

Duplicate Data Entry for Access

Garry Robinson and Taha Kass-Hout



Redundant data is bad, but double data entry can be good. Garry Robinson shows you how and why.

ABOUT 12 years ago, when the mining company that I worked for ran all of its word processing on a Digital Vax mini-computer, there was a legal secretary who lost a 50-page legal document. After 15 minutes of looking, the system administrator had to explain to her that retrieval of the document was going to be difficult and wouldn't retrieve all of the work that she'd done that day. Funnily enough, she said this was okay and that she loved "a good type." She went back down the corridor and hammered out all 50 pages again in a few hours. Unlike programmers, there are many data entry and computer-trained people who enjoy using a keyboard. Likewise, you'll find that a large number of these people also wouldn't do a really good job of checking data entry against written reports. Yet there are many applications where it's absolutely critical that the data entered be verified.

This brings me to a challenge that co-author Taha Kass-Hout brought in. He asked for a form that would easily allow checking of data that was already entered into an Access table. This article demonstrates a user interface for data entry and verification that provides you with alternative approaches to managing the verification

process using Visual Basic. The scenario is that the user enters the data twice: once to enter the data and a second time to confirm that the data is correct.

Data entry user interfaces

When writing a user interface for a data entry specialist, there are some design criteria that you should factor into your interface. First, you can safely assume that the person doing the data entry won't use a mouse (pop it in the drawer during the testing phase). Your interface will consist of plain old text boxes, hot keys, and a maximum of five buttons. An example of this (really dull) form can be seen in [Figure 1](#). This form has a number of features that make it simple for the data entry person. The top menu offers no unnecessary choices that might excite a computer programmer (such as database compression or exporting) but would confuse a user.

The form footer consists of four buttons to enter, save, navigate, and close the form. This actually makes keyboard access to the buttons a problem, as you can't readily tab from the last field in the form to the bottom menus. To get around this, we assigned a hot key to each of those buttons that's flagged to the end user by an underline under the character that activates the button. To program this, open the form in design view and show the Caption property for your button (see [Figure 2](#)). Add an ampersand ("&") before the letter that you want to assign to your button. Now the button can be activated using Alt keys, and our form has become keyboard-friendly.

Data verification

When Taha and I were developing the software design for validating data, we came up with a solution that used table design information and control collections and just a

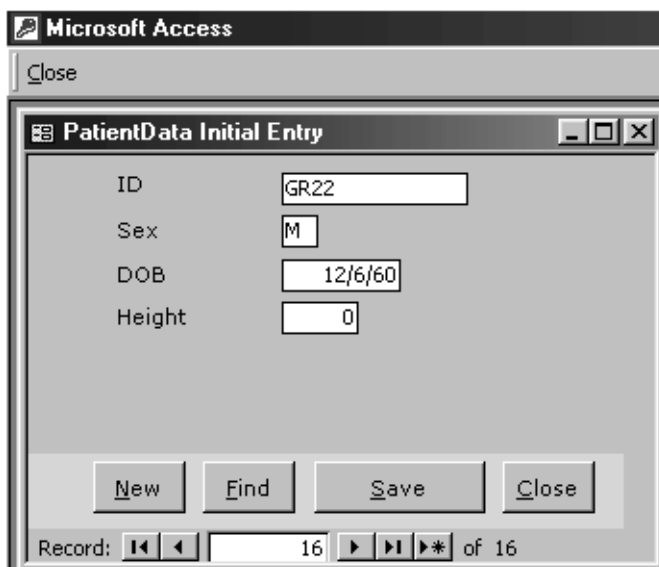


Figure 1. The very simple data entry interface used to input the initial data.

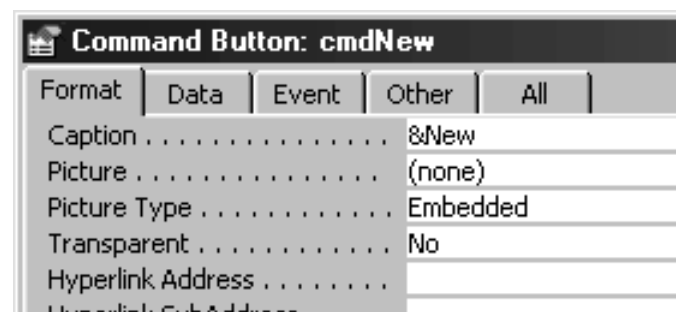


Figure 2. Making your command buttons accessible from the keyboard requires an ampersand in the caption.

single subroutine. For this article, though, we've split the solution into a basic form that most people should start with, and an advanced form that's appropriate to forms with many fields or systems where duplication is to be added to many forms. So on to the actual process that we came up with...

When the data entry form is opened, we open a recordset for the table that we are interested in:

```
Set rst = CurrentDb.OpenRecordset("tblData", _
dbOpenDynaset)
```

You can see this form in design view in [Figure 3](#). Importantly, the design of this form consists of controls for all of the fields in the table to be verified, with each control having the same name as a field in the table. There's also a second set of fields that have the same names but with a suffix of 2. If there's a field called Height, there will be controls called Height and Height2. The "2" fields handle differences and will only be used if a difference is found between the two entries.

The verification works by comparing the data that the user just entered with data that's already in the table. If there's no difference between the two controls for a field, an extra field in the table called "checked" is set to True, indicating that the row has been verified. If, on the other hand, there's a difference, then all of the fields where there are differences are displayed on the screen. The user can then make whatever changes are necessary and save the record—which is now assumed to be correct. In your company you may want the user to re-enter the data again or modify the existing data before saving the record.

Visual Basic used for verification

After the user enters the record and presses the verification button (with Alt-V), the following code does the verification check. The routine begins by checking to see whether the unique ID field in the table exists in the

form's recordset. The Chr function in this code is used to insert double quotes in the criteria string that does the search. If there's no entry that matches, the data entry person may save the current record because it indicates that the record is new to the table.

```
rst.FindFirst "ID = " & Chr(34) & Me!ID & Chr(34)
If rst.NoMatch Then
```

The most interesting part of the code comes when we actually have a match and wish to verify that the data is correct. The following code is from the simple version of our verification form. This sample code verifies a text field, a date field, and a numeric field. As you can see, the wonderful world of null values turns what should have been three simple tests into 30 lines of VBA code. The original data (which is now the current record in the recordset) is compared to the unbound field on the form into which the user enters data. Differences are displayed in the second field for the control:

```
flgDiff = False

If Nz(UCCase(Me!Sex), "") <> _
Nz(UCCase(rst!Sex), "") Then
  If IsNull(rst!Sex) Then
    Me!Sex2 = "Null"
  Else
    Me!Sex2 = UCCase(rst!Sex)
  End If
  flgDiff = True
End If

If CDate(Nz(Me!DOB, dateOne)) <> _
CDate(Nz(rst!DOB, dateOne)) Then
  If IsNull(rst!DOB) Then
    Me!DOB2 = "Null"
  Else
    Me!DOB2 = rst!DOB
  End If
  flgDiff = True
End If

If Val(Nz(Me!Height)) <> Nz(rst!Height) Then
  If IsNull(rst!Height) Then
    Me!Height2 = "Null"
  Else
    Me!Height2 = rst!Height
  End If
  flgDiff = True
End If
```

As you can see, the different data types require different code when performing the tests.

Advanced verification form

In the advanced form, the verification code uses the control collection of the form and the recordset's field collection to avoid hard-coding field names into the code. This code only works because of the naming convention that requires that text box names be the same as the recordset field names. Because the table will have fields that we don't want to verify, these will need to be ignored in the recordset by using function-wide constants like AutoKeyField, IdField, and CheckedField. The recordset field type property is used in this code to determine the data type of the field:

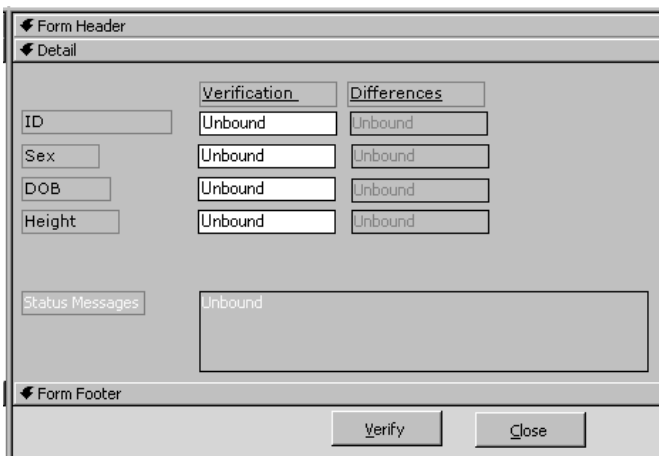


Figure 3. The data verification form is unbound and has fields for data and errors.

```

flgDiff = False
For Each fld In rst.Fields
  Select Case fld.Name
    Case AutoKeyField, IdField, CheckedField
      'Don't check these fields

    Case Else

      Select Case rst.Fields(fld.Name).Type
        Case dbText, dbMemo
          If Nz(UCCase(Me.Controls(fld.Name)), "") <> _
            Nz(UCCase(rst.Fields(fld.Name)), "") Then
            If IsNull(rst.Fields(fld.Name)) Then
              Me.Controls(fld.Name & "2") = "Null"
            Else
              Me.Controls(fld.Name & "2") = _
                UCCase(rst.Fields(fld.Name))
            End If
            flgDiff = True
          End If
        End Select
      End Select
    Next fld

  Etc etc...

  End Select
End Select
Next fld

```

After the fields are all verified, the advanced form opens a function that updates the data from the unbound form fields back to the recordset. The code for this function is quite concise:

```

rst.Edit
For Each fld In rst.Fields

  Select Case fld.Name
    Case AutoKeyField, CheckedField, IdField
      ' Skip the auto key field and checked fields

    Case Else
      rst(fld.Name) = Me.Controls(fld.Name)
    End Select

  Next fld

  rst(CheckedField) = True
  rst.Update

```

The download database

Double data entry is a method to minimize data entry errors. Data is entered twice; the program then compares the two values for each field in real time and identifies values that don't match. Discrepant entries are checked on the original data forms and corrected. Double data entry identifies data entry errors but at the cost of

Further Reading and Resources

- Is this the first time that an Internet movie has been generated for *Smart Access*? The following address allows you to see the duplicated entry software running in a special movie form: http://capastatistic.com/E-Projects/Components/DoubleUp/Sample/DoubleEntry_viewlet.html
- *Microsoft Windows User Experience 1999: Official Guidelines for User Interface Developers and Designers*, "Access Keys," page 60 (from Microsoft Press).

doubling the time or the personnel required for data entry.

Future program development includes the following:

- Rechecking or re-entering a random portion of the data. If the error rate is acceptably low, additional data editing is unlikely to be worth the effort and cost.
- Logging the discrepant entries.
- Providing a one-to-many double data entry advanced form.

Included in the Source Code file that's available at www.smartaccessnewsletter.com are examples of the software in Access 97, 2000, and 2002. When you open the software database, there are three forms. FirstEntry is a simple form for data entry and has very little in noteworthy coding. DoubleEntryBasic is the simpler of the two samples of the double entry code. The code in this form is specific to the table and is the template that you're most likely to use. The third form is called DoubleEntryAdvanced. This form has code that's been generalized to minimize the customized code behind the form. If you have a lot of fields in the table or think you might be adding new fields at a later stage, this format may be for you. There's also a basic report to show what's been entered and checked. ▲

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

Garry Robinson works for GR-FX Pty Limited, a company based in Sydney, Australia. If you want to keep up-to-date with his latest postings on Access issues, visit his company's Web site at www.gr-fx.com or sign up for his Access e-mail newsletter by sending a blank e-mail to tips@gr-fx.com. He assists other programmers around the world through the Internet, which is where he ran into Taha. Taha A. Kass-Hout, MD, MS, established capastatistic.com, a company based in Houston, TX. The company specializes in e-surveys and statistical analysis.

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

eNewsletters

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