Smart Access

*Solutions for Microsoft® Access® Developers*

# Using Invisible Forms to Track Users

Tobi K. Hoffman


Migrating a project into the Citrix platform provided a workable environment for Tobi Hoffman's company's people nationwide to access a single database, yet presented some unexpected challenges. Since Citrix forces all users into a single application, keeping track of users' individual security clearances was almost impossible—until the application got a form that no one saw.

THE Citrix system provides some special programming challenges. Citrix allows people to connect to a remote computer and work with the applications on that computer as if it were an extension of their own desktop. The users are actually running the application on the Citrix server, and (here's the problem) the users are opening multiple instances of the application—each user gets his own copy of the same EXE, all running on the remote server. Therefore you break what many of us have learned as a basic rule of Access development: Don't have many people using the same front-end database. The novice Access developer often learns this lesson by putting a front-end database (with its forms, queries, and reports) on a network server to be shared by all users, and then wondering why everything runs so slowly. In a crisis, especially if the database also holds all the tables, forms, queries, reports, and code, they find that their databases are frequently corrupted.

The lesson is that you should split the database into two parts:

- A compiled (.mde) front end that holds the queries, forms, reports, and whatever code you need to support your application
- A back end with the tables containing the actual data

Each user gets the front end installed on his or her computer, while the back-end database is installed on a network server. The tables in the back end

*Continues on page 4*

**February 2003**

Volume 11, Number 2

**1** **Using Invisible Forms to Track Users**
*Tobi K. Hoffman*

**3** **Editorial: Coming Next: Access 11**
*Peter Vogel*

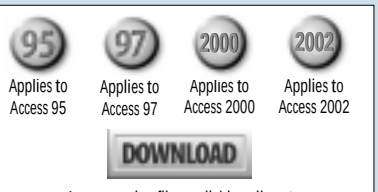
**8** **Navigating Through Recursion, Part 2**
*Christopher Weber*

**12** **Tip: An Object-Oriented Speed Tip**
*Peter Vogel*

**13** **Working T-SQL: Using Triggers**
*Russell Sinclair*

**18** **Access Answers: Combining Tables**
*Peter Vogel*

**20** **February 2003 Downloads**




Accompanying files available online at www.smartaccessnewsletter.com

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.

# Using Invisible Forms...

*Continued from page 1*

get linked into each front end from the network server. This, of course, creates a problem when it comes time to update the many copies of the front end scattered around on the users' computers.

In Citrix, both databases reside on the server, and everyone comes in there and uses the same front end to link to the back-end tables. All the processing is done on the Citrix server, and Citrix takes care of sending the desktop image to the users.

Given these problems, you may wonder why you should even consider using Citrix. It may just be that Citrix is forced on you, of course. However, Citrix is useful in a number of situations—see my sidebar "Introducing Citrix" (on page 5) for more information. Among other things, as you'll see, since there's only one copy of the front-end database, upgrades are easy.

## The problem: Controlling user access

My application (available in the Download file at www.vb123.com/kb) demonstrates the kinds of problems that occur when you use a single front-end database. My application provides four levels of user access, which I call Office, Manager, Finance, and Developer (Developer is restricted to my boss and me). Each user has a login name, password, and security
level that determines what the user can see and change. Users work for a particular office; offices manage a set of customers.

There are some drawbacks to using Citrix that are independent of Access. All users have a shortcut to the MDB file on their Citrix desktop (as well as a shortcut to the user's manual). Some people mistakenly set up the application by dragging a copy of the database—instead of the shortcut to the database—onto their desktop. These people, of course, fell behind whenever I installed an upgrade since they were using their own copy of the database instead of the central copy that I'd update. Eventually something would stop working, or a new feature that I'd announced wouldn't show up, and I'd get a phone call. Both the application's login window and its main menu contain a version number, so diagnosis of that problem is easy.

The major problem came, though, when I set up a form for reports. In the reports, each Office-level user should see only their own office and only their programs. I also let Office users create a subset of their programs for their reports. The Managers each oversee two offices, and Finance users wanted to be able to look at all offices—so those two groups couldn't have the same limitations as Office users. Also, from Manager level up, the user should be able to choose from any combination of offices and customers within those offices.

To implement this access, I set up two list boxes where the users could select their office and programs and populate temporary tables (see Figure 1). The list box that lists all the company offices isn't shown to the Office users because they don't have a choice here (see Figure 2). For Manager users, selecting an office from the list box fires an update to the temporary table tblOfficeSet to add that office. The Customers list box is updated with all Customers for which the selected offices have projects. Selecting a Customer from lstCustomer updates the temporary table tblCustomerSet with the customers to be processed.

I used these temporary tables in the queries behind the reports, and it worked fine to limit the data—most of the time. Some of the time, however, users wouldn't get the data that they wanted.

A simple experiment demonstrated the problem. My boss and I both logged into the database. I selected an office and customers for a report. He made a different selection. I called up a report—and got his selection. Since Citrix has us using the same front end, the temporary table is the same for both of us. Since my boss made his selection and changed the temporary table between the time of my selection and when I called the report, my report was based on his selection.

In general, the temporary tables worked for loading the list boxes since there was only the smallest of time

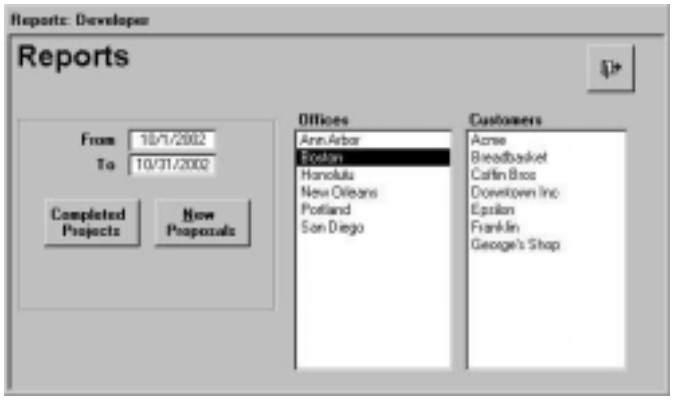



Figure 1. The complete Reports form.

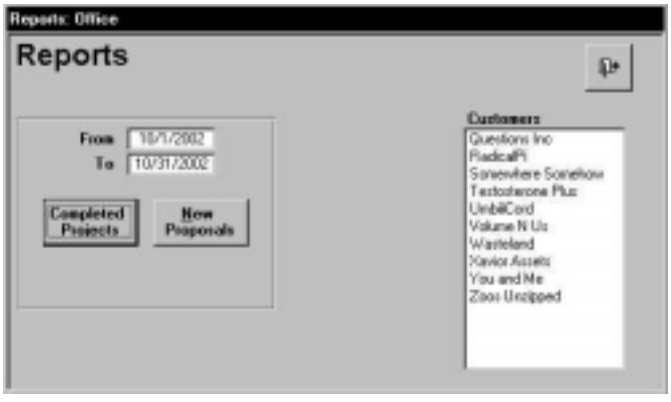



Figure 2. The Reports form for Office-level users.

intervals between the table updates and the list update. But the gap between selection and report generation was long enough to frequently cause problems.

A partial solution was to create multiple front ends, and have each office use a different one. Updating the new versions would then mean copying any new versions of the compiled front end to various office versions—easily done through an old-fashioned DOS batch file:

```
echo off
echo This will update the Tracker. Be sure that the
echo links to the back-end databases have been
echo changed to Citrix.
echo There should be no *.LDB files in the Tracker
echo directory.
dir Tracker*.ldb
echo         Ctrl-C to exit
```

# Introducing Citrix

Citrix provides an architecture that lets my company's people in various offices across the country connect to a remote server as if they were physically located there. They can log in to Citrix and see a desktop with shortcuts to an application that lets them enter and retrieve data. Both front-end and back-end databases that make up the application reside on a server that's local to my network. If I wanted, the front end could link to a back end anywhere on the intranet at the main office. The company purchases a set of Citrix licenses, which limits the number of simultaneous users. At this point, we've never come close to using all 30 licenses. While we have more people than that set up to use Citrix, the largest number of simultaneous users has been 12.

Citrix isn't a perfect solution. There's a right way and a wrong way to log out of Citrix, for instance. The Citrix desktop has the usual little X in the top right corner, so people assume that closes their Citrix session. So it does, but it doesn't log the user out properly (in Citrix parlance, it only does a disconnect). As a result, the user is still using a license (eventually their session will time out and release the license). The correct way out is through the Windows Start button and selecting "Logoff." If you implement a Citrix solution, make sure that everyone is aware of this.

Programming a database for Citrix use involves some different design decisions than those required for a networked system. First of all, to facilitate the speed of data transmission, your best choice is to design around a low-resolution screen with few colors. I generally program to a 600x800 resolution, though I develop on a system with higher resolution. More colors in your application means more data to transfer, which means slower data transmission.

In general, user actions take longer because of transmission times back and forth to the server. So, the more information that you can give to the user to let them maximize their UI interactions, the better. I use a color code for controls to signal to users what they can and can't do. On combo boxes, for instance, a pink background means you must select from the list, light green means you can enter something new (I based this on traffic lights). For text boxes, a yellow background means the data is locked, and light blue means that the user can double-click for special functions. However, a solid light pink isn't an option with 256 colors, so I use a pink dot pattern for my no-new-entries combo boxes (which unfortunately makes the box just a bit harder to read).

I use the standard system colors for all other purposes because people can customize their desktop colors in Citrix, just like they can on their own machines.

Upgrades must be handled differently in Citrix also. I can't upgrade the database while anybody is using it, so I needed a method to see who was logged in. On a networked system, you can look at the database's LDB file in Notepad and see who has opened the database. In Citrix, however, all the names in the LDB file are the same. I already had a login routine in the front end, which writes to tblUserLog in the back-end database, with time-in and time-out data for the user. Reading that table, I found that some people tended to minimize the database, forget that they'd already opened it, and then open it again, sometimes as many as three times. The potential problem here is write conflicts if they reopen the same record. I built in a login detector to check the table for previous logins, and blocked users from logging in twice.

This works fine until users don't log out because they've left Citrix the wrong way or have been disconnected. So I made a UserLog database that can tell me who is logged in at the present time. It also allows me to delete a user's login entry if they do get disconnected.

When it comes time to upgrade, I have a form that uses the timer to check the log every five seconds. If I can't find a time when no one is logged in, I may wait until the list is down to one or two users and give those people a call to ask them to log off. Then as I watch, I see their names disappear and can do a quick upgrade.

Another factor with the upgrade is that I need to relink the back-end tables each time. The Citrix environment isn't the one I want to develop on (and I don't want to work with real data during development). So I copy the back end to my local computer in order to have it up-to-date, and link my development database to that. When I've made my MDE file ready, I log on to Citrix. My Citrix desktop has a shortcut to my development directory on my computer, so I can drag the MDE file from there to the Citrix server. I have a routine that detects if it can find a specific table, and if it can't, brings up the link form so I can choose which back-end database to be linked to.

Occasionally this routine fails: Where it should have deleted an old link, it hasn't and the routine adds a second link. After an upgrade I always check that there's no such table as tblOffice1 and open at least one of the linked tables. Finally, I run my DOS batch file.

```
pause
echo on
copy BasicTracker.mde TrackerAnnArbor.mde
copy BasicTracker.mde TrackerBoston.mde
copy BasicTracker.mde TrackerDallas.mde
copy BasicTracker.mde TrackerHonolulu.mde
copy BasicTracker.mde TrackerNewOrleans.mde
copy BasicTracker.mde TrackerPortland.mde
copy BasicTracker.mde TrackerSanDiego.mde
pause
```

As you can see, I had the batch file list all the LDB files currently in the application directory on the Citrix server. This would tell me which copies of the database were currently in use. That would let me decide whether to upgrade just those databases that weren't in use, or wait until later when I could do everyone.

## Separating users

While this let me keep the company's offices separate, it wouldn't let me keep the users separate. I thought about using a single-entry table to hold just the current user's ID and user level, updating the table as the user logged in. But that wouldn't work, since the database would show only the last user logged in, even though earlier users were still busy in the database.

An invisible form, frmCurrent, called up by the system's login routine, was the solution to my problem and eventually to other problems too. While users share the tables in the database, each user gets his or her own copy of every form. The invisible form holds data in unbound controls, comes up as hidden, and never closes until the database is closed. Nobody knows it's there, so nobody fools with it! Here's the code that opens that form and updates the unbound controls on it:

```
DoCmd.OpenForm "frmCurrent",,,, acFormReadOnly, _
                                acHidden

 lngUserID = Me.cboName

 Forms!frmCurrent!txtUserID = lngUserID
 Forms!frmCurrent!txtUserLevelID = Me.cboName.Column(3)
 Forms!frmCurrent!txtOfficeID = Me.cboName.Column(4)

 DoCmd.Close acForm, "frmLogin"
 DoCmd.OpenForm "frmMainMenu"
```

To monitor the form, I can go to the Access menu, select Window | Unhide, and see what the form's unbound text boxes hold (see Figure 3). I tested the form by logging in and then having my boss log in—the scenario that killed the system before. On my screen, the form continued to reflect my information, and nothing my boss did would change it.

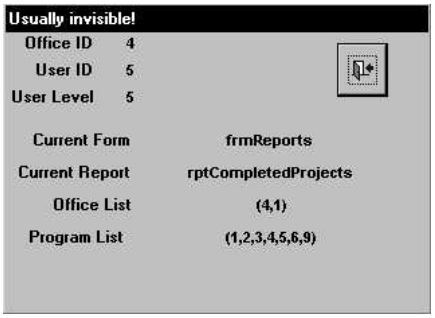



Figure 3. The invisible form revealed, holding a user's data.

I began by putting the data for the UserID and OfficeID in the form, but soon found that it was helpful to put the UserLevelID there as well. This field was used by the Menu command buttons to call up different versions of forms depending on the user's level. My other change was to add the UserID field to my temporary tables. This allowed me to keep track of which entries in the temporary table belonged to which user.

How did I use the invisible form, once I loaded it? One example is the Reports form, where users select which reports they want to use. The Reports form pulled the txtUserID text box's value from frmCurrent to use in managing the records in the tblOfficeSet. In this code, I delete the records from the temporary table that belong to the current user. The following query loads the table with data and updates data on my invisible form using a routine called ListBoxItems (you can find it in the sample download database). Finally, I make the list box on the Reports form invisible for some users and visible for others:

```
Select Case Forms!frmCurrent!txtUserLevelID
  Case 1, 2   ' single office
    Me.lstOffices.Visible = False
    DoCmd.RunSQL "Delete * from tblOfficeSet " & _
                 "WHERE CurrentUserID = " _
                 & Forms!frmCurrent!txtUserID
    DoCmd.OpenQuery "qryAppendOfficeSet"
    Forms!frmCurrent!txtOffices = _
                     ListBoxItems("Offices")
    Me.lstOffices.Visible = False
  Case 3, 4, 5
    Me.lstOffices.Visible = True
End Select
```

I also used the invisible form's settings to build criteria strings to use in selecting data for the reports. My general-purpose routine GetCriteria builds a string that limits the user's Program and Office selections:

```
Function GetCriteria
    Dim strC As String
    Dim strO As String

    strC = Forms!frmCurrent!txtPrograms
    If strC = "X" Then
        GetCriteria = "X"
        Exit Function
    End If
    strO = Forms!frmCurrent!txtOffices
    If strO = "X" Then
        strO = Forms!frmCurrent!txtOfficeID
    End If

    GetCriteria = "[ProgramID] in " & strC & _
                  " AND [OfficeID] in " & strO
End Function
```

If selections have been made in both list boxes, or in the Programs list box for the Office level, the function GetCriteria returns a string like this one:

```
[ProgramID] in (1,2,3,4,5,6) AND [OfficeID] in (3,4,6)
```

*Continues on page 19*

# Using Invisible Forms...

*Continued from page 6*

The parts in parentheses are pulled directly from the txtPrograms and txtOffices fields in frmCurrent.

Here's an example of GetCriteria in use when displaying a report:

```
strC = GetCriteria
If strC = "X" Then
    Exit Sub
End If

stDocName = "rptNewProposals"
DoCmd.OpenReport stDocName, acPreview, , strC
```

E-mail provides a step toward a paperless office, so I was asked to provide the ability for users to e-mail the report that they were currently viewing. Of course, with many users in the database, each could have a different report open while I needed to know which report the current user was viewing. I added another text box to frmCurrent to hold the name of the report on the screen and filled the text box in during the Report_Open event of each report:

```
Private Sub Report_Open(Cancel As Integer)
   Forms!frmCurrent!txtReport = "rptCompletedProjects"
End Sub
```

Not only does this allow my e-mail routine to determine the user's current report, but it also let me write another function to save the report as a snapshot, RTF file, or spreadsheet. I made these forms modal to keep the report from being closed during my processing. When the report was sent or saved, I stored the report's filter criteria in the invisible form where I could retrieve it to regenerate the report.

## A final problem

For quite a while after getting the reports together with the filters, I found that I still had a potential problem with two of my subreports. I based these subreports on two temporary tables, tblOfficeSet and tblCustomerSet. The subreports just showed the currently selected offices and customers. This would work only as long as someone else didn't get in on the same front end and change the selection. The report itself would show the correct data for the selection seen on the form, but the offices and customers would be those selected by the second person.

I tried putting the same filters in at various places, like this, in srpCustomers:

```
Private Sub Report_Open(Cancel As Integer)
    Me.Filter = "ProgramID in " & _
        Forms!frmCurrent!txtPrograms
    Me.FilterOn = True
End Sub
```

The value of txtCustomers would be a list of CustomerIDs, something like "(3,5,8,9,10,11,12)". The filter line that I generated would read "CustomerID in (3,5,8,9,10,11,12)". If I brought the subreport up by itself, everything worked. When I used the report as a part of another report, I'd get the runtime error, "The setting you entered isn't valid for this property." I needed a different solution.

Up to this point, the two tables, tblOfficeSet and tblCustomerSet, held just the ID and Customer or Office fields. In theory, they really only needed the user ID field; having the customer and office in there also kept me from having to include the tables tblOffice and tblCustomer in many queries. However, I had also added the UserID field in order to manage the records on a user-by-user basis. By using the UserID field, the subreports could be based on a query like this one:

```
SELECT tblProgramSet.*
FROM tblProgramSet
WHERE (((tblProgramSet.CurrentUserID)=
    [Forms]![frmCurrent]![txtUserID]));
```

And I reached this solution one day after I sent out a warning about the possibility for interference causing an occasional mismatch between the heading and the report!

The Citrix environment has worked well for several months now. I do my development work on my local

computer and upgrade the application periodically by logging onto Citrix, uploading the database, and running my DOS file. If the DOS file shows someone using one of the databases, I check who is logged in and call to ask them to leave the system. The company purchased 30 Citrix licenses, but actual usage has never come to even half that number of simultaneous users.

My solution isn't the only possible one. Instead of a completely separate form to hold user-specific data, I could have merely used fields in the Main Menu form (with their Visible property set to False so that the data was invisible). However, that design clutters the Form Design window and would have forced me to keep the Main Menu form open all the time. The invisible form holds all the data I want to put in it and provides a consistent reference throughout the application. Since starting in March of 2002, I've only had one instance of database corruption unrelated to the multiple users (that is to say, I could duplicate it on my standalone development system). So we've either been lucky, or Citrix has provided the stability that we need. ▲

DOWNLOAD INVISIBLE.ZIP at www.vb123.com/kb

Tobi K. Hoffman has been an Access developer for six years, an avid programmer for far longer (for both fun and profit), and currently works for Computer Science Corporation, a division of General Dynamics, in Needham, MA.

# February 2003 Downloads

INVISIBLE.ZIP—Tobi Hoffman has provided a version of the application that uses an invisible form to hold user-specific information. While the application was designed to be used in Citrix, the technique is useful wherever there's information to be shared among forms. (Access 2000)

- NAVIGATE2.ZIP—This file includes the code and reports that make up Chris Weber's navigation reporting system. Thereport demonstrates the techniques he uses to create a hierarchical report. (Access 2000 and Access 97)
- ADPTRIG2.ZIP—Russell Sinclair has provided SQL files for all of the triggers that he used in his application. These triggers are designed to work with the Northwind database that ships with SQL Server.
- AA0203.ZIP—Peter Vogel's sample database includes the tables and queries that demonstrate his SQL table for joining two tables. (Access 2000)

For access to all current and archive content and source code, log in at www.vb123.com/kb with your unique subscriber user name and password. For access to this issue's Downloads only, click on the "Source Code" button, select the file(s) you want from this issue, and enter the User name and Password at right when prompted.

User name **campus**

Password **dream**

Editor: Peter Vogel Contributing Editors: Mike Gunderloy, Danny J. Lesandrini, Garry Robinson, Russell Sinclair
CEO & Publisher: Mark Ragan Group
Publisher: Connie Austin Executive
Editor: Farion Grove Production
Editor: Andrew McMillan

## Questions?

Customer Service:
Phone: 800-493-4867 x.4209 or 312-960-4100
Fax: 312-960-4106

## Subscription rates

United States: One year (12 issues): $169; two years (24 issues): $287
Canada:* One year: $189; two years: $321
Other:* One year: $194; two years: $330

Single issue rate:
$20 ($22.50 in Canada; $25 outside North America)*

* Funds must be in U.S. currency.

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

Pinnacle
A division of Lawrence Ragan Communications, Inc.
316 N. Michigan Ave., Suite 400
Chicago, IL 60601

POSTMASTER: Send address changes to Lawrence Ragan Communications, Inc., 316 N. Michigan Ave., Suite 400, Chicago, IL 60601.