

Smart Access

Solutions for Microsoft® Access® Developers

Temporary Tables with No Bloat

Doug Den Hoed



Temporary tables are great for extending query functionality, storing transient data, and solving concurrency issues. Unfortunately, using a linked Jet table to store that data will cause database bloat. This month, Doug Den Hoed shares his technique for creating bloatless CSV temp tables on the fly.

SOMETIMES you need to hold some data in a table to work with it over the course of a transaction, but once the transaction is complete the table can be discarded. This is the essence of temporary tables. I've taken advantage of temporary tables in many of my applications. Unlike SQL Server and Oracle, however, the Jet engine doesn't directly support the concept of temporary tables. To make things worse, even using a standard table as a holding area and deleting the records after processing doesn't work very well in Access. Jet doesn't recover the space from deleted records until the MDB is compacted, a behavior Access developers call "database bloat." A Jet application that uses anything like temporary tables will just grow and grow until it consumes the user's hard disk.

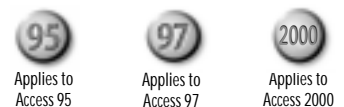
I tried a number of ways to use temporary tables efficiently in Access. At one point I was writing code to open a new MDB, create a table in it, link the tables back to my original MDB, insert the temporary data, use the record, and then delete the MDB. No luck: Bloat occurred anyway in my source MDB. I was also running into concurrency problems. If two users were running a procedure that created a temporary table, I needed some way to keep their data separate. "Temporarily" defeated, I explained to my users that they'd need to compact their databases often, and I even created compaction routines for my users to run. Then I found a solution.

Continues on page 5

January 2001

Volume 9, Number 1

- 1** Temporary Tables with No Bloat
Doug Den Hoed
- 4** Editorial: Access Applications
Peter Vogel
- 10** User-Driven Menus
Keith Bombard
- 14** Everything Doesn't Happen at Once: Loosely Coupled Events
Peter Vogel
- 17** Access Answers: Data Projects and SQL Server
Andy Baron
- 22** January 2001 Source Code



www.vb123.com/smart

In code, an underscore (_) as the last character of a line indicates that the line has been wrapped for layout purposes. In Access 95 and up you can use the code as it appears, but in Access 2.0 you must recombine the wrapped lines.

Access Applications

Peter Vogel

I keep hearing that developers aren't using Access to create commercial applications anymore. That might be true, but you wouldn't know it by me. One thing that I keep being asked to do is review Access applications that developers are preparing to market commercially. I've seen three in the past three months, and they've all been very interesting, ranging from sophisticated service billing systems to specialized legal office management. There are lots of benefits to this side of my business: I get to meet some interesting people, I get to talk about Access with people who care about it, and I probably learn as much as I help. There are benefits to readers like you, too, as these developers can often be convinced to write up some of the techniques that they use for *Smart Access*.

It's not unusual for articles in *Smart Access* to reflect the work of developers creating and supporting commercial applications. Hardin Brothers' article on managing broken references in our July 1999 issue is one example, as were Stuart Kinnear's articles on installing Access applications (in the February, March, and April 2000 issues). Lately, Doug Den Hoed has been sharing the techniques that his development team came up with as part of creating their knowledge base product.

You can check out their product (called The KB™) at www.thekb.com (you can also download a 15-day free trial version). I think that it's a very interesting tool with one major drawback: It's hard to describe. The simplest description is that The KB lets you track information about your development project. The most complicated description is that The KB lets you build a knowledge base about your project.

The goal of The KB is to gather together in one flexible storage mechanism all of the information that you need to manage a development project and allow you to access it dynamically. As the project changes, The KB's users can quickly update their information to provide a dynamic and accurate picture of where the project is and where the project is going. That description makes The KB sound like a project management system, but it aspires to be more than that.

In addition to tracking the progress of your project, The KB also provides a place to record design decisions, who made the decisions, the process that led to a particular design choice, and much else about the project. A team that uses The KB should be in a better position to

visualize their project. Described that way, it's hard to imagine any large project doing without those services. With the information gathered together in one flexible location, new team members can get up to speed faster, existing team members have a common reference source, and future teams can learn from the project's successes and failures.

Doug and his team have explored a lot of innovative techniques in building their application. Doug's article in last month's issue on using GIFs to manage the overhead of using graphs is one example. His contribution this month on using comma separated value (.CSV) files is equally interesting (I'd never have thought that there was a use for comma-delimited files before reading Doug's article). Next month, he'll return to show how to create Gantt charts from Access data.

The coding techniques that The KB pushed Doug to develop are neat. What really interests me is the breadth of data that The KB must try to manage. There's a tremendous amount of disparate information that must be stored. To put it another way: It sounds like the perfect application to be developed in Access. To my mind, Access remains the preeminent rapid application development tool for creating the front end for databases. In the past, Access has really only worked well with the Jet database (so much so that "Jet" and "Access" are almost synonymous). With ADO and the changes that have begun in Access 2000, we might see Access become the best tool for creating a front end for any database. ▲

FREE	SQL Server	Visual Basic	Web Development	Access
	Java	Visual C++	Delphi	Oracle
	XML	Linux	FoxPro	IS Consultant

FREEeNewsletters.com Pinnacle Publishing®

XML • Web Development • SQL Server • Visual Basic • MS Access • Oracle • Visual C++ • Delphi
• FoxPro • XML • Web Development • SQL Server • Visual Basic • MS Access • Oracle

Sign up now for Pinnacle's FREE eNewsletters!

Get tips, tutorials, and news from gurus in the field delivered straight to your Inbox.

http://www.FREEeNewsletters.com

XML • Web Development • SQL Server • Visual Basic • MS Access • Oracle • Visual C++ • Delphi
• FoxPro • XML • Web Development • SQL Server • Visual Basic • MS Access • Oracle

Temporary Tables...

Continued from page 1

Needing temporary help

I was compelled to find a better way while preparing to release a commercial application called The KB™. The KB is knowledge base (“KB”) software designed to manage team-oriented, task-driven projects—projects like building Access applications, for instance.

Certain charts in The KB let users dynamically sort a datasheet’s column headers (see my article in the May 2000 issue of *Smart Access*, “Taking Back the Power with the Access Runtime”) and then use that data to construct a Gantt chart in MS Graph (I’ll cover this in an upcoming article). The process makes extensive use of temporary tables, and I worried about the bloat from running those charts. Since we were creating a commercial product, I wondered how I could impose my “compact often” workaround on an anonymous, international audience. So, instead of storing my transient chart data in Jet, I devised a technique to write my temporary data to a local comma separated value (.CSV) file, link the CSV file, and then use the linked CSV in my queries like any other table. This technique turns out to be fast, causes no bloat, and solves concurrency issues since users each have their own local datasource for their charts. Do note that I use these tables only for reading data and don’t try to update them (which is generally all that’s required of a temporary table).

I ran some tests to benchmark these different bloat scenarios and summarized the results in [Figure 1](#). You can generate your own benchmarks using the sample database that’s available in this month’s Source Code file at www.smartaccessnewsletter.com. It contains a routine called `tmpCreateFromListbox` that can be added to your applications to send criteria from a multi-select list box to a CSV file and automatically link the file back to your database to be used in SQL queries. I’ll walk through that function to show off the technology, and then I’ll give you some examples of ways I’ve applied the TempCSV technique.

The sample database’s “skills application” illustrates how my function could be used. The sample is based on a real application that I built for one of my clients. The

original solution used a lot of code but didn’t require any temporary tables. By applying the techniques that I’ll show you here, I rewrote the solution to use SQL and considerably less code.

The problem that the sample addresses is to find a consultant who meets a specific set of criteria. The criteria can be divided into three categories (industry, skills, and language), and the user is allowed to specify multiple values in each category. In [Figure 2](#) (on page 6), you can see the main form from the sample database at the top of the figure and the query that retrieves the results at the bottom. In the form’s top three list boxes, you can multi-select combinations of Industry (in this case, Oil & Gas), Skill (here, Access, ADO, DAO, SQL Server, and Visual Basic), and Language (in this case, <Any>) values. Each list box has two columns: the ID (hidden) and the value (shown). So, in my example, `ttmpIndustry.CSV` has 1 for Oil & Gas; `ttmpSkill.CSV` has 1, 4, 5, 6, and 2 for my five skill choices; and `ttmpLanguage.CSV` has 6, 5, 1, 2, 4, and 3, since I chose <Any>. Clicking one of the buttons on the right calls my `tmpCreateFromListbox` function, which creates temp CSV tables that contain just the IDs for the selected items.

The query (`qselQualifiedConsultant`) at the bottom of [Figure 2](#) uses the CSV tables as part of a SQL statement that retrieves all of the employees who meet the criteria selected in the list box (thanks to Peter Vogel for helping me work out the SQL for this). The CSV tables restrict the data to show only the consultants who match the values selected from the lists. The SQL statement joins each CSV

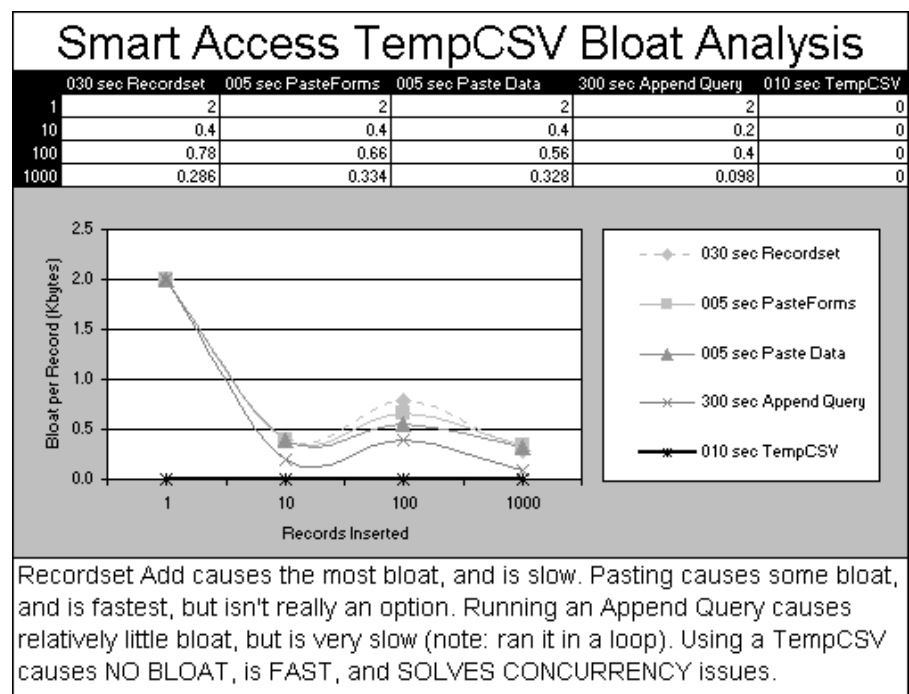


Figure 1. Bloat per row benchmarks.

table to the corresponding Xref table, which list the industries, skills, and languages for each consultant. Those tables are, in turn, linked to the list of consultants, which limits the rows to only those consultants who meet the criteria in all three tables. The Group By converts the records from the joins to a single row, which I then display in a message box, on a report, on the form, or on a subform, depending on which button you click to run the application. The sidebar “No More Compacting?” has a further explanation of the buttons on the form.

Linking tables

Unlike a table, a comma-delimited file doesn't contain a description of its structure. To link a CSV, you need to create a Link Specification so that Access knows how to parse the information stored in the CSV. The simplest way to build that specification is to link a text file to your database through the Access user interface. Since my tables were going to be generated as my application ran, I had to create a dummy file to link to Access in order to create the specification. My three CSV files could use the same specification since they had identical layouts: one field (datatype Long) per record. To create the dummy file, I opened a new file in Notepad, typed “1”, hit Enter, typed “2”, hit Enter, typed “3”, and then saved the file as ttmpLongID.txt. I now had a text file of three records, each record having one field, and Long data in each field.

In Access, I followed these steps to link in my dummy table and create my Link Specification:

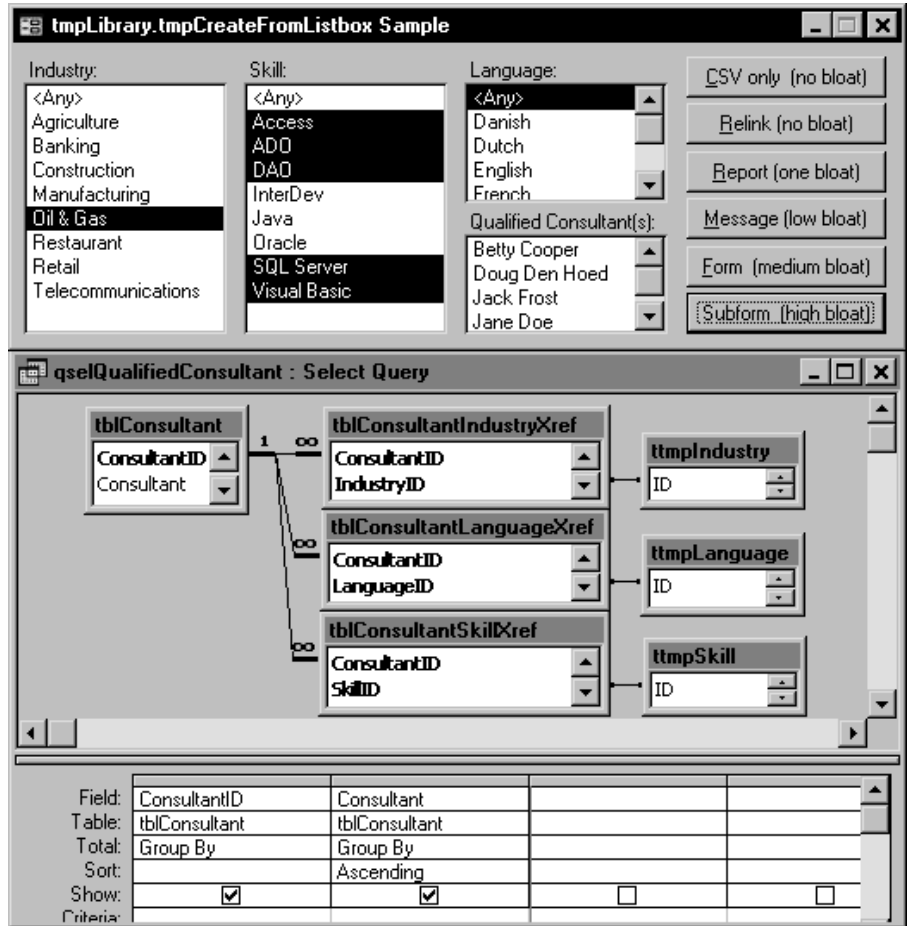


Figure 2. tmpCreateFromListbox example.

1. From the File menu, I selected Get External Data | Link Tables.
2. In the Link dialog box that appeared, I changed Files of Type to Text Files, selected the file ttmpLongID.txt, and clicked on the Link button.
3. In the first form of the Linked Text Wizard that appears next, I chose Delimited and clicked on the Next button.
4. The next form in the Wizard lets you select the

No More Compacting?

My sample database includes some extra buttons that demonstrate something I discovered while writing this article: Some very common Access actions cause database bloat. I found that, in both MDB and MDE formats, creating, loading, and linking CSV files causes no bloat. Running a report that refers to CSV tables sometimes adds about 8K across multiple runs. Walking a recordset to format a

MsgBox causes less, about 4K every time, on the OpenRecordset. Opening a bound form causes some bloat every time it opens (again, 8K). Changing the RowSource on a subform causes the most bloat every time (12K). Use my sample form to test the effect on your database, as you might get different results on your machine, but I suspect the general behavior is unavoidable.

character that you'll use to delimit the fields in your record. I selected Comma as delimiter and clicked on the Next button.

- The form that appears next in the Wizard allows you to assign names and data types to the fields in the text file (the Wizard scans my dummy text file to determine how many fields are in it). I named the field "ID," set its data type to Long Integer, and clicked on the Next button.
- The last form in the Wizard lets you assign a name to the table. I named my table Linked Table TtmpLongID and then clicked on the Finish button.

The Wizard has inserted a row in MSysIMXSpecs that describes my Link Specification, which I can now use to link to other tables with the same layout. The entry in MSysObjects for the link to my dummy table uses that specification in its connection string as "DSN=TtmpLongID Link Specification," for instance.

My tmpCreateFromListbox function accepts four parameters: the name of the table to link, the specification to use, the list box to draw the data from, and a force link flag. To save processing time, I normally only link the table if it's not already linked (for instance, the first time on a new computer). However, passing a True in the force link parameter to the call will cause the routine to drop any existing link to the table requested and create a new one. A call to link a file called ttmpIndustry using my predefined specification and load the table with data from the list box lstIndustry (without forcing a relink) would look like this:

```
Call tmpCreateFromListbox("ttmpIndustry", _
    "TtmpLongID Link Specification", _
    Me.lstIndustry, _
    False)
```

I use ttmp as a prefix for all of my temporary tables. I also give my tables the same name as the file that they're linked to. So, in the preceding example, I'm linking the file ttmpIndustry.CSV as the table ttmpIndustry.

Following is the start of the tmpCreateFromListbox function. The first thing that it does is check to make sure that at least one item in the list box is selected and exit if not:

```
Public Function tmpCreateFromListbox( _
    ByVal strTempTable As String, _
    ByVal strLinkSpec As String, _
    ByRef lstSource As ListBox, _
    Optional ByVal fForceRelink As Boolean = False _
    ) As Boolean

If lstSource.ItemsSelected.Count <= 0 Then
    GoTo Exit_tmpCreateFromListbox
End If
```

Providing there's at least one row selected in lstSource, tmpCreateFromListbox continues, initializing

the fFirstRow flag and strCSVFile location. The code also deletes any existing versions of the strCSVFile and then opens a new file for Output:

```
fFirstRow = True
strCSVFile = tmpApplicationPath & "\" _
    & strTempTable & ".CSV"
On Error Resume Next
Kill strCSVFile
On Error GoTo Err_tmpCreateFromListbox
lngFileNumber = FreeFile
Open strCSVFile For Output As #lngFileNumber
```

Next, the function determines whether the list box is disabled and checks for a special "*" value in the selected row. Either condition means that every value in the list box should be inserted to the TempCSV, a decision that's stored in the fDoAll variable:

```
fDoAll = False
If Not lstSource.Enabled Then
    fDoAll = True
Else
    For Each varItem _
        In lstSource.ItemsSelected
        If lstSource.ItemData(varItem) = "*" Then
            fDoAll = True
            Exit For
        End If
    Next varItem
End If
```

Next, the code writes the selected items in the list box to the TempCSVfile or, if fDoAll is set to True, every item:

```
If fDoAll Then
    varItem = 0
    Do Until varItem >= lstSource.ListCount
        If lstSource.ItemData(varItem) <> "*" Then
            Write #lngFileNumber, _
                CLng(lstSource.ItemData(varItem))
        End If
        varItem = varItem + 1
    Loop
Else
    For Each varItem In lstSource.ItemsSelected
        Write #lngFileNumber, _
            CLng(lstSource.ItemData(varItem))
    Next varItem
End If
```

With the file now loaded with the data, the function issues a DCount against the TempCSV to check the link. If the DCount fails, or the routine has been passed the force link parameter, the code deletes the current link and relinks using the name of the Link Specification passed to it:

```
On Error Resume Next
Close #lngFileNumber
Debug.Print DCount("...", strTempTable, "1=2")
If Err <> 0 Or _
    fForceRelink Then
    On Error GoTo Err_tmpCreateFromListbox
    On Error Resume Next
    DoCmd.DeleteObject acTable, strTempTable
    On Error GoTo Err_tmpCreateFromListbox
    DoCmd.TransferText acLinkDelim, _
```

```

strLinkSpec, strTempTable, _
strCSVFile, False
Else
  On Error GoTo Err_tmpCreateFromListbox
End If

```

On exit, the function will close the CSV file, no matter what else happened in the routine. Without that precaution, subsequent attempts to work with the file will fail. Similarly, any objects that refer to the TempCSV must also be closed before you can refresh them, or you'll get "Err 70: Permission Denied" when you try to open the file for Output. Once all of the temporary tables are linked, I run the query qselQualifiedConsultants that generates the list of consultants.

When you use tmpCreateFromListbox in your applications, remember that you can store any data you wish in the CSV file, not just Long values as I chose to do. Just remember to create the Link Specification first. You can choose a different delimiter, which is essential if the data you intend to write to the file has commas in it.

Extending query functionality

So how can you use these temporary tables? For my

first example, I'll take a user who enters some project information into The KB and then generates a chart to review the impact of those changes. In **Figure 3**, you can see our KB product in its docking station mode. In the top right subform, two Treeview controls allow the users to show and hide the data they're interested in (see my October 2000 article, "Taming the Treeview Control") to set the overall context for the data they'll work with. The user then refines the search in the top left subform by choosing The KB project, which in turn drives its parent, Software Development, into the template field. In this example, Doug Den Hoed has been chosen in the solver field.

When the user clicks the filter button, The KB will search for any "KB Items" that match the filter (in this case, the result corresponds to the question, "What is assigned to Doug Den Hoed in The KB project that isn't closed?"). The results are displayed in the bottom left pane. The user can scroll right to confirm that each KB Item has a budget, revised budget, and estimate to complete hours. Double-clicking the row for KB Item 923 pops up its detailed form. The user enters some actual

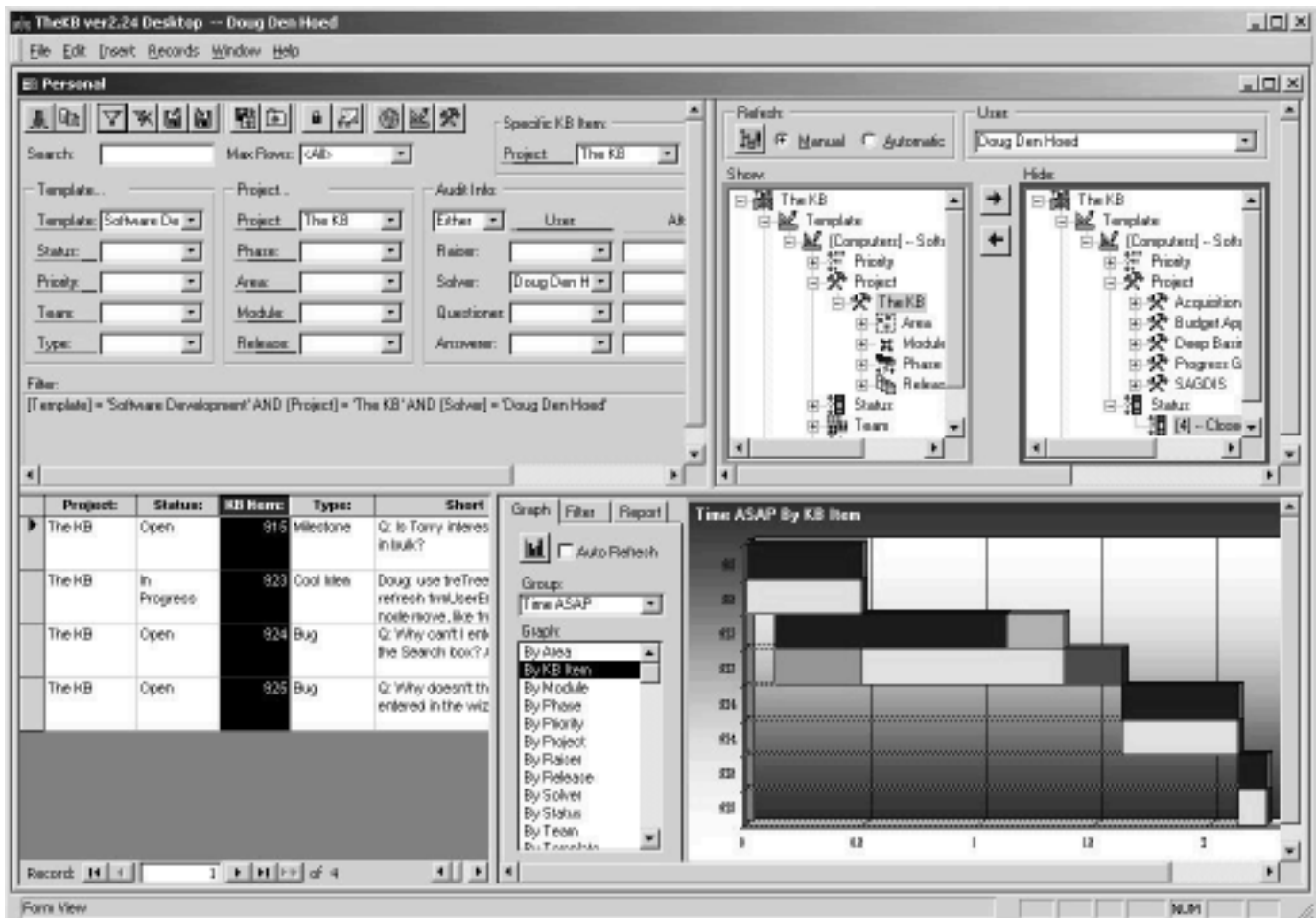


Figure 3. The KB in docking station mode.

time against the project on that form before closing the form. Double-clicking the KB Item column will sort the tasks from oldest to newest.

In the bottom right subform, the user selects the Time ASAP group to indicate the kind of chart that he or she wants to generate. I coined the “ASAP” term to represent a class of Gantt-like charts that display a critical path of remaining work, summarizing the earliest time to finish. I use ASAP charts to answer questions from my clients such as, “Doug, if you drop everything but X, could you finish X by next Friday?” Within the ASAP group, the user chooses to display the By KB Item chart and then clicks the chart Refresh button.

The chart shows a lot of information, distinguished by color. For each KB Item, there are two sets of bars. The upper bars show the original (blue) and revised (cyan) budgets, in hours. The lower bars show the actual to date and under budget (green), the estimate to complete and under budget (yellow), and the portion over budget (red). The ASAP chart lines up only the work yet to be done to drive its critical path. Item 915 will take half a day, and then it lines up the remaining portion of item 923. Then, when 923 is finished (over budget, incidentally), items 924 and 925 are shown, projecting an overall 2.25-day duration.

This dynamic chart generation is made possible by temporary tables. I use CSV files to hold the data that drives the ASAP charts. After the users have made their selections, all of the KB Items are present in the datasheet, filtered, sorted in the right order, with their budget, estimate to complete, and actual amounts. So, The KB code just walks the datasheet’s recordset to load the CSV file with entries for each KB Item, including some extra columns to drive the bars on the ASAP chart. The ASAP chart then joins the information in the temporary tables to the real tables, grouping to roll it up when requested, and displays the chart.

Storing transient data

Another way I’ve used TempCSVs is to simulate a row-level view of data. My team developed this technique during our first replication project back in Access 95 BPR (Before Partial Replication), when we had to send all data everywhere. We used a temporary Jet table to hold data that controlled what the current user was permitted to see, and then we joined that control table to every form, report, and recordset in the application. The join restricted the data to only the rows that pertained to the current

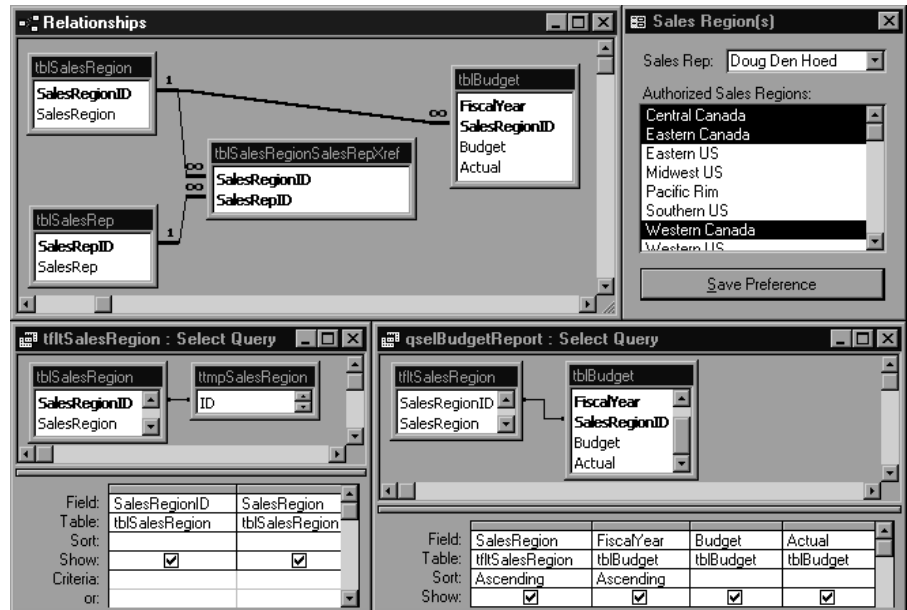


Figure 4. Horizontal segmentation of data.

user, an effect we called “horizontal segmentation.”

In Figure 4, you can see an extension of our original concept. In the top left corner, the data model shows the many-to-many relationship between Sales Regions and

Continues on page 21

User-Driven Menus

Keith Bombard



If you've ever worked with Access to develop large, multi-user databases with thousands of objects, then you'll want to think about limiting user access to those objects. In this article, Keith Bombard shows you a simple and proven technique to manage access using user-driven menus.

RECENTLY I had the opportunity to develop a very large Access 97 system for a high-volume electronic printing company on the East Coast. This company builds, prints, and mails high-end brokerage and mutual fund statements for more than 300 client companies. The system's primary mission was to track the flow of print-job work orders as they made their way through the plant. Originally, the system was to be used by a limited number of users and departments. It turned out to be one of those never-ending development projects with an unlimited budget and suffering from chronic scope creep. The system and the user base grew and grew until it had grown to hundreds of forms and reports, more than 50 users, and eight different departments.

It wasn't long before users started to express their dismay at having to wade through a torrent of menu choices that they knew nothing about and cared nothing about. Each department needed specific parts of the system, so the logical solution was to group users into Menu Groups. The plan was to give each user in the department a Menu Group ID that would control which menus a user would be presented with. The project's menu system could be adapted to accommodate this enhancement without affecting the basic menu approach already in place, making the transition transparent to the end user.

This approach helped to solve another problem that was becoming a major issue: enforcing the security scheme. If users could be limited to what they saw on their menus, they would stay on course and be much less likely to run into dreaded security violation messages.

The approach that I used is one that I've found has universal appeal to users: dual list box menus (see [Figure 1](#)). The top-level list box holds "Main Topic" choices, while a subordinate-level list box holds the selected Main Menu's "SubTopic" choices. Selecting a Main Topic displays the related SubTopic. Selecting a SubTopic opens the form, table, or report associated with that SubTopic. The Main Topics displayed to the user form a Menu Group. A user table holds the Menu Group

ID for a given user so that the correct list of Main Topics for the current user is displayed. The whole system is driven by a set of tables that can be updated at any time. In fact, while I've built an extensive system to manage my menus, implementing the menus requires very little code at all.

Tables make it work

I use a set of four tables to implement my menu system. Before describing those tables, I should explain my naming convention. I always number the tables in my databases (a tip from an ex-mainframe DB2 compatriot who never had the flexibility in table naming we've become accustomed to in Access). Numbering tables provides a whole new dimension to working with tables, and I highly recommend it—especially when working with a large number of tables in a database. Numbered tables are sorted in numeric order in the database window, so finding the table you want is easier when you can't remember the exact table name or your table names are similar, provided you group your tables by number.

Most of the systems I work with have upwards of 200-300 tables, and I'd be at a great loss without a numbering scheme. I use the tables numbered in the 900+ range as system/administration tables. This makes it doubly easy to pull these tables into projects to quickly

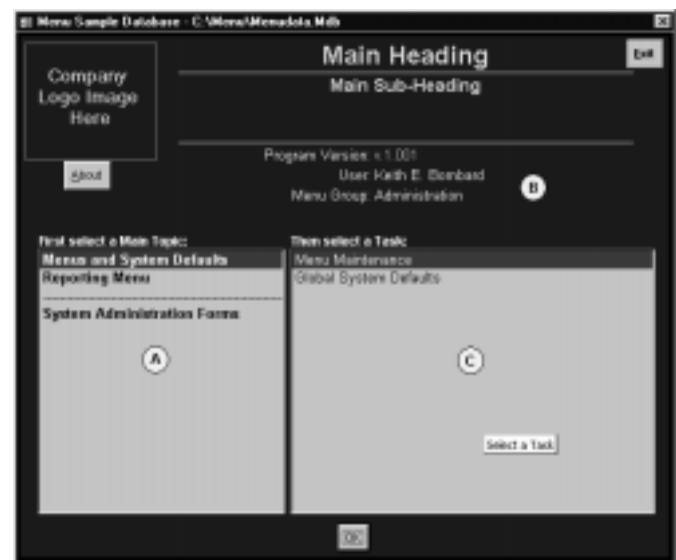


Figure 1. A dynamic menu, keyed to individual users.

establish the same base-level administration services in each new project. The tables that I use when implementing my multi-level menus are:

- tbl907_MenuMaster: The top-level menu table. It holds the MenuGroupID key field and the basic information about the Menu Group, and it establishes each Menu Group in the system. Menu Groups are collections of Main Topics and SubTopics.
- tbl906_MainMenuTopic: The Menu Groups' Main Topic descriptions.
- tbl905_MainMenuSubTopic: Each Main Topics' SubTopic descriptions and other critical object data needed at runtime.
- tbl900_UserMaster: This table is the main user table. It holds basic information specific to each user. A key field in this table is the MenuGroupID field. Each user is assigned a specific MenuGroupID, which keys them into the proper Menu Group at logon.

The relationship diagram shown in [Figure 2](#) illustrates the tables and relationships that support my multi-level menus.

Forms make it work

The tables aren't much use unless you can update them. Three forms support the menuing tables:

- Menu Maintenance form
- User Master form
- Main Menu form

Menu Maintenance form

The Menu Maintenance form (see [Figure 3](#)) implements the concept of Menu Groups and is used to manage all menus in the application. Four data tables and four subforms are shown in this form that display Menu Groups, Menu Main Topics, Menu SubTopics, and the User Master records. This form is used to manipulate the Main Menu display choices for each Menu Group. Users can do the following:

- Add, change, or delete Menu Groups (area A)
- Add, change, or delete Main Topics in a specific Menu

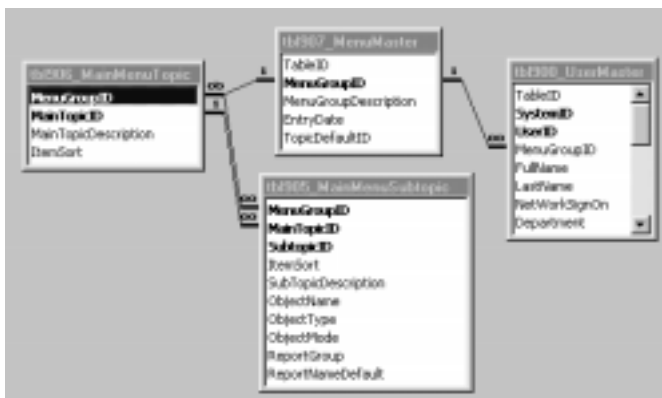


Figure 2. Menu-table relationships.

Group (area B)

- Add, change, or delete SubTopics from a selected Main Topic choice (area C)
- Add, change, or delete a user's MenuGroupID assignment (area D)

Users can also be reassigned into Menu Groups.

Menu Groups define the highest level of menu processing. Menu Groups usually represent a group of personnel performing similar tasks. Examples include separate Menu Groups for the data entry and administration groups. The administration group might even be defined as an unabridged list of menu choices.

As you move your cursor into different Menu Group records, related data dynamically displays in the other three subforms. To add a new Menu Group, enter a new MenuGroupID value (for the sake of consistency, try to stay in the same numeric series as the previous rows) along with a menu description. To delete a Menu Group, select the row for deletion with your mouse and press the Delete key. The form won't let you delete a Menu Group if a user is still linked to the group. When this happens, you'll need to change that user's MenuGroupID in the bottom subform to a different one.

Main Topics are subordinate to the Menu Groups and function as high-level categories for associated SubTopics. These choices display in the left-side list box of the Main Menu form. The sort order column is used to control the display, sorting in both the Main Menu form and this Maintenance form. These values can be edited to change the display position of the listed item. You can even use decimal values to position menu entries between other entries. As you move your cursor into different Main Topic records, related data dynamically displays in the SubTopic subform.

SubTopics are the specific menu choices appearing in the right-hand list box of the Main Menu. They're subordinate to the Main Topics and hold the target object



Figure 3. The Menu Maintenance form.

references that open when the user executes a menu choice from the Main Menu form. As in the Main Topic subform, the Sort Order field controls display sorting. SubTopics can reference a table, form, query, report, or macro. Sort Order values can also reference a custom report-menu form.

Users who belong to the selected Menu Group are listed in the bottom subform. You can remove users from a selected group by changing their MenuGroupID, assigning a user into the new Menu Group.

The MenuGroupID field

The MenuGroupID field in the user table determines which menu the user will see when the Main Menu form loads. I've seen a wide range of uses for user tables in Access systems, from no user table at all up to very elaborate user tables that hold lots of information. I think that too much data in a user table can lead to extra administration costs and even, in extreme cases, user resentment about the amount of data stored about them. On the other hand, systems without a user table can be mired in hard code in order to implement user-specific processing. To implement my menuing system, you'll need a field in a user table that matches the value of Access's own CurrentUser variable. Figure 4 shows the user table that I use for my menuing system.

I use a function called CUserID() to return the three-digit alphabetic UserID field from the user table (I usually use user initials as my UserID). The function is quite simple: It uses the Access CurrentUser function, performs a lookup in the user table by matching CurrentUser with the FullName field, and returns the UserID for that user. Once the UserID is retrieved, it's stored in the global variable glbUserID, and the variable is used in the rest of the application. This technique speeds up repeated processing by limiting the table lookup to the first occurrence:

```
Function CUserID() As String
```

```
On Error GoTo CUserID_ERR

If Len(Trim(glbUserID)) = 0 Then
    glbUserID = DLookup("UserID", _
        "tbl1900_UserMaster", _
        "CurrentUser() = FullName")
    CUserID = glbUserID
Else
    CUserID = glbUserID
End If

CUserID_EXIT:
Exit Function

CUserID_ERR:
Select Case Err.Number
Case Else
Call ScrError(Err, Error, _
    "CUserID", MsgHeader)
Resume CUserID_EXIT
End Select
End Function
```

The user table that's included in the sample database with this month's Source Code files (available at www.smartaccessnewsletter.com) was pulled from a recent project. The preceding user form is an abridged version of the form used in that project. You'll find additional fields in the table that aren't displayed on the form.

Of course, in order for this function to work, the FullName field in the user table should exactly match the Access user name (Access's CurrentUser variable). I recommend that new users be added to the system using the Add New User button on this form. Code behind the Add form (not shown here) will add the new user's name and PID to the system workgroup file at the same time the new user is added to the user table. I also recommend that you use the "/user UserName" command line parameter syntax in the Access launch shortcut on each user's desktop. This will ensure that the correct user is logged in at startup.

Menu selection

The Main Menu form (see Figure 1) implements Main Topic and SubTopic menu selections for each user. This is a dual list box form; the left-hand list box holds the Main Topics, and the right-hand list box holds the Main Topics' subordinate SubTopics.

The Main Topic list box (A) has a SQL statement as its row source that links the current user's MenuGroupID field to the Main Topic MenuGroupID field (see Figure 5).



Figure 4. The User Master form displays essential data for each user.

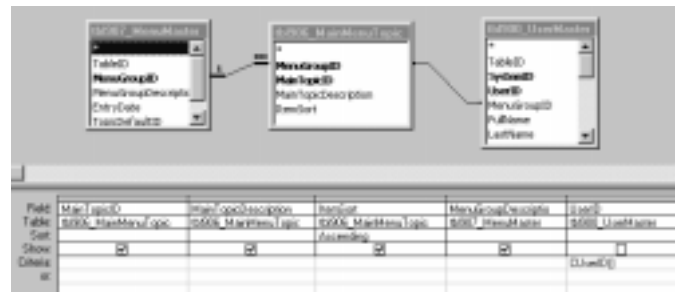


Figure 5. SQL Select row source for the Main Topics list box.

The result is that only Main Topics belonging to the user's Menu Group are listed. Users with the same Menu Group see the same menu choices. When you scroll through the Main Topic choices in the left-hand list box, the AfterUpdate event fires and requeries the right-hand list box, refreshing the corresponding SubTopic listing.

Using the information in the tables, you can load the two list boxes displayed in Figure 2. When a user selects an entry in the SubTopic list box, you can use the information from the tbl905_MainMenuSubTopic to open the Access object associated with the entry. I load the information into columns in a list box. In the following sample code, the list box is called lstST. The code checks to see whether the object associated with the list box entry is a Form, Report, or Table in order to select the right DoCmd method. If the item is a Form, the code then checks to see in what mode to open the Form (Add, Edit, or default) in order to set the right parameters. Finally, the code uses the object name to open the object:

```

If Me![lstST].Column(2) = "Form" Then
  If Forms![frmMainMenu]![lstST].Column(3) _
    = 2 Then
    DoCmd.OpenForm _
      Forms![frmMainMenu]![lstST].Column(1)
  Else
    DoCmd.OpenForm _
      Forms![frmMainMenu]![lstST].Column(1), _

```

```

      ' ' ' -
      Forms![frmMainMenu]![lstST].Column(3)
  End If
End If
If Me![lstST].Column(2) = "Report" Then
  DoCmd.OpenReport _
    Forms![frmMainMenu]![lstST].Column(1), _
    A_PREVIEW
ElseIf Me![lstST].Column(2) = "Table" Then
  DoCmd.OpenTable _
    Forms![frmMainMenu]![lstST].Column(1), _
    acViewNormal, acReadOnly
End If

```

The real benefit of this system is that the menus are completely table-driven. Adding a new form to the Main Menu just consists of adding the appropriate entries to the underlying tables. You'll find a complete implementation of the system with this month's Source Code files, ready to add to your application. ▲

 [TBLMENU.ZIP at www.smartaccessnewsletter.com](http://www.smartaccessnewsletter.com)

Keith Bombard is a contract programmer who's employed by Howard Systems International and specializes in large Access implementations. He's been developing MS Office-based solutions for multiple clients since 1993. Prior to that, Keith was a systems manager in the financial industry. He's currently on an extended assignment building three large Access databases for the Connecticut Department of Environmental Protection. Keith.Bombard@PO.STATE.CT.US, KeithBombard@Home.com.

Everything Doesn't Happen at Once: Loosely Coupled Events

Peter Vogel



If you're running Access on a Windows 2000 computer, you can take advantage of COM+ to build more sophisticated applications. Peter Vogel looks at using Loosely Coupled Events to create applications that run asynchronously.

NOT everything in your application has to happen at the same time. In fact, applications that require all of their components to be online and available at the same time are more difficult to maintain and demand more resources than applications that can be segmented. If you can break your application up so that some processing can be “de-coupled” from the rest of the application, you can gain some significant benefits. The object is to have parts of your application run asynchronously, allowing components of your application to respond to requests from the other parts of your system at some convenient time.

For instance, a sales order application might accept orders all day without making any updates to support shipping the goods that have been ordered. Sometime after the sales order system shuts down, the portion of the application that does the shipping updates could run and complete the processing. There are four major benefits to this kind of design:

- The order system can continue to run even if the shipping database isn't available.
- Contention problems over the data are reduced because updates are deferred until data is available.
- Response times on the front end of the system (the user interface) are improved because less is being done during data entry.
- Shipping updates can be done during off hours to reduce demands on hardware.

The last benefit is important because it allows your application to be more scalable. You might have orders being entered at the rate of several dozen per second at your peak hours. However, the de-coupled shipping process could run along behind the order entry process, falling behind during the peaks but catching up with the orders during the slow periods. The load on your computer is distributed more evenly over the day.

QC, LCE, MSMQ

In Windows 2000 with COM+, building this kind of functionality into your Access application is easy. COM+ provides two services that allow you to de-couple portions of your system and implement asynchronous processing: Queued Components (QCs) and Loosely Coupled Events (LCEs).

You can achieve much of the functionality of QCs and LCEs by using Microsoft Message Queue (MSMQ). However, QCs and LCEs let you accomplish the same things as MSMQ with much less effort. To use MSMQ directly, you have to learn how to use the MSMQ objects to create a message and put it on a queue. You also have to create a listener program to process your messages—something that often requires creating an NT service. With QCs and LCEs, you just need a little Visual Basic and, thanks to VBA being the programming language for both Access and Visual Basic, you should feel right at home.

A QC is an object that you can create, call its methods, and release without the object ever actually being loaded into memory. Instead, a message is placed on an MSMQ queue for later processing. At some later time, COM+ will then load the object into memory, call the requested method, and release the object. There are two problems with QCs: You need MSMQ, and you have to initiate processing of the queued messages.

LCEs, on the other hand, are almost a free gift from COM+. You create an LCE object using the `New` keyword, exactly as you'd create any other object. With the object created, you call the object's methods, passing whatever parameters you want. You then set the object variable to `Nothing` to destroy the object. After you release the object variable, COM+ will create the object and run the methods you requested. This code creates an LCE, calls a method, and releases it:

```
Dim objLCE As Ship.LCEComponent
Set objLCE = New Ship.LCEComponent
objLCE.UpdateShip strCustNo, strOrderNo
Set objLCE = Nothing
```

If this code looks like the code you'd use to call an ordinary component, you're right—it is. From the

programmer's point of view, an LCE is indistinguishable from an ordinary component. The LCE can then start up Access and run the de-coupled portion of your application.

There are two limitations with LCEs. The major limitation is design: You must have an application whose components can be run independently of each other. The other limitation is that the communication between your Access application and the LCE is one-way: You can pass data to the LCE, but you can't have the LCE return data to you. After all, by the time the LCE actually runs, your application might have completely shut down.

Creating an LCE

Creating an LCE is relatively straightforward. First, in Visual Basic, you must create an ActiveX DLL project. I gave my project the name Ship. You then need to use a Class module to define the interface that your LCE will implement. Defining the interface begins by giving the Class module the name that you want to use for your LCE (in my case, that's LCEComponent). The next step is to write the methods that make up the LCE's interface. A method is just a subroutine, declared as Public, in a Class module. This code declares the UpdateShip method that I used in my sample code:

```
Public Subroutine UpdateShip( _  
    ByVal CustomerNumber As String, _  
    ByVal OrderNumber As String)  
  
End Sub
```

To support the limitations of an LCE, the method must be a subroutine (because an LCE can't return a value), and its parameters must be declared ByVal instead of the default ByRef (again, because values can't be returned from the LCE).

You don't put your code in the Class module that defines your interface. While that Class module is the one that your application will use when working with your LCE, you actually put the code for the methods for your LCE in a separate Class. While your application refers to the Class with the interface, when it comes time to execute your LCE's code, COM+ will load the Class module with the code and execute it.

LCEs separate the code from the interface for a simple reason: It allows you to execute several different routines when an application calls a single LCE method. COM+ will load and execute all of the different implementations of any LCE interface. This leads to another way of using LCEs.

When you write an application, you can define an LCE and have your application call it but never write the code that will execute. Instead, other developers who want to integrate with your application can write code to be run when your application calls the methods of an LCE. When your application calls the methods of the LCE,

none, one, or many components that implement that interface may execute. Using LCEs allows others to integrate their processing with your application.

Implementing an LCE interface is easy to do. First, you add another Class module to your Visual Basic project. You then add an Implements statement to the new Class module that tells Visual Basic that you want to implement your LCE interface. Since the name that I gave the Class module that defined my LCE interface was LCEComponent, this is the line I use:

```
Implements LCEComponent
```

The next step is to add the methods of the interface to the new Class module. These methods must include the name of the interface that they're to implement. To implement the UpdateShip method that I defined in the LCEComponent Class module, I'd write this code:

```
Public Subroutine LCEComponent_UpdateShip( _  
    ByVal CustomerNumber As String, _  
    ByVal OrderNumber As String)  
  
End Sub
```

Finally, you add the code that implements your method to the Class module that implements the LCE interface. The name of this Class doesn't matter because your application never uses it directly (I called mine Ship1). Assuming that the code to do the shipping updates is already part of my Access application, the only thing that I needed to do is call the appropriate routines in my Access project. Sample code to load Access, open an MDB file, and call a subroutine in the project would look like this (you'll need to add Access to your Visual Basic project's References list to make this work):

```
Dim acc As Access.Application  
Set acc = New Access.Application  
acc.OpenCurrentDatabase "LCEDemo.MDB"  
acc.Run LCEDemo.PurchaseUpdate _  
    CustomerNumber, OrderNumber  
acc.CloseCurrentDatabase  
acc.Quit  
Set acc = Nothing
```

The two parts of the application are now de-coupled. The front end of the application will call the UpdateShip method of the LCEComponent, passing whatever information is required. Sometime after that, all of the Class modules that implement the LCE interface will be loaded into memory and executed. The sample code that calls the PurchaseUpdate routine in the LCEDemo method will then execute the back end of the database, using the information passed from the front end.

Setting up an LCE

All through this article, I've been telling you that an LCE component looks exactly like a standard Visual Basic object. And it does—to the programmer. However, to

make the component an LCE, it must be set up in Windows 2000 Component Services. It's the process of setting the component up in Component Services that converts a standard ActiveX DLL into an LCE.

To begin, select Start | Programs | Administrative Tools | Component Services. Expand Component Services to see the Computers folder, expand the Computers folder to see My Computer, and then expand that item to see the COM+ Applications folder (see Figure 1). Right-click on the COM+ Applications folder and select New | Application to start creating a COM+ application for your LCE. Just click the Next button in the Wizard that appears, select Create an Empty Application on the second screen, and give your application a name.

After your application is created, right-click on the Components folder that it will contain and select New | Component. The Wizard that appears will walk you through the process of adding your LCE to the application. After clicking on the Next button, the second form in the Wizard will offer you three choices. Selecting "Install new event class(es)" lets you set up an LCE. Clicking on that button brings up the screen that will let you add DLLs to your application. Since I had a single DLL that contained both my interface definition and my implementation, I only needed to add it. In Figure 2, you can see the result of adding the DLL I've described here.

After your component has been added, you'll see entries for your interface class and for the classes that implemented your interface. You can expand the interface class to reveal a Subscriptions folder, where you'll specify

which objects to execute when some application calls your LCE interface. To assign components to the interface, right-click on the interface object's Subscriptions folder and select New | Subscription. This final Wizard will let you first select the LCE interface and then select all of the components to be run when the interface is called. With that, your application is ready to run asynchronously.

The benefits of de-coupling the parts of your application can be valuable. Even if you don't want to implement your LCE, just calling an LCE in your application can allow others to integrate their processing with your application. And, as you've seen, thanks to COM+ and Windows 2000, making it all happen is simple. ▲

DOWNLOAD [LCEVENTS.ZIP at www.smartaccessnewsletter.com](http://www.smartaccessnewsletter.com)

Peter Vogel (MBA, MCSD) is the editor of *Smart Access* and a principal in PH&V Information Services. PH&V specializes in system design and development for COM/COM+ based systems. Peter has designed, built, and installed intranet and component-based systems for Bayer AG, Exxon, Christie Digital, and the Canadian Imperial Bank of Commerce. He's also the editor of Pinnacle's *XML Developer* newsletter, wrote *The Visual Basic Object and Component Handbook* (Prentice Hall), and is currently working on a book on XML for Apress. Peter teaches for Learning Tree International, wrote its Web application development course, is technical editor of its COM+ course, and co-developed the Internet version of its Relational Database Design course. His articles have appeared in every major magazine devoted to VB-based development and in the Microsoft Developer Network libraries. Peter also presents at conferences in North America and Europe. peter.vogel@phvis.com.

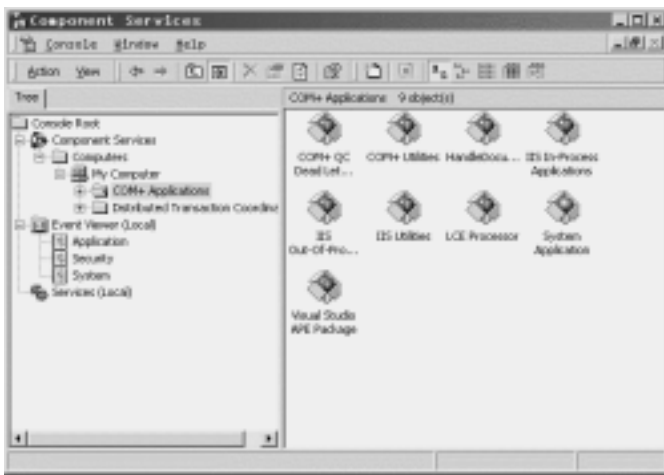


Figure 1. Windows 2000 Component Services showing the components installed on the computer.

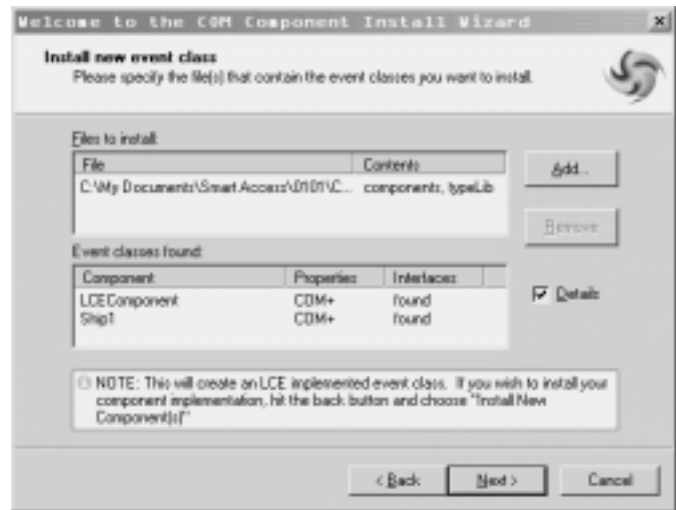


Figure 2. A COM+ application after adding a DLL to create an LCE.

Earn money!
And impress your friends!

Have a hot tip? Send it in! If it shows up in the pages of *Smart Access*, we'll send you \$25. See the back page for the address where you can send your tips.

Data Projects and SQL Server

Andy Baron

2000

This month, Andy Baron returns with a variety of answers that focus on issues related to using Access with SQL Server.

I have an Access 2000 ADP application running against SQL Server 7.0 with SP1 installed. My question is about a combo box. The row source type for the combo box is “stored proc,” and it works fine. However, I need to pass a parameter value to the stored procedure. My stored procedure works fine with a parameter when used outside of the combo box, and I even get prompted for the parameter if I don’t pass it. However, what I want to do is have my stored procedure repopulate the combo box list with the parameter taken from another control on the form. Can I code this on the property sheet (what is the syntax?), or will I have to code a function and have it populate the list? I’ve tried several combinations but keep getting runtime errors, as Access thinks that the entire line on the row source property is the name of the stored procedure.

Passing parameters to a stored procedure is a little easier with the record source of a form, where you can use the new Input Parameters property. For a combo box’s row source, you need to use code to dynamically change the value of the row source—and getting that syntax right can be tricky. Assuming that you were using the Categories and Products tables from Northwind, and that you had created a ProductsByCategoryID stored procedure, you could do something like this:

```
Private Sub cboCategoryID_AfterUpdate()  
    Me.cboProduct.RowSource = _  
        "Exec ProductsbyCategory " & _  
        Me.cboCategoryID  
End Sub
```

In a bound form, you’d also want to run that same line of code in the Current event of the form, so that the product list will change based on the category selected on the current record. Note that this won’t work unless you include the Exec (or Execute) statement as part of the text that you use to set the row source.

I’m using a form in an ADP that has a SQL statement as its underlying record source. To limit the number of records displayed on the form, I set the Server Filter By Form property to Yes. When I open the form, it opens in the Filter By Form view, and when I click on the text boxes, they become combo boxes. Using the arrow beside the field, I

expect to get a list of all records in the data table, which would allow me to choose one of the values as a filter criterion. However, all I’m getting is Is Null and Is Not Null as options. What can I do to make it list all of the values for the fields from which the user will want to choose?

There are two properties that affect how those lists are populated. First, on the Data tab of the Properties dialog box, each text box has a Filter Lookup property that you can set to Never, Database Default, or Always. If you set this property to Always, a combo box list of distinct values will be created for that text box in ServerFilterByForm view. Selecting Never for the Filter Lookup property causes the combo box for that field to contain only two items: Is Null and Is Not Null. The Database Default option will either populate the list with values or with the simple Is Null/Is Not Null based on a second property.

The second property that affects the behavior of ServerFilterByForm appears on the Edit/Find tab of the Tools | Options dialog box. Here, you can check off an option to “Show list of values in Records at server.” Checking this option causes fields that have their Filter Lookup property set to Database Default to become populated dropdown lists in ServerFilterByForm view. Clearing this check box causes those same fields to get the simple Is Null/Is Not Null list. (There’s also a property on that Edit/Find tab of the Tools | Options dialog box that allows you to prevent dropdown lists in ServerFilterByForm view from displaying more than a specified number of rows if the Database Default setting is in force.)

The filter that gets created in this view is applied on the server before the form is loaded with data, so these lists can be a good way to encourage the user to retrieve a limited number of records and keep the retrieved recordset small. Just remember, however, that those combo boxes don’t get populated by magic—Access runs queries to get all of the data for those lists. So, limiting which fields even get a list of values is a good idea.

This code works in Access 97 to give me the size of a text field in the recordset underlying a form:

```
intLength = Me.RecordsetClone. _  
    Fields(Me!txtCustomer.ControlSource).Size
```

In other words, the underlying field's properties are available through the Fields collection of the form's RecordsetClone property. Can I do this in an Access 2000 database?

In Access 2000, the RecordsetClone property will still return a DAO recordset by default in an MDB. In ADPs, however, it's an ADO recordset that's returned by default. Even in MDBs, you can have the RecordsetClone return an ADO recordset if you've programmatically assigned an ADO recordset to the form's Recordset property.

So your question is whether you can get this result if you're working with a form that's using an ADO recordset. The answer is that you can, but your code needs one small change. In ADO, the field object has a DefinedSize property that will give you the information that you want:

```
intLength = _  
    Me.RecordsetClone.Fields( _  
    Me!txtCustomer.ControlSource).DefinedSize
```

I'm banging my head against the wall trying to get Input Parameters to work in an Access 2000 report. The report loads fine if I hard-code the Input Parameters; however, if I try passing a new record source and Input Parameters to the report, it doesn't seem to work! I'm using the following code in the Open event of the report, where MyCustodian() returns a valid string and pAccountListSource contains a valid stored procedure on the SQL Server:

```
Me.InputParameters = "@Custodian=MyCustodian()"  
Me.RecordSource = pAccountListSource
```

When the code runs, I always get asked for the data for the parameter. If I enter a value when asked, the correct information is returned on the report. What am I missing?

First, when using Input Parameters, you need to specify the data type after the name of the parameter. For example, you need this code to specify the data type for the Custodian parameter:

```
@Custodian varchar(15) = MyCustodian()
```

But, unfortunately, that won't solve your problem. You've discovered something that just doesn't work the way you'd expect it to work in Access 2000. Your code would work fine in a form, but Input Parameters can't be set at runtime in Access 2000 reports. The only way around this problem is to open the report in design view and set the properties before running the report. This gets pretty kludgy because you'll also have to hide the report for the time that it's in design view.

Another thing that you'd expect to work is to be able to just set the record source in the Open event, using

something like this:

```
Me.RecordSource = "Exec MyProc 5"
```

Or, in your case, this:

```
Me.RecordSource = "Exec " & pAccountListSource _  
    & "" & MyCustodian() & ""
```

Once again, this kind of thing works fine in forms but, alas, not in reports.

So, if you need to use several different stored procedures as the record sources of reports, you might find it easiest to save several copies of the report, one for each stored procedure. There's no problem using your function in the Input Parameters property—the function will be correctly evaluated at runtime.

Another thing that does work fine in reports is to include a reference to a control on a form in your Input Parameters, like this:

```
@Custodian varchar(15) = Forms!MyForm!MyControl
```

This code will even work if the form is hidden, giving you another way to dynamically change the parameter value being passed to a report.

How can I use VBA code to simulate a random autonumber? I've moved my Access 97 database to SQL Server. But the autonumber (identity) on SQL Server doesn't seem to have a random setting. I want to generate random autonumbers from my Access form without using the SQL autonumber.

If you still have the Jet database engine around, as you would for sure if you're using an MDB in Access, then you can just open a recordset on a table that has a random autonumber field, do an .AddNew to add a record to the recordset, fill in some value in another field in the record to force the autonumber to be generated, get the autonumber fields value, and then cancel the insert to prevent the record being added:

```
Public Function GetRndAuto() As Long  
    Dim rst As DAO.Recordset  
    Set rst = _  
        CurrentDb.OpenRecordset("tblRandomAuto")  
    rst.AddNew  
    rst!DummyField = 3  
    GetRndAuto = rst!RandomAutonumberField  
    rst.CancelUpdate  
    Set rst = Nothing  
End Function
```

If you want to keep Jet out of the picture, you can use the VBA Rnd function, which returns a number between 0 and 1 (not including 1). You can multiply this value by the largest number that you want to have returned to get a number between 0 and that number. Before calling the Rnd function, you should always call

the Randomize statement. The Rnd function algorithm relies on a seed value to produce its result, and unless you specify a seed, each result is used as the seed for the next iteration. So, the same seed will always generate the same "random" series of numbers. The Randomize statement makes sure that you don't get the same series every time by using the current date/time to create a unique seed.

You probably should call the Rnd function twice—once to get an absolute value in the desired range, and a second time to generate a random sign (+/-), unless you want all positive numbers. For example, this code calls Randomize, generates a number between 0 and 2 billion, and then determines whether to convert that number into a negative value:

```
Dim lngRand as long
Randomize
lngRand=cLng(Rnd * 2000000000)
If Rnd >= .5 Then
    lngRand = lngRand * -1
End If
```

In case you ever decide to generate your random numbers in a SQL Server stored procedure, Transact-SQL includes similar functionality to the Rnd function in T-SQL's RAND function. The RAND function accepts a seed parameter. You can do the same thing that VBA's Randomize statement does by passing time-related data to the RAND function:

```
SELECT RAND( (DATEPART(mm, GETDATE()) * 100000 )
+ (DATEPART(ss, GETDATE()) * 1000 )
+ DATEPART(ms, GETDATE()) )
```

[In DAO, it was a fairly simple task to reset a querydef's SQL property through code. How do I accomplish the same thing with ADO \(or ADOX\)? I can retrieve the CommandText from an ADOX View object just fine, but I don't seem to be able to set it and have it save the results. I don't have to create a new one, do I?](#)

You can indeed use ADOX to change the SQL statement behind a View. All that's necessary is to use an ADO Command object to work with the View. Each View has a Command object associated with it that you can access through the View's Command property. All you need to do is create an object variable that can work with an ADO Command object and point that variable at the View's Command property.

However, the tricky part is the way you need to use that Command object variable once you've set it up. In theory, if an object variable is set equal to the View's Command object, then you should be working with the Command object behind the View through the object variable. As a result, changing the object variable's CommandText should change the CommandText of the View's Command object. That's the theory. In practice,

after setting the CommandText property through the object variable, you must set the View's Command property equal to the updated Command object to make the change work. The code looks like this:

```
Dim cat As ADOX.Catalog
Dim cmd As ADODB.Command

Set cat = New ADOX.Catalog
Set cat.ActiveConnection = _
    CurrentProject.Connection
Set cmd = _
    cat.Views("MyQuery").Command
cmd.CommandText = _
    "My new SQL string..."

Set cat.Views("MyQuery").Command = cmd

Set cmd = Nothing
Set cat = Nothing
```

[How can I use ADO to change the properties of a pass-through query? I want to be able to change the SQL, and sometimes I also need to change the ODBC connect string.](#)

The code for this is very similar to the previous answer, even though this code applies to MDBs rather than ADPs. The same ADOX Catalog and Command objects can be used in an MDB as in an ADP. For pass-through queries, you use the Procedures collection of the Catalog object, rather than the Views collection.

The Jet OLE DB provider supplies some custom properties that you can use to set the properties that are specific to pass-through queries. It's not always obvious what the exposed properties are. For instance, if the query contains a SQL statement that returns records, the query's Record Returns property must be set to True. The OLE DB property that's equivalent to the Returns Records property that you see in the Access properties window for a pass-through query is Jet OLEDB:Pass Through Query Bulk-Op. Set this query to False if your query returns records.

This routine, passed a query name, updates the query's SQL statement, connection string, and whether or not the query will return records:

```
Public Sub PassThroughFixupADO( _
    ByVal strQueryName As String, _
    ByVal strSQL As String, _
    Optional varConnect As Variant, _
    Optional fRetRecords As Boolean = True)
Dim cat As ADOX.Catalog
Dim cmd As ADODB.Command

Set cat = New ADOX.Catalog
Set cat.ActiveConnection = _
    CurrentProject.Connection
Set cmd = cat.Procedures(strQueryName).Command
cmd.Properties( _
    "Jet OLEDB:ODBC Pass-Through Statement") = True
cmd.CommandText = strSQL
If Not IsMissing(varConnect) Then
    cmd.Properties( _
        "Jet OLEDB:Pass Through Query Connect String") _
        = CStr(varConnect)
End If
```

```
cmd.Properties( _
  "Jet OLEDB:Pass Through Query Bulk-Op") = _
  Not fRetRecords
Set cat.Procedures(strQueryName).Command = cmd
Set cmd = Nothing
Set cat = Nothing
End Sub
```

In Access, does the MaxRecords property of a query affect where the processing of the records is performed? In other words, if I'm using linked ODBC tables in a query and I set the MaxRecords property, will selection and processing happen on the server, or is the processing performed on the client?

There's a common misconception that when you create a query in Access using linked ODBC tables, all of the data gets sent to Access and is processed on the client. That usually isn't true. If Access can convert the query to a valid ODBC SQL statement, it will, and the query will be processed on the server. Only the result set travels back across the wire. The only time you get extra data is if you include something in your query (like a call to a custom VBA function) that can only be processed by Access.

MaxRecords processing is always performed on the server (at least if the server supports it). If you have an Access-specific piece of business in your query, such as a user-defined function in the WHERE clause, that won't cause the MaxRecords processing to move to the client. This means that when using MaxRecords, you might not get the results you expect. Here's an example that you can try out by linking to the titleauthor table in the pubs sample database. This SQL statement uses a user-defined function (MyFunction):

```
SELECT dbo_titleauthor.au_id
FROM dbo_titleauthor
WHERE ((MyFunction([title_id]))="MC"));

Public Function MyFunction(strInput As String) _
  As String
  MyFunction = Left(strInput, 2)
End Function
```

Running this query against SQL Server's pubs database will return three records.

Now set MaxRecords to 2 in the query properties window, and the query will return zero records, not two. Why? The way to find out is to run a trace in SQL Profiler and watch the actual T-SQL statements that are being processed in SQL Server. Because the MaxRecords setting is applied on the server, the statements processed on the server are:

```
SET ROWCOUNT 2
SELECT "title_id","dbo"."titleauthor"."au_id",
  "dbo"."titleauthor"."title_id"
FROM "dbo"."titleauthor"
```

Notice that the query requests the title_id (twice!) even though it's not in the SELECT clause of the original Access query. This is because Jet rewrites the SQL

statement before sending it to SQL Server. Jet removes the portion of the WHERE clause that uses the user-defined function because it can't be processed on the server. Jet adds the title_id field to the SELECT clause because it's needed by the Access function when the WHERE clause is processed on the client. This is an example of a query that requires some processing to be done in Access: All of the records in the table need to be returned because of a function in the predicate that can't be resolved on the server. But this doesn't negate MaxRecords processing on the server, because ROWCOUNT was still set to 2 on the server.

Since ROWCOUNT was set to 2 on the server, only the first two records of the records retrieved on the server are returned. Since those two records don't happen to include any of the ones with "MC," the client-side Jet processing eliminates those two records and the query returns zero records.

Now change the query to use a predicate that ODBC/SQL Server can resolve:

```
SELECT dbo_titleauthor.au_id
FROM dbo_titleauthor
WHERE (((Left([title_id],2))="MC"));
```

This time, what gets processed on the server is a more complex set of statements. These statements include the following:

```
SET ROWCOUNT 2
SELECT "dbo"."titleauthor"."au_id",
  "dbo"."titleauthor"."title_id"
FROM "dbo"."titleauthor"
WHERE ({fn left("title_id" ,2 )}= 'MC' )
```

The strange "{fn...}" syntax is an escape sequence for ODBC SQL. In this case, T-SQL has its own Left function, but the {} syntax is used to allow ODBC functions that aren't part of T-SQL to be processed by SQL Server.

This statement returns the expected two records, but that's partly a matter of luck. You really can't be sure which records will be returned unless you use an ORDER BY clause, since it will depend on the unpredictable order in which the query processes its results. Without it (as in the examples), you really can't be sure which records will be returned.

And here's one final question that I can't resist including, even though it has nothing to do with SQL Server...

[My application was working fine in Access 2000, until I began using SR-1. Why has Microsoft disabled using something like the following code?](#)

```
DoCmd.TransferText acExportDelim,
  "ExportAccount", "ExportAccount", _
  (strExport & ".Act"), True
```

If I save the file using a .txt extension instead of .Act, this works fine. However, saving to a file with .Act produces error 3027 stating that the database is locked. I found that most extensions have been disabled. Is there a good reason? They should at least include a better error message! It was painful to locate the solution.

Unfortunately, in an effort to close up security holes, Microsoft sometimes makes changes that break existing code (ask anyone who's written code that automates Outlook!). The purpose here was to keep folks with more brains than sense from being able to create diabolical Access viruses that would destroy important system files. But in this case, the new security feature is completely within your control, once you know where to look in the Registry. A new Registry key was added that allows you to override the new limitations on which text files can be manipulated by Jet. You can add .Act back in by editing the following Registry key:

HKEY_LOCAL_MACHINE\Software\Microsoft\Jet\4.0\Engines\Text

Change the value named "DisabledExtensions" to:

!txt, csv, tab, asc, tmp, htm, html, act

Note that this list begins with an exclamation point, which is used here as a logical Not operator, so even though the Registry value is called "DisabledExtensions," the list actually includes the enabled ones. ▲

Contributing Editor Andy Baron, based in Singer Island, FL, is an independent developer, a senior consultant with MCW Technologies (www.mcwtech.com), and a trainer for Application Developer's Training Company (www.appdev.com). Since 1995, Andy has been selected by Microsoft as an Access MVP, based on his contributions to Microsoft's online support. Along with Mary Chipman, Andy is co-author of *Microsoft Access Developer's Guide to SQL Server*. Andy_Baron@msn.com.

Temporary Tables...

Continued from page 9

Sales Reps. The top right corner shows a preference form (frmSalesRegionPreference in the sample database), where a user can select the Sales Region(s) to work with. The Save Preferences button calls tmpCreateFromListbox, killing, creating, loading, and linking a CSV table called tmpSalesRegion that lists the selected Sales Regions. The bottom left corner shows the filtering query (tfltSalesRegion), where the tmpSalesRegion table restricts the master tblSalesRegion table to the user's current preference. This query, tfltSalesRegion, can be used in a form or report, or in place of any recordset, that's "preference-specific." The bottom right corner shows an example of a query (qselBudgetReport) that might drive a preference-specific budget report.

Although we now use partial replication on the project I mentioned, we've used temporary tables to store transient data and provide horizontal segmentation of data on many projects since then.

Solving concurrency issues

My last example demonstrates how I've used CSV files to allow many users to simultaneously run the same reports with varying criteria.

For performance reasons, I always install the MDB that contains my application's front end (forms, reports, and so forth) on each user's desktop. This is faster than pulling forms and reports over the network. My front ends generally have a single Report Submission form like the one shown in [Figure 5](#) (on page 22). The top left combo

box is bound to a table that lists available reports. When you select a report, the combo box's AfterUpdate event fires, checks the combo box's hidden columns, and enables the list boxes as appropriate for the selected

report. You can then select the criteria for the report from the list box. When you click Preview or Print, the code creates CSV tables for each list box, including the disabled ones (this is why the code in my function treats disabled list boxes as if all of the items in them were selected).

Most of what look like different reports to my users are actually the same underlying report. Each report's query consults a function to perform a dynamic Group By, depending on which version of the report has been selected. I link every CSV that might need to be restricted, knowing that if that list box isn't enabled, every value will be present.

It sounds complicated, and it sounds like a lot of work. Okay, it was. However, it also means that I have only five real reports to maintain, but my users see 90 versions. The reports run fast, too: The longest running report completes in less than 10 seconds with realistic volumes, regardless of what Group By has been chosen. And, because the application front end is local, the users each get their own CSV tables and can run the same reports without affecting other users' criteria.

These examples should give you some ideas to extend query functionality, store transient data, and solve concurrency issues in your applications. I invite you to use my tmpCreateFromListbox function and to

The screenshot shows a 'Reports' window with the following fields:

- Report: Forecast Oil By Province
- \$US to \$CDN: 0.66
- Year: 2001
- Country: Canada
- Province: Alberta
- Area: North Dakota / Saskatchewan, Northeastern British Columbia, Southern Alberta / Montana, West Central Alberta
- Field: Alderson, Beaver Creek/Roosevelt, Benson, Brownfield, Calais, Chain/Delia/Craigmyle
- Status: Contingent
- Budget: Base Chain, Base Clarke, Base Fox, Base Knutson, Base Lougheed, Base Rabbit
- Project: 10-29 Bluesky Well, 11-18 Oil Recompletion, C19I Slave Point, Land Purchase, Ocean Acquisition, Seismic Banif Play
- Capital Group: A&D, Completions
- Comment: Smart Access Example for TempCSV article

Figure 5. A consolidated report submission form.

experiment with my TempCSV technique. Finally, pay close attention to the way your code causes database bloat. And keep compacting! ▲

DOWNLOAD [CSVTEMP.ZIP at www.smartaccessnewsletter.com](http://www.smartaccessnewsletter.com)

Doug Den Hoed is a founder of Lumina Systems Delivery in Calgary, Canada, which specializes in customized software solutions using Access, Visual Basic, InterDev, SQL Server, and Oracle. Doug recently released The KB™ (www.thekb.com), the commercial Access application that inspired this article. doug.denhoed@home.com.

Downloads January 2001 Source Code

- [CSVTEMP.ZIP](#)—Doug Den Hoed provides a Christmas gift with a sample application that uses comma-delimited files to store temporary information. The application includes a standard routine for loading CSV files with data from a list box. Finally, the package includes a sample application that demonstrates how typical Access activities can cause your MDB to expand. (Access 97)
- [TBLMENU.ZIP](#)—Keith Bombard has sent along all of the administrative forms for his user- and table-driven menuing system. The sample application shows how menus are generated from the tables in the system. (Access 97)
- [LCEVENTS.ZIP](#)—Peter Vogel's download shows how to use Loosely Coupled Events to break your Access application into two parts that can process asynchronously. You'll need Visual Basic to compile some of the code. (Access 97)
- [SA0101.ZIP](#) and [SA0101.CHM](#)—The latest update to the cumulative *Smart Access* index comes in Access 2.0, Access 95, and Access 97 versions. It includes full descriptions of every article since the January 1999 issue of *Smart Access*, as well as an index to each month's Developer Disks/Subscriber Downloads. (The index is available in "SA0101.ZIP—The Complete January 2001 Source Code.")

Your subscription just became even more valuable— upgrade for FREE to gain full online access!

Pinnacle Publishing is pleased to announce that all subscribers will now have free online access to the entire *Smart Access* Web site. In addition to the print newsletter and 24x7 access to the Source Code files, your subscription now includes access to the fully searchable past-issue archives. You'll also be able to read the current issue online, before it even hits the mail. All you have to do is upgrade online, and it's FREE!

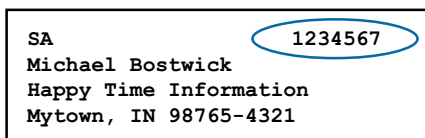
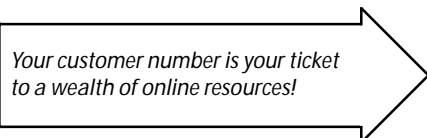
It's easy to gain access

Point your browser to <http://www.myaccesssource.com/> upgrade, and then enter your customer number as it appears on your mailing label. (If you don't have a mailing label

available, call us at 800-788-1900 or 770-992-9401, and a Customer Service Representative will provide the number for you.) You'll be asked to create a unique user name and password—and that's all there is to it! You'll receive a confirmation e-mail that provides a record of your personal Login information; you can then access all of the valuable resources available on the site simply by clicking the Login button on the *Smart Access* home page and entering your new user name and password when prompted.

Upgrade your subscription today!

Upgrade at www.myaccesssource.com/upgrade—it's free, and you'll gain access to a wealth of Access resources.



Smart Access Subscription Information:
1-800-788-1900 or <http://www.smartaccessnewsletter.com>

Subscription rates:
United States: One year (12 issues): \$139; Two years (24 issues): \$250
Canada:* One year: \$154; two years: \$265 Standard
Other:* One year: \$159; two years: \$270

Single issue rate: \$15 (\$17.50 in Canada; \$20 outside North America)*

Australian newsletter orders:
Ashpoint Pty., Ltd., 9 Arthur Street,
Dover Heights, N.S.W. 2030, Australia.
Phone: +61 2-9371-7399. Fax: +61 2-9371-0180.
E-mail: sales@ashpoint.com.au
Internet: <http://www.ashpoint.com.au>

* Funds must be in U.S. currency.

Editor Peter Vogel; Contributing Editors Andy Baron, Mary Chipman, Ken Getz, Mike Gunderloy, Paul Litwin, Garry Robinson; Publisher Robert Williford; Vice President/General Manager Connie Austin; Executive Editor of IT Farion Grove; Executive Editor Heidi Frost

Direct all editorial, advertising, or subscription-related questions to Pinnacle Publishing, Inc.:
1-800-788-1900 or 770-992-9401
Fax: 770-993-4323

Pinnacle Publishing, Inc.
PO Box 769389
Roswell, GA 30076-8220

E-mail: smartacc@pinpub.com

Pinnacle Web Site:
<http://www.pinnaclepublishing.com>

Access technical support:
Call Microsoft at 425-635-7050

Smart Access (ISSN 1066-7911) is published monthly (12 times per year) by Pinnacle Publishing, Inc., 1000 Holcomb Woods Pkwy, Bldg 200, Suite 280, Roswell, GA 30076-2587.

POSTMASTER: Send address changes to *Smart Access*, PO Box 769389, Roswell, GA 30076-8220.

Copyright © 2001 by Pinnacle Publishing, Inc. All rights reserved. No part of this periodical may be used or reproduced in any fashion whatsoever (except in the case of brief quotations embodied in critical articles and reviews) without the prior written consent of Pinnacle Publishing, Inc. Printed in the United States of America.

Brand and product names are trademarks or registered trademarks of their respective holders. Microsoft is a registered trademark of Microsoft Corporation. Microsoft Access is a registered trademark of Microsoft Corporation. *Smart Access* is an independent publication not affiliated with Microsoft Corporation. Microsoft Corporation is not responsible in any way for the editorial policy or other contents of the publication.

This publication is intended as a general guide. It covers a highly technical and complex subject and should not be used for making decisions concerning specific products or applications. This publication is sold as is, without warranty of any kind, either express

or implied, respecting the contents of this publication, including but not limited to implied warranties for the publication, quality, performance, merchantability, or fitness for any particular purpose. Pinnacle Publishing, Inc., shall not be liable to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to be caused directly or indirectly by this publication. Articles published in *Smart Access* do not necessarily reflect the viewpoint of Pinnacle Publishing, Inc. Inclusion of advertising inserts does not constitute an endorsement by Pinnacle Publishing, Inc., or *Smart Access*.



The Source Code portion of the *Smart Access* Web site is available to paid subscribers only. Log in for access to all current and archive content and source code. For access to this issue only, go to www.smartaccessnewsletter.com, click on "Source Code," select the file(s) you want from this issue, and enter the User name and Password at right when prompted.

User name

Password