & NToZ(Me!txtLastNumber)
       End If
   Else
       ReportFilter
       If Me.chkPreview = True Then
         DoCmd.OpenReport mstrNameOfReport, _
               acViewPreview, , mstrReportFilter
       Else
         DoCmd.OpenReport mstrNameOfReport, _
                acViewNormal, , mstrReportFilter
       End If
   End If

End Sub
```

For other non-label reports, the ReportFilter code

that's used to build the Where clause on the report record source is shown here:

```
Private Sub ReportFilter()
   mstrReportFilter = ""
With Me
   'Format
   If (!cboFormatLU <> -1) Then
      If mstrReportFilter = "" Then
         mstrReportFilter = _
               "[FormatID] = " & !cboFormatLU
      Else
         mstrReportFilter = mstrReportFilter & _
               " And [FormatID] = " & !cboFormatLU
      End If
   End If
   'Music Category
   If (!cboCategoryLU <> -1) Then
      If mstrReportFilter = "" Then
         mstrReportFilter = "[MusicCategoryID] = " & _
                !cboCategoryLU
      Else
         mstrReportFilter = mstrReportFilter & _
               " And [MusicCategoryID] = " & _
               !cboCategoryLU
      End If
   End If
   'Artist
   If (!cboArtistLU <> -1) Then
      If mstrReportFilter = "" Then
         mstrReportFilter = "[RecordingArtistID] = " _
                     & !cboArtistLU
      Else
         mstrReportFilter = mstrReportFilter & _
               " And [RecordingArtistID] = " & _
               !cboArtistLU
      End If
   End If
End With
End Sub
```

The concepts involved in building this report area could be extended even further. If you didn't want to manage individual controls in the form, you could build separate subforms that could be changed out using the same type of working table.

Print Preview

We always build in a Print Preview to view a report before the user sends it to the printer. However, we recognized that in some situations, a user may normally want to send the report directly to the printer and only occasionally preview it. To support that, we provide an optional Print Preview with a "Print Preview" check box. The setting for the Print Preview check box is stored in a table in the database so that once the check box is set, the next time the user opens the reporting form, the Print Preview setting is at the same setting as the last time it was used:

```
Private Sub chkPreview_AfterUpdate()
   Dim rst As New ADODB.Recordset
   rst.Open "tblSystemInfo", CurrentProject.Connection, _
       adOpenKeyset, adLockO pointimistic
   rst!Preview = Me.chkPreview
   rst.Update
   rst.Close
End Sub
```

There's probably more that could be said on reports, and user interface design for that matter; however, the concepts described here have worked well for us. Our applications have a consistent appearance to the user that's independent of the exact business function that the application supports. In addition, it's much easier to have our programmers get started building an application because user interface design concepts don't have to be re-created with each application. And last (but most importantly), when you do things the same way over and over again, you get better at it, you learn to improve on it, and you can make more money from it.

All of the applications that we've developed for clients have had relatively small budgets. We have to be able to quickly get to the point where the client can see some results. This lets the client begin to realize that the application will be able to improve their operations or meet their specific business objectives. The sooner that we're able to show them some results, the more likely they'll be to spend more money to achieve as much benefit as possible from the application in their business. ▲

DOWNLOAD **REPORTUI.ZIP at www.smartaccessnewsletter.com**

Dennis Schumaker has more than 27 years of business and consulting experience in both the public and private sectors. Dennis holds a bachelor's degree in mechanical engineering and a master's in nuclear engineering from Ohio State University, and an MBA from the University of Michigan. He's also a Microsoft Certified Professional plus Internet, a Microsoft Certified Systems Engineer (MCSE), and a Microsoft Certified Trainer (MCT). Dennis is an officer with Schumaker & Company, a management consulting and professional services firm and a Microsoft Certified Partner headquartered in Ann Arbor, MI. Schumaker & Company provides professional advisory services and project management services to both public and private sector clients. dschumaker@schuco.com.